13th International Software & Internet

Quality Week

San Francisco, California

May 30--June 2, 2000

New Century New Beginnings

Conference Proceedings

eValid

Your e-Business Partner

13th International Software / Internet Quality Week
(QW2000)
Conference Theme:
New Century, New Beginnings

San Francisco, California USA
30 May 2000 - 2 June 2000

TECHNICAL PROGRAM

NOTE: The program selections, speaker biographies, and presentation abstracts may be incomplete in some instances; this material is being updated constantly. All material presented is based on the best information available and may be subject to change. Updated 20 March 2000.

QUICK ACCESS TO THE FOUR DAY PROGRAM AT QW2000	
Pre-Conference Tutorials	Tuesday 30 May 2000
Conference Day 1	Wednesday 31 May 2000
Conference Day 2	Thursday 1 June 2000
Conference Day 3	Friday 2 June 2000
Post Conference Workshops	Friday 2 June 2000
Vendor Demonstration Sessions	Wednesday 31 May 2000 Thursday 1 June 2000

The QW2000 program is linked to speaker biographies and presentation abstracts.
Click on a ***Speaker*** or on a ***Title*** to read the Presentation Summary and Author Biography.
Paper descriptions will appear in a separate popup window.

REGISTER FOR QW2000

[BACK TO TOP]

T	Tuesday 30 May 2000 PRE-CONFERENCE TUTORIALS

T	Tuesday 30 May 2000 PRE-CONFERENCE TUTORIALS						
8:30 - 12:00	**Tutorial A1** Johanna Rothman (Rothman Consulting Group) [USA] *Life as a New Test Manager*	**Tutorial B1** Norman Schneidewind (Naval Postgraduate School) [USA] *A Roadmap to Distributed Client Server Software Reliability Engineering*	**Tutorial C1** Michael Deck (Cleanroom Software Engineering, Inc.) [USA] *Requirements Analysis Using Formal Methods*	**Tutorial D1** Bill Deibler (SSQC) [USA] *Making the CMM Work: Streamlining the CMM for Small Projects and Organizations*	**Tutorial E1** Ross Collard (Collard & Company) [USA] *Test Planning*	**Tutorial F1** G. Bazzana & E. Fagnoni (ONION s.r.l.) [Italy] *Testing Web-based Applications: Techniques for Conformance Testing*	**Tutorial G1** Ed Kit (Software Development Technologies) *Software Testing in the Real World*
12:00 - 1:30	TUTORIAL DAY LUNCH AND NETWORKING						
1:30 - 5:00	**Tutorial A2** Mr. Robert Binder (RBSC Corporation) *How to Write A Test Design Pattern*	**Tutorial B2** John Musa (Consultant) *Developing More Reliable Software Faster and Cheaper*	**Tutorial C2** Tom Gilb (Results, Inc.) *Requirements Engineering for Software Developers and Testers*	**Tutorial D2** Rob Baarda & Tim Koomen (IQUIP Informatica BV) *Stepwise Improvement of the Testing Process using TPI(tm)*	**Tutorial E2** Linda Rosenberg, Ruth Stapko, & Albert Gallo (NASA GSFC) *Risk-Based Object Oriented Testing*	**Tutorial F2** Adrian Cowderoy (MMHQ) *Cool Q - Quality Improvement for Multi-Disciplinary Tasks in Website Development*	**Tutorial G2** Chris Loosley & Eric Siegel (Keynote Systems) *Web Application Performance*
5:00 - 6:00	Welcome Networking Reception						

[BACK TO TOP]

8:30 - 12:00	**Tutorial A1** Johanna Rothman (Rothman Consulting Group) [USA] *Life as a New Test Manager*	**Tutorial B1** Norman Schneidewind (Naval Postgraduate School) [USA] *A Roadmap to Distributed Client Server Software Reliability Engineering*	**Tutorial C1** Michael Deck (Cleanroom Software Engineering, Inc.) [USA] *Requirements Analysis Using Formal Methods*	**Tutorial D1** Bill Deibler (SSQC) [USA] *Making the CMM Work: Streamlining the CMM for Small Projects and Organizations*	**Tutorial E1** Ross Collard (Collard & Company) [USA] *Test Planning*	**Tutorial F1** G. Bazzana & E. Fagnoni (ONION s.r.l.) [Italy] *Testing Web-based Applications: Techniques for Conformance Testing*	**Tutorial G1** Ed Kit (Software Development Technologies) *Software Testing in the Real World*
12:00 - 1:30	TUTORIAL DAY LUNCH AND NETWORKING						
1:30 - 5:00	**Tutorial A2** Mr. Robert Binder (RBSC Corporation) *How to Write A Test Design Pattern*	**Tutorial B2** John Musa (Consultant) *Developing More Reliable Software Faster and Cheaper*	**Tutorial C2** Tom Gilb (Results, Inc.) *Requirements Engineering for Software Developers and Testers*	**Tutorial D2** Rob Baarda & Tim Koomen (IQUIP Informatica BV) *Stepwise Improvement of the Testing Process using TPI(tm)*	**Tutorial E2** Linda Rosenberg, Ruth Stapko, & Albert Gallo (NASA GSFC) *Risk-Based Object Oriented Testing*	**Tutorial F2** Adrian Cowderoy (MMHQ) *Cool Q - Quality Improvement for Multi Disciplinary Tasks in Website Development*	**Tutorial G2** Chris Loosley & Eric Siegel (Keynote Systems) *Web Application Performance*
5:00 - 6:00	Welcome Networking Reception						

[BACK TO TOP]

Wednesday 31 May 2000 CONFERENCE DAY #1	
QW2000 Exhibition: 10:00 AM to 6:00 PM	
1	**PLENARY SESSION**
	Plenary Session Introduction: Edward Miller (Software Research, Inc.)

Wednesday 31 May 2000
CONFERENCE DAY #1

QW2000 Exhibition: 10:00 AM to 6:00 PM

1 8:30 - 10:00	**PLENARY SESSION**					
	Plenary Session Introduction: Edward Miller (Software Research, Inc.)					
	Keynote 1P1: Stu Feldman (IBM) *Internet and E-Commerce: Issues and Answers*					
	Keynote 1P2: Bill Gilmore (Intel Corporation) *The Intel Software Corporate Quality Network*					
10:00 - 10:30	REFRESHMENTS IN EXHIBIT HALL					
2 10:30 - 12:00	**Vendor Technical Track**	**Technology Track** OO Automation	**Applications Track** Conformance & Internationalization	**Internet Track** E-Commerce Experience	**Management Track** Managing Testing	**QuickStart Track**
	Session 2V1 Ms. Melissa Borza (Computer Associates) *Enterprise Change and Configuration Management*	**Paper 2T1** Michael Silverstein (SilverMark, Inc.) *Automating Testing of Object-Oriented Components Using Intelligent Test Artifacts*	**Paper 2A1** James Andrews (The Open Group) *Automated Conformance Testing for IT & T Product Certification*	**Paper 2W1** Ted Fuller (Agency.com) *Notes From The Front Lines: How to Test Anything and Everything on a Web Site*	**Paper 2M1** Joel Fleiss (VeriTest) *The ABCs of Managing a Software Testing Project*	**Session 2Q** James Bach (Satisfice, Inc.) *The Heuristic Approach to Testing*

8:30	**Keynote 1P1:** Stu Feldman (IBM) *Internet and E-Commerce: Issues and Answers*					
	Keynote 1P2: Bill Gilmore (Intel Corporation) *The Intel Software Corporate Quality Network*					
10:00 10:00 10:30	REFRESHMENTS IN EXHIBIT HALL					
2 10:30 - 12:00	**Vendor Technical Track**	**Technology Track** OO Automation	**Applications Track** Conformance & Internationalization	**Internet Track** E-Commerce Experience	**Management Track** Managing Testing	**QuickStart Track**
	Session 2V1 Ms. Melissa Borza (Computer Associates) *Enterprise Change and Configuration Management*	**Paper 2T1** Michael Silverstein (SilverMark, Inc.) *Automating Testing of Object-Oriented Components Using Intelligent Test Artifacts*	**Paper 2A1** James Andrews (The Open Group) *Automated Conformance Testing for IT & T Product Certification*	**Paper 2W1** Ted Fuller (Agency.com) *Notes From The Front Lines: How to Test Anything and Everything on a Web Site*	**Paper 2M1** Joel Fleiss (VeriTest) *The ABCs of Managing a Software Testing Project*	**Session 2Q** James Bach (Satisfice, Inc.) *The Heuristic Approach to Testing*
	Session 2V2 Mr. Mark VanName (ZD Labs) *Benchmarking the Future*	**Paper 2T2** James Elder & Ricard Roma-i-Dalfo (Microsoft) *Object Based Machine Automation (OSIRIS Project)*	**Paper 2A2** Juichi Takahashi (Florida Institute of Technology) *Is Special Software Testing Necessary Before Releasing Products to an International Market?*	**Paper 2W2** Steven Porter (API / Independent) *From Web Site to Web App: Ensuring Quality in a Complex Environment*	**Paper 2M2** Johanna Rothman (Rothman Consulting Group) *The Influential Test Manager*	
12:00 - 1:30	CONFERENCE LUNCH AND NETWORKING IN EXHIBIT HALL					

	Vendor Technical Track	Technology Track Hi-Tech Testing	Applications Track Using Technology	Internet Track WebSite Performance I	Management Track Process Feedback	QuickStart Track
3 1:30 - 3:00	**Session 3V1** Mr. Michael R. Perdue (Software Development Technologies) *Automating Test Design and Inspection*	**Paper 3T1** Robert T. Bauer & Russell F. Ingram (Levetate Design Systems) *Building a Parallel Test Environment*	**Paper 3A1** David W. Carman (Telcordia Technologies) *Measuring Test Effectiveness: The Use and Misuse of Test Coverage*	**Paper 3W1** Alberto Savoia (Velogic Inc.) *The Science of Website Load Testing*	**Paper 3M1** Phil Lones (Lucent Technologies) *A Practical Approach to Testing Software in an Evolutionary Delivery Environment*	**Session 3Q** Robert V. Binder (RBSC Corporation) *Best Practices for Object-Oriented Systems*
	Session 3V2 Mr. Christian Hote (PolySpace) *Runtime Error Detection*	**Paper 3T2** Robert Oshana (ObjectSpace, Inc.) *Performance Engineering of an Embedded System Application*	**Paper 3A2** Linda Hayes (WorkSoft) *Table Driven Testing: Making Automation Accessible*	**Paper 3W2** B. M. Subraya & S. V. Subrahmanya (Infosys) *Performance Testing: A Methodical Approach to E-Commerce Applications*	**Paper 3M2** Cem Kaner (Attorney At Law) *Yes, But What Are We Measuring?*	
3:00 - 3:30	REFRESHMENTS IN EXHIBIT HALL					
4 3:30	**Vendor Technical Track**	**Technology Track** Test Automation	**Applications Track** Production Modes	**Internet Track** WebSite Performance II	**Management Track** Intellectual Property	**QuickStart Track**

- 5:00	**Session 4V1** Mr. Steve Nemzer (VeriTest) *Zero Defect or Simship? The Global Testing Challenge*	**Paper 4T1** Rainer Stetter (Software Factory GmbH) *Software Quality for Embedded Systems*	**Paper 4A1** William E. Lorensen & James Miller (GE Corporate Research & Development) *Visualization Toolkit Extreme Testing: A Production Release Every Day*	**Paper 4W1** Pat Garverick (Landmark Systems Corporation) *Testing the Performance Impact of a Web-based Application*	**Panel 4P** Doug Whitney & Pete Nordquist (Intel Corporation, Home Products Group) *Protecting Intellectual Property in an Open Source World* *PANEL SESSION*	**Session 4Q** Tom Gilb (Result Planning Limited) *Pitiful and Powerful Measures of Software Metrics*
	Session 4V2 Mr. John Bowman (Compuware Corporation) *Test Process and Test Automation - The Pathway To Success*	**Paper 4T2** Mark Blackburn & Joseph Fontaine (Software Productivity Consortium) *Application of the Test Automation Framework for Model Analysis and Test Generation*	**Paper 4A2** Kevin VanFlandern (Microsoft, Inc.) *Benchmarking Large Windows Based Applications*	**Paper 4W2** Steven Rabin (Interworld Corp.) *eCommerce Performance Management Lifecycle -- Benchmarking, Methodology and Criteria*		
5:00 - 6:00	EXPO RECEPTION Drinks and hors d'oeuvres are served in the Expo Hall.					
7:15	QW2000 SPECIAL EVENT: Baseball at the New Pac Bell Park SF Giants vs. Philadelphia Phillies Ticket Preference Procedure					

[BACK TO TOP]

Thursday 1 June 2000 CONFERENCE DAY #2	
QW2000 Exhibition: 10:00 AM to 6:00 PM	
5	**PLENARY SESSION**

[BACK TO TOP]

Thursday 1 June 2000
CONFERENCE DAY #2

QW2000 Exhibition: 10:00 AM to 6:00 PM

PLENARY SESSION

Plenary Session Introduction:

Edward Miller

(Software Research, Inc.)

5 8:30 - 10:00	**Keynote 5P1:** Leon J. Osterweil (Unviersity of Massachusetts) *Determining the Quality of Electronic Commerce Processes (5P1)*					
	Keynote 5P2: Mr. Rainer Pirker (IBM / Austria) *The Need for Quality -- e-business Performance Testing (5P2)*					
10:00 - 10:30	REFRESHMENTS IN EXHIBIT HALL					
6 10:30 - 12:00	**Vendor Technical Track**	**Technology Track** Advanced Techniques I	**Applications Track** Software Components	**Internet Track** Internet Time	**Management Track** Practical Testing	**QuickStart Track**
	Session 6V1 Mr. Ted Burnett (Watchfire) *Why Web QA Is Essential in all Phases of the Development Process*	**Paper 6T1** Alan Myrvold (Entrust Technologies Limited) *Feeling Tcl-ish? Applying Tcl to Real Test Tasks*	**Paper 6A1** Jean Hartmann & Claudio Imoberdorf (Siemens Corporate Research) *Functional Testing of Distributed, Component-Based Software*	**Paper 6W1** Anand Sundaram (RSW Software, Inc.) *Managing E-Business Quality in Internet Time*	**Paper 6M1** Lisa Crispin (TRIP.com) *Guerilla Tool Selection*	**Session 6Q** Tobias G. Mayer (eValid, Inc.) *WebSite Testing*
	Session 6V2 Michael Aivazis (ParaSoft Corporation) *Methods for Effective Unit Testing*	**Paper 6T2** Elisabeth Hendrickson (Aveo Inc.) *Quality in an ASP Environment*	**Paper 6A2** Jerry Gao, Kamal Gupta & Shalini Gupta (San Jose State University) *Design for Testability of Software Components*	**Paper 6W2** Lisa Crispin (TRIP.com) *Stranger in a Strange Land -- Bringing Quality Assurance to a Web Startup*	**Paper 6M2** Brian Lawrence & Johanna Rothman (Coyote Valley Software / Rothman Consulting, Inc.) *Testing in the Dark*	
12:00 - 1:30	CONFERENCE LUNCH AND NETWORKING IN EXHIBIT HALL					

	Vendor Technical Track	Technology Track Java	Applications Track COTS	Internet Track WebSite Quality	Management Track Defect Tracking	QuickStart Track
7 1:30 - 3:00	**Session 7V1** Ms. Greta Cohen and Mr. Jim Baack (Superior IS) *Outsourced Testing for the Web*	**Paper 7T1** Andreas Spillner & Ulrich Breymann (Hochschule Bremen) *Semantic Differences Between C++ and Java: Consequences for the Review and Test Process*	**Paper 7A1** Yingxu Wang (Centre for Software Engineering) *A Practical New Approach to COTS Testing*	**Paper 7W1** Jeanette Folkes & Bert Lamar (Ogilvy Interactive) *The Challenges of Web Testing*	**Paper 7M1** Patrick Copeland (Microsoft) *Redesigning a Testing Organization for Delivery to the Web*	**Session 7Q** Brian Marick, James Bach & Cem Kaner [USA] *Evaluating Test Suites*
	Session 7V2 Mr. Bruce Katz (Rational) *Test Early, Test Often: Applying Today's Best Testing Practices and Techniques to the Full Software Development Lifecycle*	**Paper 7T2** Charles White (Segue Software, Inc.) *Functional Testing of CORBA based Systems in Java*	**Paper 7A2** Scott Trappe (Reasoning Inc.) *Find the Defects that Traditional Testing Misses with Automated Software Inspection Services*	**Paper 7W2** Andrea MacIntosh & Wolfgang Strigel (QA Labs Inc.) *The Living Creature - Testing Web Applications*	**Paper 7M2** Rex Black (Rex Black Consulting Services, Inc.) *The Fine Art of Writing a Good Bug Report*	
3:00 - 3:30	REFRESHMENTS IN EXHIBIT HALL					
	Vendor Technical Track	Technology Track Advanced Techniques II	Applications Track Specifications	Internet Track Commercial Quality	Management Track Certification	QuickStart Track

8 3:30 - 5:00	**Session 8V1** Mr. Jim Bampos (Vanteon) *Testing E-Commerce Applications*	**Paper 8T1** Atif M. Memon, Martha E. Pollack, & Mary Lou Soffa (University of Pittsburgh) [USA] *A Planning-Based Approach to GUI Testing*	**Paper 8A1** John Musa (Consultant) *A Good Idea! But How Do We Get People To Use It?*	**Paper 8W1** Adrian Cowderoy (MMHQ) *Technical Quality is Just the Start -- The Real Battle is Commercial Quality*	**Paper 8M1** D. J. Law (QWest Communications) *Certification Programs for Software Quality and Test Professionals*	**Session 8Q** Thomas Drake (ICCI) *Testing Network Based Software Systems -- The Future Frontier*
	Session 8V2 Mr. Mark Myers (Teradyne SST) *A Model-Based Technique for Test Program Creation*	**Paper 8T2** W. T. Tsai, et. al. (Arizona State University) *Automatic Test Case Generation for GUI Navigation (SB1)*	**Paper 8A2** Giuseppe Lami, Stefania Gnesi, Mario Fusani & Fabrizio Fabbrini (Istituto di Elaborazione dell'Informazione) [Italy] *Quality Evaluation of Software Requirements Specifications*	**Paper 8W2** Steven Watson (CNET Inc.) *Quality Assurance Challenges in the Internet Industry*	**Paper 8M2** Marc Zasada (VeriTest) *What Does Application Certification Mean in the Software Industry?*	
5:00 - 6:30	Nick Borelli, Panel 8P Chair Special Panel Session: *ASK THE QUALITY EXPERTS!* Stump the Quality Experts If You Can! Post Your Questions LIVE on the Web! QW2000 Advisory Board Members Will Answer All! (In Cooperation With Microsoft Corporation)					

[BACK TO TOP]

3:30 - 5:00	**Session 8V1** Mr. Jim Bampos (Vanteon) *Testing E-Commerce Applications*	**Paper 8T1** Atif M. Memon, Martha E. Pollack, & Mary Lou Soffa (University of Pittsburgh) [USA] *A Planning-Based Approach to GUI Testing*	**Paper 8A1** John Musa (Consultant) *A Good Idea! But How Do We Get People To Use It?*	**Paper 8W1** Adrian Cowderoy (MMHQ) *Technical Quality is Just the Start -- The Real Battle is Commercial Quality*	**Paper 8M1** D. J. Law (QWest Communications) *Certification Programs for Software Quality and Test Professionals*	**Session 8Q** Thomas A. Drake (CRTI) *Testing Network Based Software Systems -- The Future Frontier*
	Session 8V2 Mr. Mark Myers (Teradyne SST) *A Model-Based Technique for Test Program Creation*	**Paper 8T2** W. T. Tsai (Arizona State Univers.) *Auto. Test Case Generati-on for GUI Navigation*	**Paper 8A2** Giuseppe Lami, Stefania Gnesi, Mario Fusani & Fabrizio Fabbrini (Istituto di Elaborazione dell'Informazione) [Italy] *Quality Evaluation of Software Requirements Specifications*	**Paper 8W2** Steven Watson (CNET Inc.) *Quality Assurance Challenges in the Internet Industry*	**Paper 8M2** Marc Zasada (VeriTest) *What Does Application Certification Mean in the Software Industry?*	
5:00 - 6:30	Nick Borelli, Panel 8P Chair Special Panel Session: *ASK THE QUALITY EXPERTS!* Stump the Quality Experts If You Can! Post Your Questions LIVE on the Web! QW2000 Advisory Board Members Will Answer All! (In Cooperation With Microsoft Corporation)					

[BACK TO TOP]

Friday 2 June 2000 CONFERENCE DAY #3						
9 8:30	**Special General Panel Session**	**Technology Track** UML Methods	**Applications Track** Risk	**Internet Track** Link Checking	**Management Track** Process Innovations	**QuickStart Track**

Friday 2 June 2000
CONFERENCE DAY #3

	Special General Panel Session	Technology Track UML Methods	Applications Track Risk	Internet Track Link Checking	Management Track Process Innovations	QuickStart Track
9 8:30 - 10:00	**Panel 9P** Mr. Brian Lawrence (Panel Chair) Panelists: Ms. Johanna Rothman Esther Derby Maureen O'Hara. *How Can I Tell When My Project's in Trouble?*	**Paper 9T1** Sam Guckenheimer (Rational Software Corporation) *Enabling Testable Architectures with UML*	**Paper 9A1** Rob Baarda (IQUIP Informatica BV) *Risk Based Test Strategy*	**Paper 9W1** Wen-Kui Chang & Shing-Kai Hon (Tunghai University) *A Systematic Framework for Ensuring Link Validity under Web Browsing Environments*	**Paper 9M1** Hong Guo, Graham King, Margaret Ross & Geoffe Stable (Southampton Institute) *Using BOOTSTRAP to Improve the Management of Software Process in a Virtual Software Organization?*	**Session 9Q** Otto Vinter (DELTA Danish Electronics, Light & Acoustics) *Experience-Based Approaches to Process Improvement*
		Paper 9T2 Tim Szymanski (Advanced Software Technologies, Inc.) *Quality Starts with Requirements: How the UML Can Help*	**Paper 9A2** Jerrold Landau (IBM Canada) *An Overview of Testing Methodology and Experience at IBM Corepoint Banking Solutions*	**Paper 9W2** Michael Weider (Watchfire) *The Web Application Process: Development & Testing*	**Paper 9M2** Richard Kasperowski (Altisimo Computing) *Opportunistic Software Quality*	
10:00 - 10:30	REFRESHMENTS					
	PLENARY SESSION					

10 10:30 - 12:30	**Plenary Session Introduction:** Edward Miller (Software Research, Inc.)
	Keynote 10P1: Marcelo Dalceggio (Banco Rio de la Plata SA) [Argentina] *Automated Software Inspection Process* QWE'99 Best Presentation
	Keynote 10P2: Sanjay Jejurikar Director of Test for Windows 2000 (Microsoft) *Engineering Process of Windows 2000*
	Keynote 10P3: Gene Spafford Director, CERIAS (Purdue University) *Information Security Requires Assurance*
12:30 - 2:00	CONFERENCE LUNCH AND AWARDS PRESENTATION

[BACK TO TOP]

Friday 2 June 2000
POST-CONFERENCE WORKSHOPS

W 2:00 - 5:00	**Technology Workshop W1** Douglas Hoffmann (Software Quality Methods LLC) [USA] *Oracle Strategies for Automated Testing*	**Applications Workshop W2** Cem Kaner (Attorney at Law) *Bug Advocacy*	**Internet Workshop W3** Edward Miller (Software Research, Inc.) *Achieving WebSite Quality*	**Management Workshop W4** Robert Sabourin (Purkinje Inc.) *The Effective SQA Manager - Getting Things Done*

[BACK TO TOP]

SB	Standby Presentations		
	Yongzhong Tu (University of Minnesota) *An Approach to Testing Component-Based Software (SB2)*	Ron Silacci (Lucent Technologies Inc.) *Principles of Multi-System Integration (SB3)*	Giora Ben-Yaacov (Synopsys Inc.) *Effective Software Testing at a Leading Electronic Design Automation (EDA) Company (SB4)*

4:30		4:30	
5:00		5:00	
5:30		5:30	

[BACK TO TOP]

Software Research, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Phone: +1 (415) 550-3020
TollFree: +1 (800) 942-SOFT [USA Only]
Fax: +1 (415) 550-3030
E-Mail: info@soft.com

QW2K Tutorial A1

Ms. Johanna Rothman (Rothman Consulting Group)

Life as a New Test Manager

BACK TO QW2000 PROGRAM

Key Points

- Using your project's definition of quality to help define when the product is ready to ship
- How to define measurable criteria
- When and what to negotiate with the project team
- How to assess the product against the criteria

Presentation Abstract

Congratulations! You've just been promoted to test manager. Now what do you do? This tutorial walks you through some of the requirements for succeeding as a test manager: creating a test group and bringing new people into the organization; organizing and managing the testing; working with the project manager and the rest of the project team; deciding how, when, and what to invest in your test infrastructure.

About the Author

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people.

Johanna publishes "Reflections", an acclaimed quarterly newsletter about managing product development. Johanna's handbook, "Hiring Technical People: A Guide to Hiring the Right People for the Job," has proved a boon to perplexed managers, as have her articles in Software Development, Cutter IT, IEEE Computer, Software Testing and Quality Engineering, and IEEE Software.

Johanna is the founder and principal of Rothman Consulting Group, Inc., and is a member of the clinical faculty of The Gordon Institute at Tufts University, a practical management degree program for engineers.

BACK TO QW2000 PROGRAM

QW2K Seminar B1

Dr. Norman Schneidewind
(Naval Postgraduate School)

A Roadmap to Distributed Client-Server Software Reliability Engineering

BACK TO QW2000 PROGRAM

Key Points

- Help practitioners implement or improve a software reliability program in their organizations, using a step-by-step approach based on an enhanced version of the ANSI/AIAA Recommended Practice for Software Reliability.
- Use case studies from the NASA Space Shuttle and the United States Marine Corps logistical systems.
- Emphasize software reliability for distributed systems.

Abstract...

Background

In general, existing software reliability prediction methodologies do not address the characteristics of distributed systems, including client-server systems. This is an amazing situation given the importance of distributed systems in contemporary society. The reason for this situation is the rapid technological development of distributed systems and networks, most notably the explosive growth of the Internet. This rapid growth has made it difficult for the software reliability community to respond to the challenge of the new technology. This tutorial was developed to address this deficiency.

Objective

The objective of this tutorial is to help practitioners implement or improve a software reliability program in their organizations, using a step-by-step approach based on an enhanced version of the ANSI/AIAA Recommended Practice for Software Reliability and case studies from the NASA Space Shuttle and the United States Marine Corps logistical systems. Modeling methods, prediction techniques, and defect analysis for distributed systems will be emphasized

Scope

In broad terms, implementing a software reliability program is a two-phased process. It consists of (1) identifying the reliability goals and (2) testing the software to see whether it conforms to the goals. The reliability goals can be ideal (e.g., zero defects) but should have some basis in reality based on tradeoffs between reliability and cost. The testing phase is more complex because it involves collecting raw defect data and using it for assessment and prediction.

These phases require a number of steps for their implementation. The following steps will be presented as a process for an organization to develop a software reliability program. These steps

provide a structured approach to the software reliability process. Each step will be discussed to provide a good understanding of the entire software reliability process. The software reliability steps and outline of the tutorial are as follows:

- State the Reliability Requirement
- Establish a Measurement Framework
- Identify the Operational Profile
- Establish Problem Severity Levels
- Collect the Data
- Estimate Model Parameters
- Select the Optimal Set of Failure Data
- Test the Software
- Make Reliability Predictions
- Adapt the Process For Distributed System Predictions
- Validate the Model
- Assess the Risks of Deploying the Software
- Make Reliability Decisions
- Use Software Reliability Tools

- The new SMERFS3 Windows-based system and software engineering reliability tool will be used to illustrate software reliability assessment and prediction.

About the Author

Dr. Norman Schneidewind is Professor of Information Sciences and Director of the Software Metrics Research Center at the Naval Postgraduate School. He is the developer of the Schneidewind software reliability model, recommended by the American National Standards Institute of Aeronautics and Astronautics Recommended Practice for Software Reliability.

BACK TO QW2000 PROGRAM

QW2K Tutorial C1

Mr. Michael Deck
(Cleanroom Software Engineering, Inc.)

Requirements Analysis Using Formal Methods

BACK TO QW2000 PROGRAM

Key Points

- Participants will be introduced to formal notations and structures for reducing the ambiguity inherent in natural language specification. For developers, this will allow a thorough understanding of potential problems prior to development and permits inspections that focus on correctness evaluation. For testers, this permits understanding of what constitutes a testing "failure" and will also supports the construction of better testing models.
- Participants will learn how to use functional models for isolating and then removing incompleteness and nondeterminism from specifications.
- Participants will learn techniques for analyzing the product's input domain especially domain components that may be overlooked e.g. system environment influences.
- Participants will see how formal models can be applied gradually and informally.

Presentation Abstract

This half-day tutorial will show testers and developers how they can use formal methods to analyze and improve software requirements. One of the significant problems of development and testing is having to work from inadequate, incomplete, ambiguous specification documents. This tutorial will introduce informal ways engineers can "think formally" about specification documents to reveal domain gaps, incompleteness, and other problem spots before they have too great an impact on the testing process. We will use a combination of the functional/denotational and model-based approaches.

About the Author

Michael Deck is an internationally-recognized expert in Cleanroom software engineering practices. His consulting company, Cleanroom Software Engineering, Inc., specializes in training project teams to tailor and use Cleanroom practices to solve real-life software process problems.

The company also practices what it teaches, applying Cleanroom practices to contract software-development projects. From 1982 to 1993 he was a member of the IBM Cleanroom Software Technology Center, where he worked closely with the inventors of the Cleanroom approach.

He has a BA in Mathematics from Kalamazoo College and an MS in Computer Science from the University of Maryland, College Park. His current research interests include application of Cleanroom to object-oriented development, real-time and embedded software, and highly reliable systems. He has published widely on various Cleanroom topics.

QW2K Tutorial D1

Mr. Bill Deibler
(SSQC)

Making the CMM Work: Streamlining the CMM for Small Projects and Organizations

BACK TO QW2000 PROGRAM

Key Points

- Implement a realistic and useful strategy for deploying software development practices in small organizations and projects
- Simplify the CMM to support appropriate, effective, flexible software development processes for any small organization or project
- Resolve apparent discrepancies between the guidance in the CMM and the needs of small, commercial and internal software development projects and organizations
- Identify and prioritize elements of advanced levels that should be considered by every organization.

Abstract...

The SEI Software CMM is a comprehensive model that can serve as a basis for assessing and improving the effectiveness of software development organizations. The CMM was derived from the requirements of government purchasing agencies overseeing large, complex, third-party development projects. Because of their large project focus, the practices described in the CMM can appear to small, internal, or commercial software development organizations to be inapplicable or burdensome and bureaucratic. Version 1.1 of the CMM is published in two technical reports containing a total of nearly 600 pages. The size of the CMM makes it difficult to uncover the interrelationships among the elements that are essential to tailoring the model to a small software development environment. It also makes the model intimidating.

About the Author

William J. Deibler II has an MSc. in Computer Science and 20 years experience in the computer industry, primarily in the areas of software and systems development, software testing, and software quality assurance. Bill has extensive experience in managing and implementing CMM- and ISO 9001-based process improvement in software engineering environments. Robert Bamford has an MA in mathematics, and has managed training development, technical publications, professional services, and third-party software development. His over 20 years of experience include the implementation of a Crosby-based Total Quality Management System, facilitating quality courses, managing education teams, and serving on a corporate quality council.

Bob and Bill are the principals of SSQC. Since 1990, SSQC has specialized in supporting organizations in the definition and implementation of Software Engineering Practices, Software Quality

Assurance and Testing, Business Process Reengineering, ISO 9000 Registration and CMM implementation. SSQC offers HM2, a unique, hybrid appraisal method that defines and correlates the position of an organization with respect to both ISO 9001 and the CMM. The results of an HM2 assessment are a plan and framework for improving software engineering processes and for implementing the requirements of the two models.

Bob and Bill have developed and published numerous courses, auditing tools, research papers, and articles on interpreting and applying the ISO 9000 standards and guidelines and the SEI Capability Maturity Model for Software. Their articles have appeared in McGraw Hill's Quality Systems Update, IEEE COMPUTER, McGraw Hill's ISO 9000 Handbook, CrossTALK, and Software Marketing Journal.

They have presented research papers at numerous national and international conferences, including those sponsored by the American Society for Quality Control (ASQC), Pacific Northwest Software Quality (PNSQC), the Software Publishers Association (SPA), Software Technology Support Center (STSC), the Software Engineering Institute (SEI) and Software Research Inc.. Their courses have been attended by software engineering professionals from many of the country's leading technology companies. Their courses have been sponsored for their members by professional associations, including the ASQC, CSU Long Beach's Software Engineering Forum for Training, Semiconductor Equipment and Materials International (SEMI), Software Engineering Institute (SEI), UC Berkeley and UC Santa Cruz.

They are active United States TAG members in the ISO/IEC JTC1 SC7 - Software Engineering Standards subcommittee which is responsible for the development and maintenance of ISO 12207 and ISO 15504 (SPICE). Their software development clients have successfully achieved ISO registration and advanced CMM maturity levels.

BACK TO QW2000 PROGRAM

QW2K Tutorial E1

Mr. Ross Collard
(Collard & Company)

Test Planning

BACK TO QW2000 PROGRAM

Abstract...

Writing effective test plans is an essential survival skill for testing & QA professionals. The objective of this workshop is to develop the participants' skills in developing system test plans. The core of the workshop is hands-on practice, where the participants actually draft a test plan for a realistic case study.

Workshop Description

The case study for this workshop begins: "Congratulations! (Or condolences.) You have just been appointed the test team leader for a new, mission-critical project for your organization. The application uses client/server, Web and data base technology. This application and the situation background are described in the next few pages. The senior managers are confident you will do a good job, and have asked you to give them a detailed presentation on how you will proceed. Your presentation has been scheduled for NEXT WEEK. What will you tell them???"

Target Audience

Systems testers & quality assurance professionals, at the beginning to intermediate levels, and also systems analysts and designers, software engineers and programmers, project leaders, auditors, and users who are involved in systems testing.

Instructional Approach

This workshop shows the participants how to: o Analyze real-world test situations. o Identify and understand the key issues the test planner needs to master, in developing the test plan. o Develop workable test plans with a reasonable assurance of test coverage and reliability.

In small teams, participants work on a series of real-life test planning scenarios based on the case study. For each scenario, we will go through four steps: (a) the instructor explains the techniques which are most likely to apply and introduces the exercise, (b) each group develops its solution, (c) the group reviews and critiques a suggested model answer, and (d) the entire class compares results and discusses the issues from the exercise. We know that having fun and testing are a strange mix of concepts in one sentence, but the intention is to both learn and have some fun too.

Exercises

The core of the workshop is writing a draft of a test plan, in a series of exercises. These exercises take 20 to 30 minutes each, and each one is followed by review and discussion by the entire class. The questions addressed by each team in the series of exercises are as follows:

- Test Objectives
- Test Scope and Priorities
- Risk Strategy
- Test Entry & Exit Criteria
- Test Resources
- Test Project Work Plan
- Allowing for Contingencies
- Test Plan Format and Organization

Author Bio...

Ross Collard is a consultant who specializes in software testing & QA. Currently, he is working as a testing consultant for American Express, AT&T, Cisco, General Electric, IBM, Intel and Lucent.

Ross Collard has been a keynote speaker for several national and international conferences on software testing and quality assurance. He has conducted seminars for businesses, governments and universities, including George Washington, Harvard and New York Universities, MIT, U.C. Berkeley and UCLA.

He has a BE in Electrical Engineering from the University of New Zealand (where he grew up), an MS in Computer Science from the California Institute of Technology and an MBA from Stanford University. His set of books on software testing is due to be published next year.

BACK TO QW2000 PROGRAM

QW2K Tutorial F1

Dr. G. Bazzana & E. Fagnoni
(ONION s.r.l.)

Testing Web-based Applications: Techniques for Conformance Testing

BACK TO QW2000 PROGRAM

Presentation Abstract

The tutorial focuses on testing methods and tools which can be successfully applied to the testing of Web-based applications, notably:

- Internet WWW servers
- Intranet dynamic applications
- Extrenet e-commerce application

For each application, the following aspects are dealt with:

- testing peculiarities
- testing methods
- testing techniques
- testing tools
- case studies

The tutorial is constituted of over 150 slides which are constantly kept up-to-date w.r.t the unprecedented growth rate and dynamism of the Internet technologies and e-business. The distinguishing feature of the tutorial is that it combines methodological aspects with in-depth knowledge and experience of technical aspects, together with an overview of over 20 testing tools which can be applied depending on various phases.

About the Author

Gualtiero Bazzana is Partner and Managing Director of ONION S.p.A., an Italian SME specialised in the fields of sw communications, technologies, consulting (for more details please refer to the . Onion Home Page.

He matured and exploited know-how in conducting various medium sized and large projects for several companies in various application domains (telecommunications, data processing, MIS, process control, ecc.), covering topics like: sw development and testing, testing methods and tools, quality planning, test planning, reliability analysis, software product evaluation, process assessment and improvement, definition of quality systems in accordance to ISO 9001 and 9000/3, reviews and inspections, FDA computer system validation and so forth. He has matured significant technical experiences especially in the telecommunications domain (notably: switching systems and GSM

mobile radio systems) in CIM and in networking, including Internet/ Intranet/ Extranet services and solutions.

His research activity spanned in various fields of software engineering, ranging from Petri Nets to development methodologies, functional and structural test coverage, metrics and related tools, CAST, reliability evaluation, software development process evaluation and improvement, management by metrics, software product quality evaluation, security, technology transfer and total quality management techniques. He has acted as Project Leader of several European Projects within the ESPRIT initiative.

He is member of the Advisory Committee of the W3C (World Wide Web Consortium) and is author of over 50 publications at international conferences/ referred journals.

He can be contacted by e-mail at: gb@onion.it.

BACK TO QW2000 PROGRAM

QW2000 Tutorial G1

Edward Kit
(Software Development Technologies)

Software Testing in the Real World

BACK TO QW2000 PROGRAM

Key Points

This tutorial will prepare you to perform cost-effective analysis, testing, and overall evaluation of a software product or system.

- Learn the essentials of performing cost-effective analysis, testing and overall evalution of software.
- Aquire a vision for testing and identify keys to testing success and discover practical, effective, proven test methods.
- Review practical use of measurement, standards, and risk management and how to develop a testing improvement action plan.

About the Author

Edward Kit, founder and president of Software Development Technologies, is well known as a test expert, author, and keynote speaker at testing conferences. His best-selling book, *Software Testing in the Real World: Improving the Process*, has been adopted as a standard by companies around the world such as Sun Microsystems, Exxon, Chase Manhattan Bank and Cisco Systems.

His feature articles in Software Development Magazine have outlined new state-of-the-practice test automation models that are currently being adopted around the world. Mr. Kit continues to advise clients on bringing practical and proven software quality practices to their development efforts.

BACK TO QW2000 PROGRAM

QW2K Tutorial A2

Mr. Robert Binder (RBSC)

How To Write A Test Design Pattern

BACK TO QW2000 PROGRAM

Key Points

- An overview of patterns in software development.
- The test design pattern schema.
- Development of several actual test design patterns.
- Participant pattern development breakout.
- Discussion.
- Patterns for pattern development.

Presentation Abstract

A Test Design Pattern expresses a commonly used solution strategy for a recurring test design problem. The Test Design Pattern template is a list of questions that must be answered to articulate such a strategy. This tutorial introduces concepts and techniques for development of test design patterns.

A successful pattern is much more than narrow synopsis of technique, leading some people to observe that patterns are "discovered" and not "developed". Concepts will be presented for identifying possible patterns and validating pattern proposals.

Participants will work on pattern identification in a small group breakout sessions and discuss their results.

About the Author

Robert V. Binder is the developer of the test design pattern schema and author of *Testing Object-Oriented Systems: Models, Patterns, and Tools*, which presents 37 test design patterns.

BACK TO QW2000 PROGRAM

QW2K Tutorial B2

Dr. John Musa
(Consultant)

Developing More Reliable Software Faster and Cheaper

BACK TO QW2000 PROGRAM

Key Points

- Software Reliability Engineering (SRE) helps you develop more reliable software faster and cheaper.
- SRE is a standard, proven, widespread best practice.
- This tutorial quickly, efficiently teaches you the practical basics of how to apply SRE to your project.

Presentation Abstract

SRE is based on two powerful ideas:

- Determine how often your customers will use the various functions of your product; then focus your resources in proportion to use and criticality. This approach greatly increases your development efficiency and hence your effective resource pool for adding customer value to your product.
- Further increase customer value by setting quantitative reliability objectives that precisely balance customer needs for reliability, timely delivery, and cost; engineer project strategies to meet them; and track reliability during test to guide release.

You will learn how to:

- Determine the reliability / availability your customers need for your product and engineer your process to deliver it
- Develop operational profiles to describe how customers will use your product
- Use operational profiles to:
  - Increase development efficiency
  - Allocate test cases and test time efficiently
  - Make test represent field use
- Process failure data to:
  - Certify components you acquire
  - Track reliability growth and guide release of systems you develop

About the Author

John D. Musa is an independent consultant. He was formerly Technical Manager of Software Reliability Engineering (SRE) at AT&T Bell Laboratories, Murray Hill, NJ. He has varied and extensive experience as a software developer and software development manager.

John Musa is one of the creators of the field of software reliability engineering (SRE). Principal author of the highly-acclaimed pioneering book "Software Reliability: Measurement, Prediction, Application" (McGraw-Hill), he has published some 100 papers. His new book "Software Reliability Engineering: More Reliable Software, Faster Development and Testing" (McGraw-Hill) focuses on practice. The IEEE elected him Fellow in 1986 for his many seminal contributions. He was recognized in 1992 as the person who had contributed the most to testing technology. His leadership has been noted by every recent edition of Who's Who in America and American Men and Women of Science.

Musa is widely recognized as the leader in the practical application of SRE. He initiated and spearheaded SRE practice at AT&T, resulting in its selection as a "Best Current Practice" and leading the team that codified that practice. He was involved in developing most of the details. His many contributions include the concept of execution time (used almost universally in the field today), the distinction between failures and faults, the operational profile and all its related technology, the ideas of operational development, Reduced Operation Software (ROS), the fault exposure ratio and software reliability prediction, and the integration of software reliability engineering into all phases of the software development process. He has been involved for some 25 years in deploying the practice, first within AT&T and afterwards worldwide through his consulting work.

He is an experienced international speaker and teacher (over 200 major presentations) with consistently outstanding feedback. One of the founders of the IEEE Technical Committee on Software Reliability Engineering, he is closely networked with the other leaders in the field, providing a broad perspective.

(John Musa's Home Page)
(j.musa@ieee.org

BACK TO QW2000 PROGRAM

QW2000 Tutorial C2

Mr. Tom Gilb
(Result Planning Limited)

Requirements Engineering for Software Developers and Testers

BACK TO QW2000 PROGRAM

Presentation Abstract

A. Narrative Description 40-60% of all software bugs which escape test to the field user have been traced to requirements and design specifications before coding. It has then been proved that Inspecting the specifications sharply reduces this problem. This workshop will explore all aspects of Specification Quality Control in a hands on practical workshop. Participants will actively experience the technology necessary to attack this quality challenge.

B. Learning Objectives. 1. To learn the problem and the solutions mainly by means of personal experience. You will get a series of individual tasks which will teach you basic principles of specification and quality control of specification.

C. Detailed Outline 1. The Ambiguity Test: proving that specifications are unintelligible.

2. Rules: Selecting strong standards for specification which enable quality control.

3. Process Control: Deciding on the economically allowable Major Defect density allowed for a specification to be released to your colleagues

4. Planning: Selecting a suitable sample to check. Selecting checklists. Allocating specialist checking roles on the team. Deciding on checking rates using optimum rate data.

5. Checking: Individual effort to find defects, and especially Major defects.

6. Data Collection: Gathering data on the checking phase: defects, Majors, Rate, Sample size.

7. Extrapolation: Calculating probable team result of unique Major defects/ Logical Page. Calculating total defect density. Calculating defects remaining after corrections {per page, total}. Calculating total future rework costs based on remaining Major defects.

8. Drawing Conclusions: Can the document exit according to our Exit Conditions? If not, what should we do?

9. Observations. What did you learn? What surprised you the most? What do you think you should do back at your own work about these things? What barriers do you see to doing them? What can you do to remove the barriers?

About the Author

Tom Gilb is an independent consultant, teacher and author. He works mainly in UK, Europe and North America. He is resident in Norway.

Tom coined the term 'Software Metrics' with the publication of his book of the same name in 1976 (European edition) and 1977 (USA edition). This work is the acknowledged (by R. Radice and W. Humphrey) as inspiration for much of the Software Engineering Institute's Capability Maturity Model Level 4 (SEI CMM Level 4). His other books include Principles of Software Engineering Management (1988, now in 13th printing) and Software Inspection (1993 with Dorothy Graham). His main professional interest is the development of powerful Systems Engineering methods (covering Requirements, Design, Quality Control and Project Management).

BACK TO QW2000 PROGRAM

QW2K Tutorial D2

Mr. Rob Baarda & Tim Koomen (IQUIP Informatica BV)

Stepwise Improvement of the Testing Process using TPI(tm)

BACK TO QW2000 PROGRAM

Key Points

Discussion of practical experiences regarding:

- The Context, Why, What, When, Where, Who, etc. to improve?
- TPI, The Test Process Improvement Model
- Application of the Model, Management of Change

Presentation Abstract

This tutorial deals with the TPI-model, which is based on current state-of-the-art test process improvement practices. The model gives practical guidelines for assessing the maturity level of testing in an organisation and for step by step improvement of the process. Since the appearance of the book "Test Process Improvement: a practical step-by-step guide to structured testing" (by Tim Koomen and Martin Pol) in May æ99, the model is rapidly becoming an international standard for improving test processes, used in numerous countries like the United States, most European Community countries, Australia, South Africa, Mexico, Columbia, China and India. The model is frequently used to help an organisation meet the test requirements at CMM level 3.

Also included in the tutorial is a general description of the application of model, which deals with how to implement and how to consolidate the improvements.

About the Author

Graduated in 1987 in Informatica at the University of Amsterdam, Tim Koomen is since 1991 a professional tester. He participated in several testing projects for clients of IQUIP Informatica in the Netherlands, a company with over 300 dedicated testers. Tim is a member of the R&D-team, covering issues like E-commerce, ERP, testfactories and TPI. Currently he is advising organisations how to improve their testing processes, using the presented model. He is the co-author of the TPI-book, translated in Dutch, English and German, and regularly presents at conferences (Eurostar '97: tutorial TPI, Eurostar '98: track presentation TPI, Quality Week Europe Æ99: TPI tutorial and two presentations, Eurostar Æ99: discussion session and presentation) and training sessions throughout Europe.

Rob Baarda is an information systems professional for more than 20 years, following the path from programmer to consultant. Since 1986 he specialised in the field of testing. Starting with developing

automated tests, Rob moved after a few years to the methodology of testing. He is now part-time researching various test subjects in the R&D department of IQUIP, besides working as an international test consultant and teaching TMap½ and TPI½. He also presented in QWE'99 about Risk Based Test Strategy.

BACK TO QW2000 PROGRAM

QW2000 Tutorial E2

Dr. Linda Rosenberg, Ms. Ruth Stapko, & Mr. Albert Gallo
(NASA GSFC)

Risk-based Object Oriented Testing

BACK TO QW2000 PROGRAM

Key Points

- Software Metrics
- Object Oriented
- Software Testing

Abstract...

Software testing is a well-defined phase of the software development life cycle. Functional ("black box") testing and structural ("white box") testing are two methods of test case design commonly used by software developers. A lesser known testing method is risk-based testing, which takes into account the probability of failure of a portion of code as determined by its complexity. For object oriented programs, a methodology is proposed for identification of risk-prone classes.

Risk-based testing is a highly effective testing technique that can be used to find and fix the most important problems as quickly as possible. Risk can be characterized by a combination of two factors: the severity of a potential failure event and the probability of its occurrence. Risk can be quantified by using the equation

Risk = … p(Ei) * c(Ei),

Where i =1,2,à,n. n is the number of unique failure events, Ei are the possible failure events, p is probability and c is cost.

Risk-based testing focuses on analyzing the software and deriving a test plan weighted on the areas most likely to experience a problem that would have the highest impact. This looks like a daunting task, but once it is broken down into its parts, a systematic approach can be employed to make it very manageable.

The severity factor c(Ei) of the risk equation depends on the nature of the application and is determined by domain analysis. For some projects, this might relative to mission criticality, for some it would be in terms of financial loss, while for others it might be related to safety. Severity assessment requires expert knowledge of the application as well as a thorough understanding of the potential costs of various failures. Musa addresses ways to estimate the severity of software module failures in

his discussion of "Operational Profiles" in his book, Software Reliability Engineering.

Both severity and probability of failure are needed before risk-based test planning can proceed. Severity assessment is not addressed here, however, because it involves so much application-specific knowledge. Instead we confine the remainder of the discussion to the other crucial part of the risk equation, assessing the likelihood of component failures, p(Ei), and we suggest a way to capture the information directly from the source code, independent of domain knowledge.

The task we address is to determine how likely it is that each part of a software system will fail. It has been proven that code that is more complex has a higher incidence of errors or problems. For example, cyclomatic complexity has been demonstrated as one criterion for identifying and ranking the complexity of source code. Therefore, using metrics to predict module failures might simply mean identifying and sorting them by complexity. Then using the complexity rankings in conjunction with severity assessments from the domain risk analysis described above would identify which modules should get the most attention. But restricting the focus to ranking module complexity is an over-simplification, and we may fail to detect some very risk-prone code. Experience has shown that object oriented programming, in particular, can result in deceptively low values for common complexity metrics. The hierarchical nature of object oriented code calls for a multivariate approach to measure complexity.

We narrow the topic further and focus specifically on object oriented software. The Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center has identified and applied a set of six metrics for object oriented source code. These metrics have been used in the evaluation of many NASA projects and empirically derived guidelines have been developed for their interpretation. In this paper, we will identify and discuss the interpretation and application of these metrics.

The purpose of the metrics information is to identify the classes at highest risk for error. While there is insufficient data to make precise ranking determinations, there is enough information to justify additional testing of those classes that exceed the recommended values. Then, combining module risk evaluation with expert criticality estimates, we have the two components needed for determining risk by class. Allocating testing resources based on these two factors, severity and likelihood of failures, amounts to risk-based testing.

Object oriented software metrics can be used in combination to identify classes that are most likely to pose problems for a project. The SATC has used the data collected from thousands of object oriented classes to determine a set of benchmarks that are effective when used simultaneously in identifying potential problems. When problematic classes are also identified by domain experts as critical to the success of the project, testing can be allocated to mitigate risk. Risk-based testing will allow developers to find and fix the most important software problems earlier in the test phase.

Author Bio...

Dr. Linda H. Rosenberg is the Division Chief responsible for the Software Assurance Technology Center (SATC) at Goddard Space Flight Center, NASA. The SATC primary responsibilities are in the areas of Metrics, Assurance tools and techniques, Risk management, and Outreach programs.

Although she oversees all work areas, Dr. Rosenberg's area of expertise is metrics. The emphasis of

her work with project managers is the application of metrics to evaluate the quality of development products. Dr. Rosenberg holds a Ph.D. in Computer Science, an M.E.S. in Computer Science, and a BS in Mathematics.

Contact Dr. Rosenberg by Email at Linda.Rosenberg@gsfc.nasa.gov or visit the SATC WebSite at satc.gsfc.nasa.gov.

BACK TO QW2000 PROGRAM

QW2000 Tutorial F2

Mr. Adrian Cowderoy (MMHQ)

Cool Q - Quality Improvement for Multi-disciplinary Tasks in Website Development

BACK TO QW2000 PROGRAM

Key Points

- Player identification. Identification of the business/marketing players. Use of profiling and stereotyping to model the great diversity that are common at most websites. Identification of the key content and domain experts who understand how to make the website effective for its primary business purpose. Identification of the tool features and legacy systems/content that constraint that website.
- Feature identification. Use of checklists, questionnaires, models and behaviour monitoring to identify the features required by the different players.
- Quality profiling to support constructive negotiation between the different players, and to assure a complete view of all important quality features.
- Commercial design. Achieving the balance between system constraints (technical quality) and system benefits. Design websites that are engaging, inspiring and (if you have the talent) cool.
- Killer-feature monitoring, to identify and track commercially sensitive system benefits.
- Use of size and complexity measures (and quality features) to identify components and design decisions that create high maintenance costs and risks for website content.
- Use of risk and cost assessment to optimize the quality profile and design complexity.
- Use of risk management planning to indicate how to improve quality in the project by appropriate staffing, testing, monitoring, and other quality-improvement actions.
- Creating a learning organization. Combining commercial training materials and in-house know-how to create a simple knowledge-base of good practice.

Abstract...

The tutorial is targeted at website producer/directors and managers. It is also strongly recommended for software quality people who are moving into the Internet business. The tutorial addresses website content and structure, and the functionality resulting from using Internet development tools. The tutorial begins by explaining the background of how the web embraces multi-disciplines and multi-practices, and how new web-savvy organizations go beyond the boundaries of traditional industries. The tutorial continues by introducing a set of quality improvement methods that have been developed over the last three years, for web-site development. An explanation is provided of how

these methods support each of the main archetypes that are used in the workplace. That is, the engineer, the hunter/trickster, the gatherer, and the warrior.

About the Author

Adrian Cowderoy is Managing Director of the Multimedia House of Quality Limited, a company which he established to promote quality-improvement methods for the production of websites and multimedia.

Mr Cowderoy was the General chair of ESCOM-SCOPE-99 and ESCOM-ENCRESS-98 conferences, and was Program chair for ESCOM 96 and 97 (The European Software Control and Metrics conference promotes leading-edge developments in industry and research, worldwide û see www.escom.co.uk). He is the METRICS-ESCOM Coordinator for IEEE METRICS 2001 and was on the Program committee of Metrics 98 and 99, European Quality Week 99 and COCOMO/SCM 96-99. In 1998 he was acting Conference Chair of the Electronics and Visual Arts conference in Gifu, Japan. He is a registered expert to the European Commission DGXIII.

He has provided consultancy and industrial training courses on quality management, risk management, and cost estimation to the aerospace and medical industries in the UK, Germany and Italy since 1995. He also lectures at Middlesex University (www.mdx.ac.uk) on e-commerce project management and managing Internet start-up's, and at City University, London (www.city.ac.uk), on project management for systems development.

Mr Cowderoy was project manager and technical director of MultiSpace, a 14-month million-dollar initiative sponsored by the European Commission in which 12 European organizations explored the potential to apply quality-improvement methods to multimedia and website development projects. (See www.mmhq.co.uk/multispace and www.cordis.lu/esprit.)

He was a Research fellow at City University from 1990-1998, and a Research Associate at Imperial College from 1986-1989. He was also a quality consultant and software developer at International Computers Limited, UK, from 1980-1985, where he worked on operating and networking systems for mainframes and distributed systems.

His academic qualifications include an MSc in Management Science from Imperial College, University of London in 1986, and is a member of the Association of MBA's. He received a BSc in Physics with Engineering from Queen Mary College, University of London, in 1979.

Mr. Cowderoy has published and presented extensively on multimedia quality and software cost estimation. He was joint editor of Project Control for 2000 and Beyond (Elsevier, 1998), Project Control for Software Quality (Elsevier, 1999), and Project Control: The Human Factor (Elsevier, 2000).

BACK TO QW2000 PROGRAM

QW2000 Keynote 1P1

Dr. Stu Feldman
(Director, IBM Institute for Advanced Commerce,
Director, Networked Computing Software Research,
IBM T. J. Watson Research Center
IBM Corporation)

Internet and E-Commerce: Issues and Answers

BACK TO QW2000 PROGRAM

Presentation Abstract

The rise of Internet usage, and in particular the explosive growth of electronic commerce is raising many problems for software researchers and practitioners. The amount of Internet commerce has been literally rising exponentially (doubling about every 9 months since 1995), and the number of participants, sites, domains, etc. Are also doubling rapidly. We are thus led to contradictory requirements: many companies have been changing their e-business strategies every 6 months, their firms are increasingly dependent on proper functioning of these business processes and systems, yet building complex highly reliable software still takes many months (if you are lucky) or years.

It turns out that e-commerce pushes limits of the state of the art in many areas, including:

- Distributed systems (with millions of servers, son to be billions of participants)
- Transactions (long lived changing multiple participants)
- Distributed massive content management
- Interface paradigms (how to manage mobile devices, speech, etc. in a world of legacy data)
- Performance and Reliability management (guarantees and expectations in a dynamic world)
- Security, Authentication
- Search (for processes and activities as well as complex content)
- New application models and structure
- Software engineering

On the last point, there are many challenges facing the software engineer. Classic processes and experience apply to centralized design, control and management. There is less support for a world of independent creation, ownership, and redefinition. Business models (which will themselves be software objects) will be defined and implemented in terms of other services and applications, each of

which might come from several sources and be administered independently.

Yet it is necessary to have expectations of performance, security, reliability and availability. We must manage the specification, design, creation, testing, and dynamic reconfiguration of key processes in such a fluid world.

This talk will address these challenges to the state of the art and practice, offering questions and some answers.

About the Speaker

Feldman did his academic work (AB, Princeton and PhD, MIT) in astrophysics and mathematics. He is a Fellow of the IEEE and a Fellow of the ACM. He has been a member of the Board of the Computing Research Association and chair of ACM SIGPLAN and is founding chair of the ACM SIG on E-Commerce.

He was a computer science researcher at Bell Labs and a research manager at Bellcore before joining IBM in mid-1995. He has published research in software engineering (and was the creator of Make), programming languages, scientific computing and other areas of computer science. He was also architect for a large new line of software products at Bellcore.

At IBM's T. J. Watson Research Center, Feldman leads a department doing research in a wide variety of network-related technologies and application enablers, including electronic commerce, parallel databases, anti-virus technology, and advanced internet multimedia. He is also the Director of IBM's Institute for Advanced Commerce, an organization created to increase IBM's intellectual leadership in e-commerce, and to forge better connections to the outside research world as well as to accelerate creation of new technologies for support of e-Business.

BACK TO QW2000 PROGRAM

The rise of Internet usage, and in particular the explosive growth of electronic commerce is raising many problems for software researchers and practitioners. The amount of Internet commerce has been literally rising exponentially (doubling about every 9 months since 1995), and the number of participants, sites, domains, etc. Are also doubling rapidly. We are thus led to contradictory requirements: many companies have been changing their e-business strategies every 6 months, their firms are increasingly dependent on proper functioning of these business processes and systems, yet building complex highly reliable software still takes many months (if you are lucky) or years.

It turns out that e-commerce pushes limits of the state of the art in many areas, including:

- Distributed systems (with millions of servers, son to be billions of participants)
- Transactions (long lived changing multiple participants)
- Distributed massive content management
- Interface paradigms (how to manage mobile devices, speech, etc. in a world of legacy data)
- Performance and Reliability management (guarantees and expectations in a dynamic world)
- Security, Authentication
- Search (for processes and activities as well as complex content)
- New application models and structure
- Software engineering

On the last point, there are many challenges facing the software engineer. Classic processes and experience apply to centralized design, control and management. There is less support for a world of independent creation, ownership, and redefinition. Business models (which will themselves be software objects) will be defined and implemented in terms of other services and applications, each of which might come from several sources and be administered independently.

Yet it is necessary to have expectations of performance, security, reliability and availability. We must manage the specification, design, creation, testing, and dynamic reconfiguration of key processes in such a fluid world.

This talk will address these challenges to the state of the art and practice, offering questions and some answers.

QW2000 Tutorial G2

Chris Loosley & Eric Siegel (Keynote Systems)

Web Application Performance

BACK TO QW2000 PROGRAM

Presentation Abstract

How do you deploy reliable, responsive e-commerce applications in the uncontrolled and relatively slow environment of the Internet? (Hint: it's not just a matter of bandwidth). Using many illustrative examples and studies drawn from their experience measuring and improving leading e-commerce sites, the presenters explain the fundamentals of web application performance engineering, from the browser to the database backend.

About the Speakers

Chris Loosley is Senior Internet Consultant with Keynote Systems. He has a technical background in the design and performance of enterprise information systems, and over 25 years experience in the field of software performance engineering. Mr. Loosley is the author of "High-Performance Client/Server" (John Wiley).

Eric Siegel is Senior Internet Consultant with Keynote Systems and wrote "Designing Quality of Service Solutions for the Enterprise (John Wiley). He was the technical leader for all of Tandem Computer's data communications specialists worldwide, and also worked for Network Strategies and MITRE. His B.S. and M.E.E. are from Cornell University.

BACK TO QW2000 PROGRAM

QW2000 Keynote 1P2

Bill Gilmore
(Intel Corporation)

The Intel Software Corporate Quality Network

BACK TO QW2000 PROGRAM

Presentation Abstract

Intel is a leading supplier of computer and internet building blocks, and has developed a powerful Corporate Quality Network (CQN) to drive improvement and quality throughout the organization. Software engineering is playing an increasingly important role in Intel's efforts, and in the last few years a software wing within CQN has been established and grown to drive software quality.

Software Quality Engineers from the Software CQN are distributed throughout the company on assignment to various divisions to establish, introduce, and nurture the adoption of "Best Known Methods". This presentation overviews the structure of this effort, and reviews the strategies, successes, and challenges of the program.

About the Author

Dr. Bill Gilmore is a senior Software Quality Process Engineer in the Corporate Quality Network at Intel, and is responsible for establishing quality processes and improving product quality in Intel business divisions. He is currently assigned to the Microprocessor Group, and has worked with the motherboards and BIOS division, and systems management divisions at Intel. This work has included CMM assessments, Product Life Cycle methodologies, and metrics and other software methods.

Prior to working at Intel, Dr. Gilmore was leader and manager for Software Process Improvement at Tektronix, Inc. He used a SEI Self-Assessment, improvement projects, Software Engineering Councils, and post-project reviews, to markedly increase awareness, priority, and participation in software process improvement.

Dr. Gilmore has a Ph.D. in Astronomy, a B.S. in Engineering Physics, and did graduate study in Organizational Development. He has published ~30 papers in Software Engineering, Strategic Planning, and Astronomy.

BACK TO QW2000 PROGRAM

The Intel Software Corporate Quality network

Intel is a leading supplier of computer and internet building blocks, and has developed a powerful Corporate Quality Network (CQN) to drive improvement and quality throughout the organization. Software engineering is playing an increasingly important role in Intel's efforts, and in the last few years a software wing within CQN has been established and grown to drive software quality.

Software Quality Engineers from the Software CQN are distributed throughout the company on assignment to various divisions to establish, introduce, and nurture the adoption of "Best Known Methods". This presentation overviews the structure of this effort, and reviews the strategies, successes, and challenges of the program.

-->

QW2000 Vendor Technical Paper 2V1

Ms. Melissa Borza
(Computer Associates)

Enterprise Change and Configuration Management

BACK TO QW2000 PROGRAM

Key Points

- Preserving the integrity of software resources is the goal of effective Change and Configuration Management (CCM). As software development becomes more complex, CCM solutions must address the new challenges inherent in managing development in today's heterogeneous, distributed environment.
- This presentation examines the challenges of Enterprise Change and Configuration Management and discusses how an integrated CCM strategy can help you successfully meet these challenges. It also explores the issues you need to address to be successful and describes Computer Associates CCM strategy for a heterogeneous, distributed environment.
- Learn how the integration of CA's best-of-breed CCM products (Endevor for OS/390 and CCC/Harvest) can provide enterprise-wide benefits for your organization.

About the Author

Melissa Borza is the Product Manager for the change and configuration management (CCM) products for Computer Associates International Inc. These products include the industrial-strength tools: Endevor=AE for = MVS, CCC=AE/Harvest, and Enterprise CCM(tm). Melissa is responsible for the marketing and support of these products, and she has written articles and has delivered seminars on this subject around the world.

BACK TO QW2000 PROGRAM

Enterprise Change and Configuration Management

Melissa Borza, Marketing Director
Computer Associates International Inc.

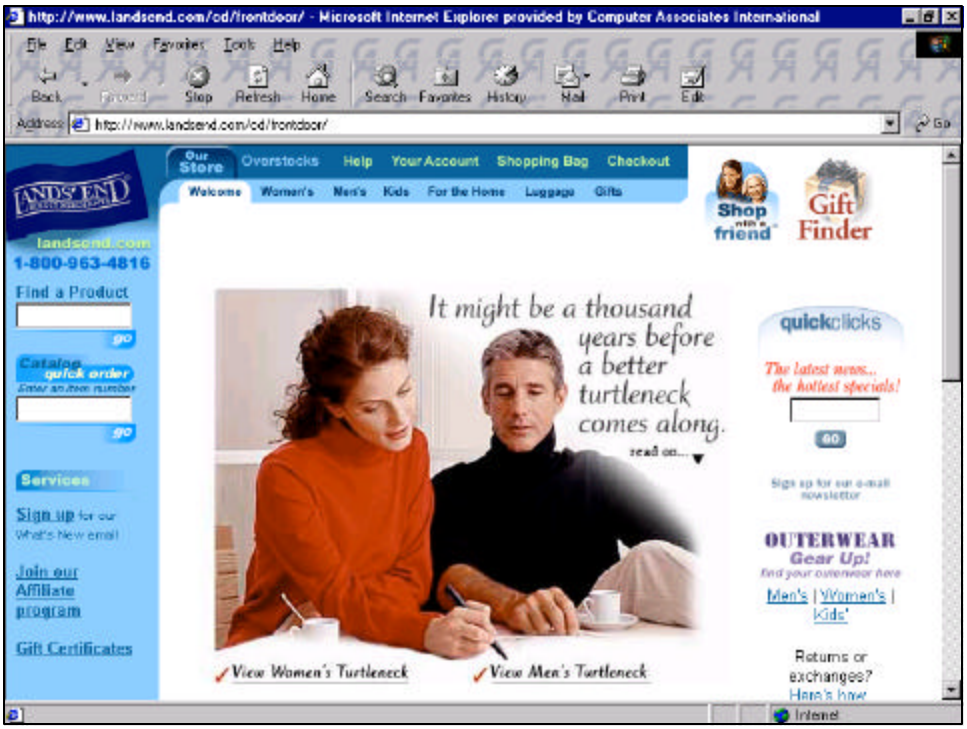
Quality Week 2000

Topics of Discussion

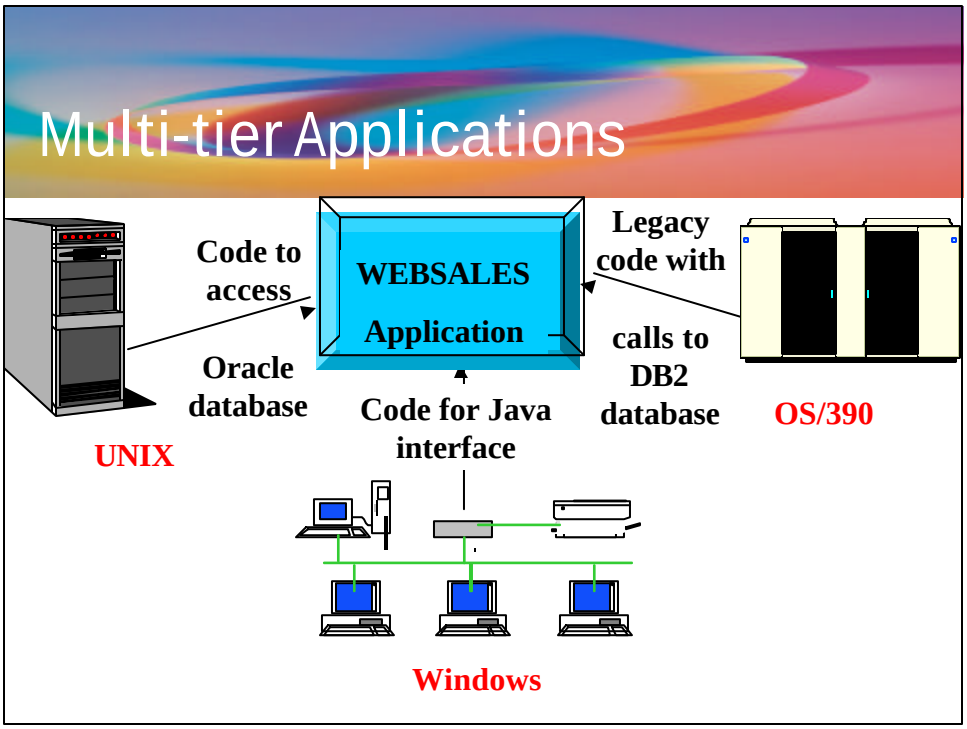
- Multi-tier applications
- Management issues
- Integrating the enterprise
- Summary

Multi-Tier Applications

- A single business system that is:
  - Developed on multiple platforms
  - Deployed on multiple platforms
- Heterogeneous *yet* integrated components
- Increasingly strategic for business, e.g. data warehousing, e-commerce



Charles Schwab | Home - Microsoft Internet Explorer provided by Computer Associates International
Charles Schwab
compare over 1,650 mutual funds
Welcome to investing made easy
Schwab Customers
Limited Time Offer
Trade with Velocity
Additional Info
Investor Satisfaction
Let Us Help You With...
Latest from Schwab
Live Events
Free Quotes & Charts
Market Snapshot
Dow Jones Industrial Average


Multi-tier Applications
Code to access
Oracle database
UNIX
WEBSALES
Application
Code for Java interface
Legacy code with
calls to DB2 database
OS/390
Windows

Management Issues: The Difficulties

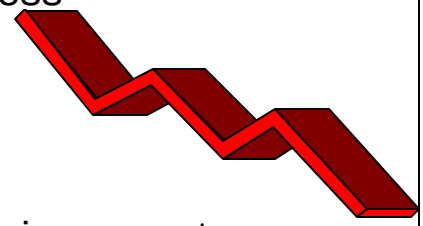
- Errors inherent in manual procedures
  - Compounded by multiple platforms
- Heterogeneous tools and procedures
  - Mainframe procedures may be different than UNIX/PC
- Disjointed approvals
  - More people, more platforms means more errors and delays

Management Issues: The Difficulties

- Non-synchronized deployment
  - Some components ready for promotion, others take longer
- Incomplete backout
  - One component must be backed out, others remain in production
- Heterogeneous users
  - Limited access to CCM tools
  - Limited expertise with CCM tools

Management Issues: The Business Consequences

- Production failures
- Decreased competitiveness
- Lost revenue
- Higher costs
- Lost productivity
- Loss of confidence in business systems



What's Needed: Enterprise Release Automation

- Leverage existing tools and procedures
- Ensure all affected components are collected
- Obtain and track necessary approvals
- Automate promotions and demotions

…regardless of platform

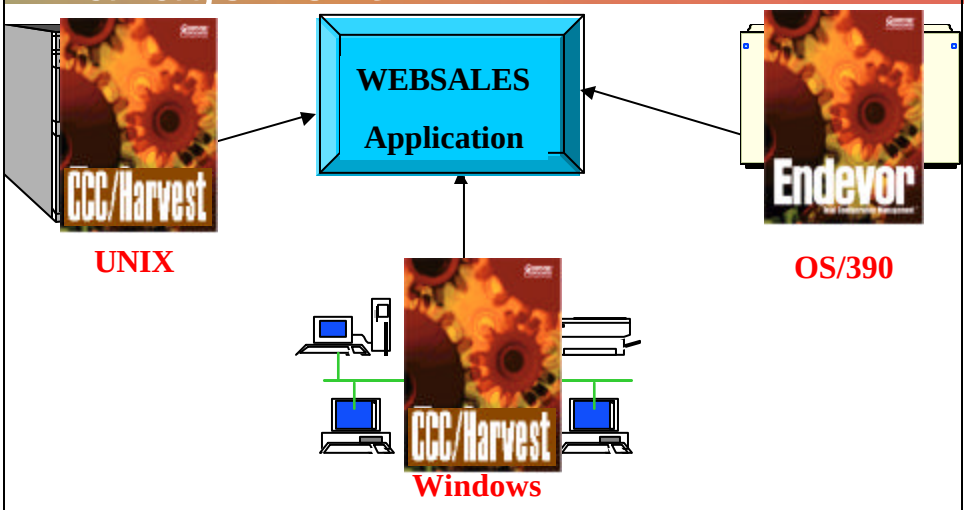
What's Needed: Enterprise Release Automation

- Centralized management
- Accessible from anywhere
- Intuitive for all types of users
  - Approvers
  - Developers
  - Managers

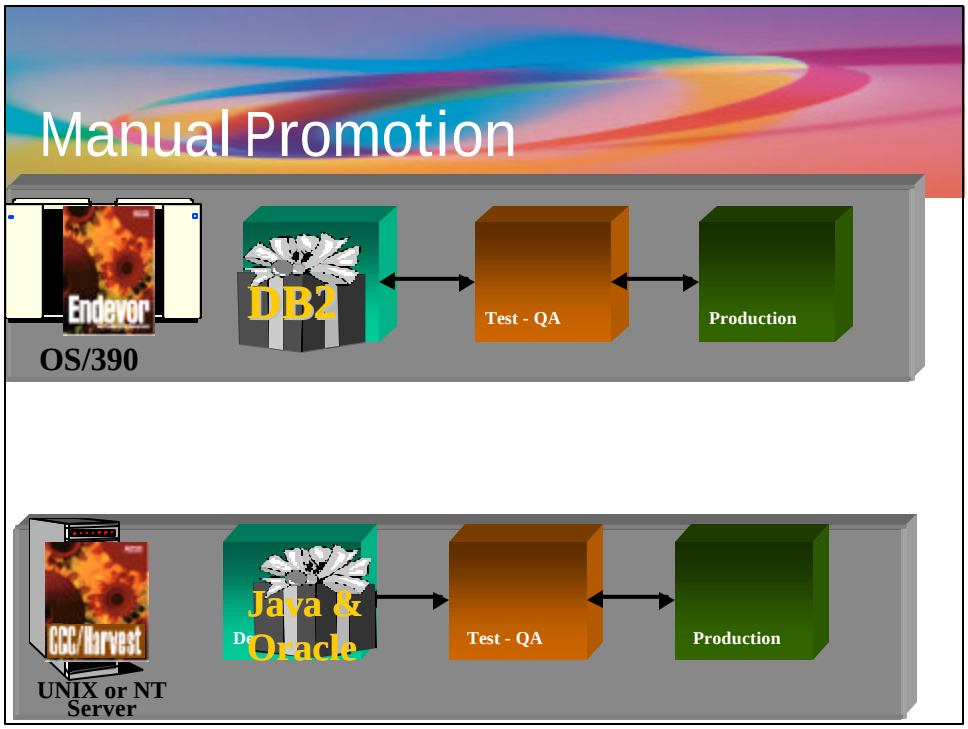
Integrating the Enterprise

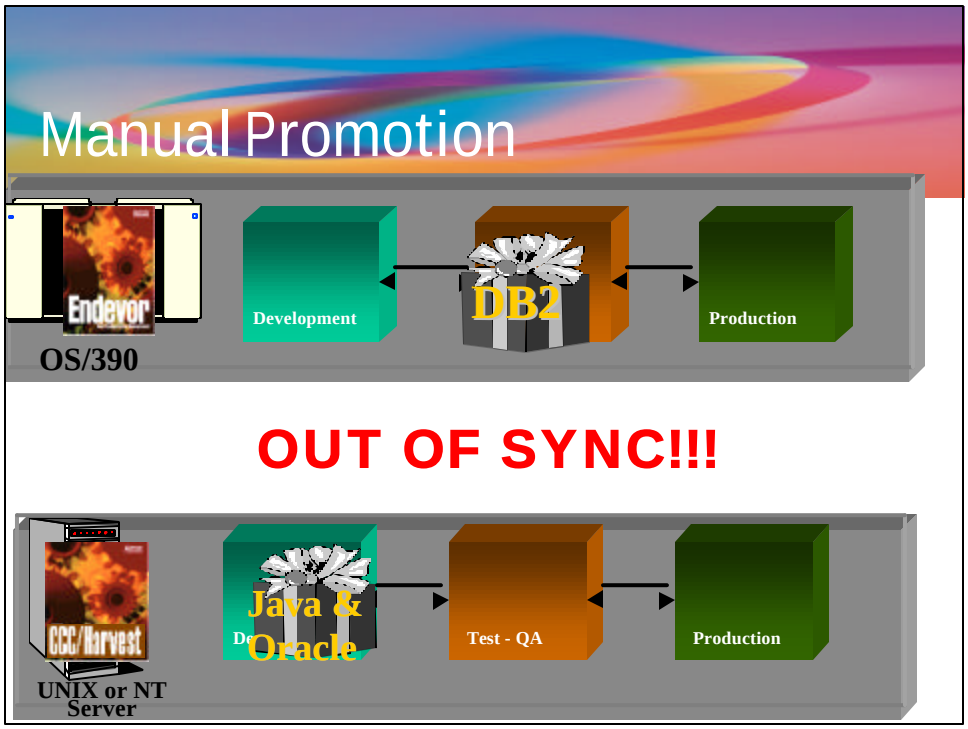
Enterprise CCM

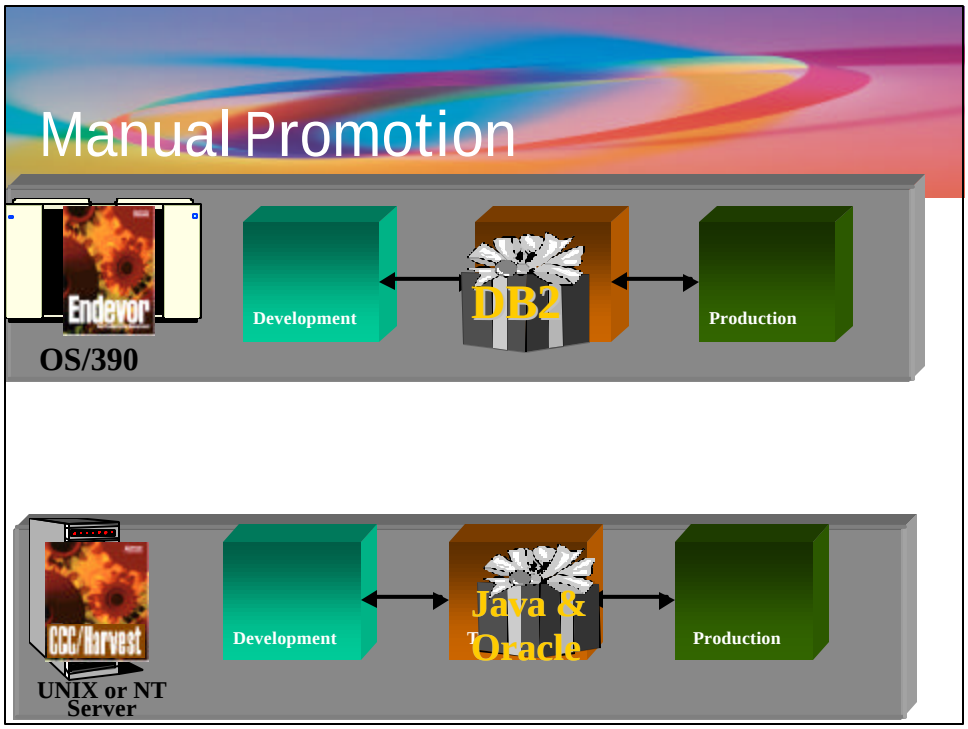
Application Life Cycle Management

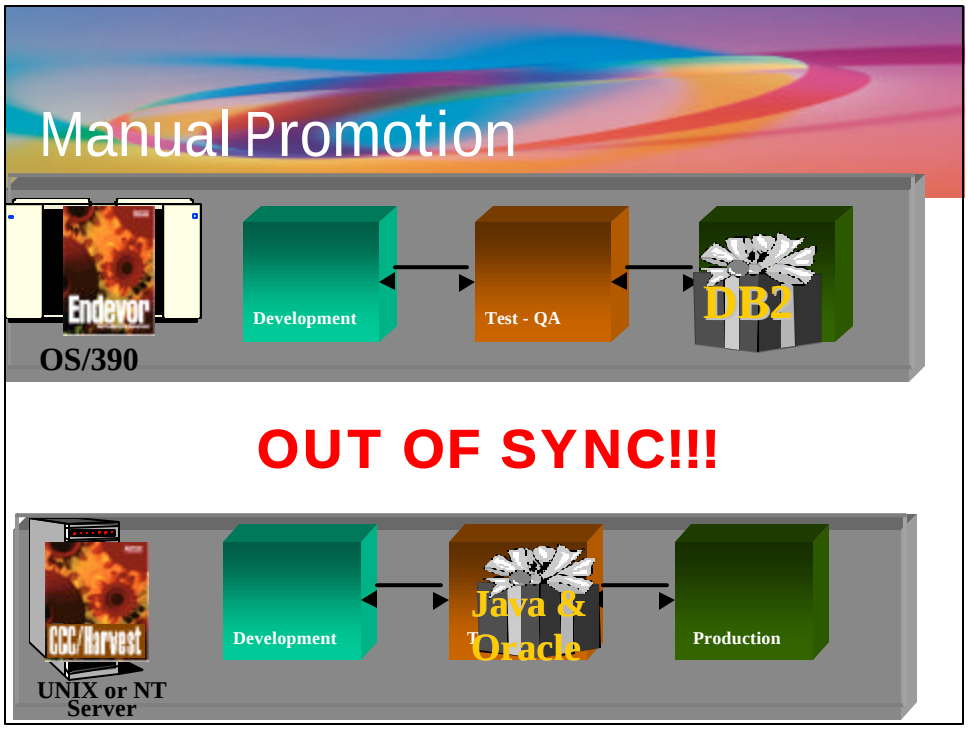


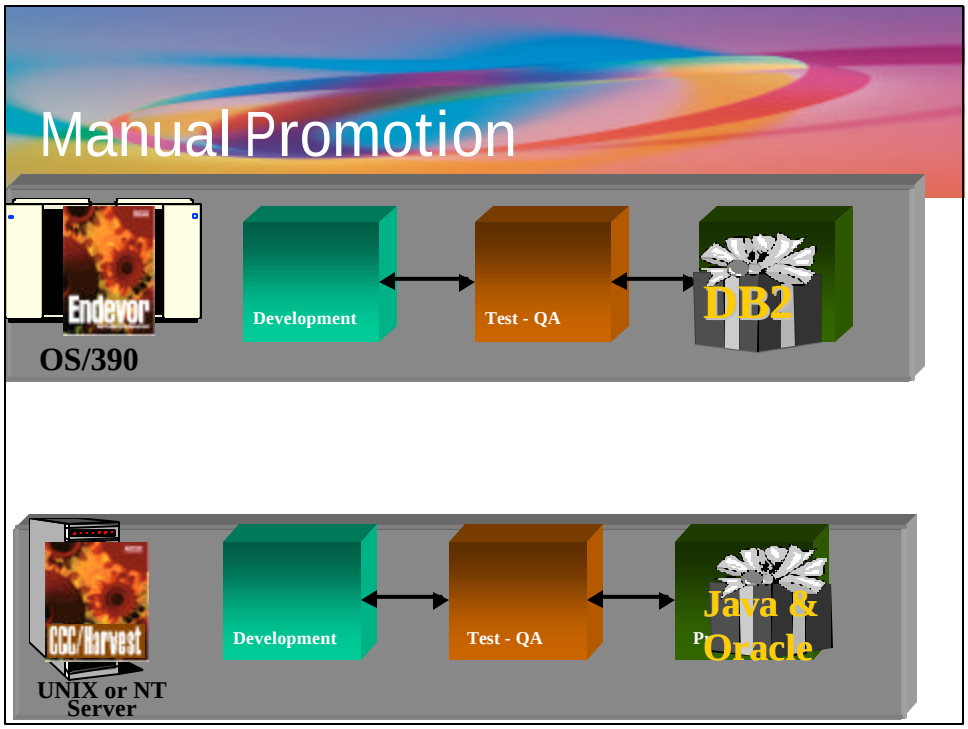
Change Packages

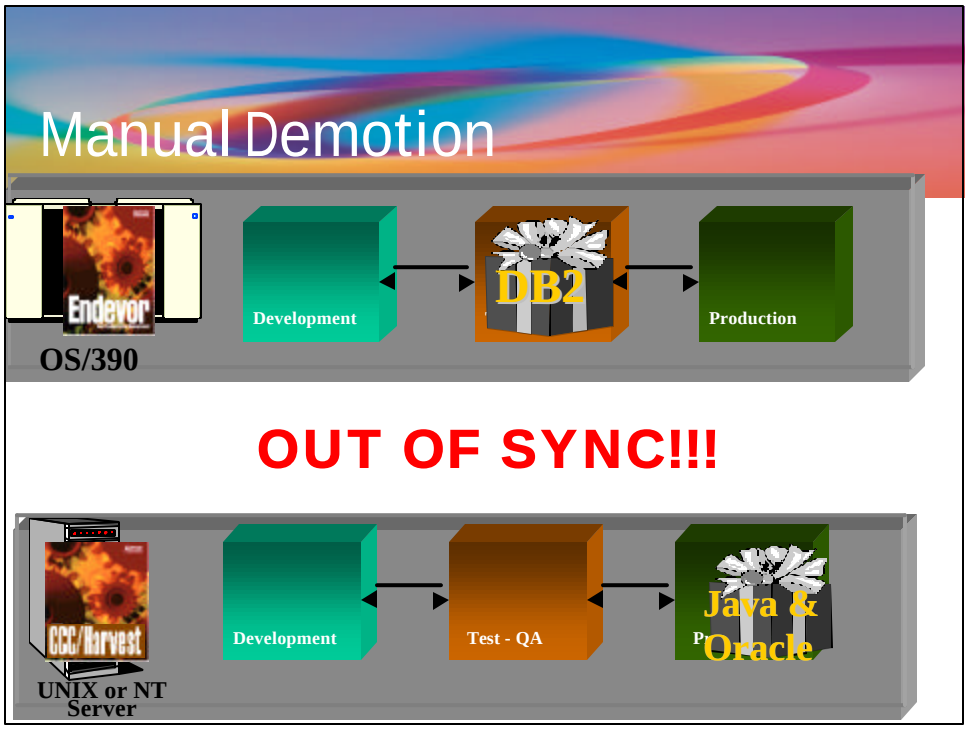

Manual Promotion
Endevor
OS/390
DB2
Test - QA
Production
CCC/Harvest
UNIX or NT
Server
Java &
Oracle
Test - QA
Production

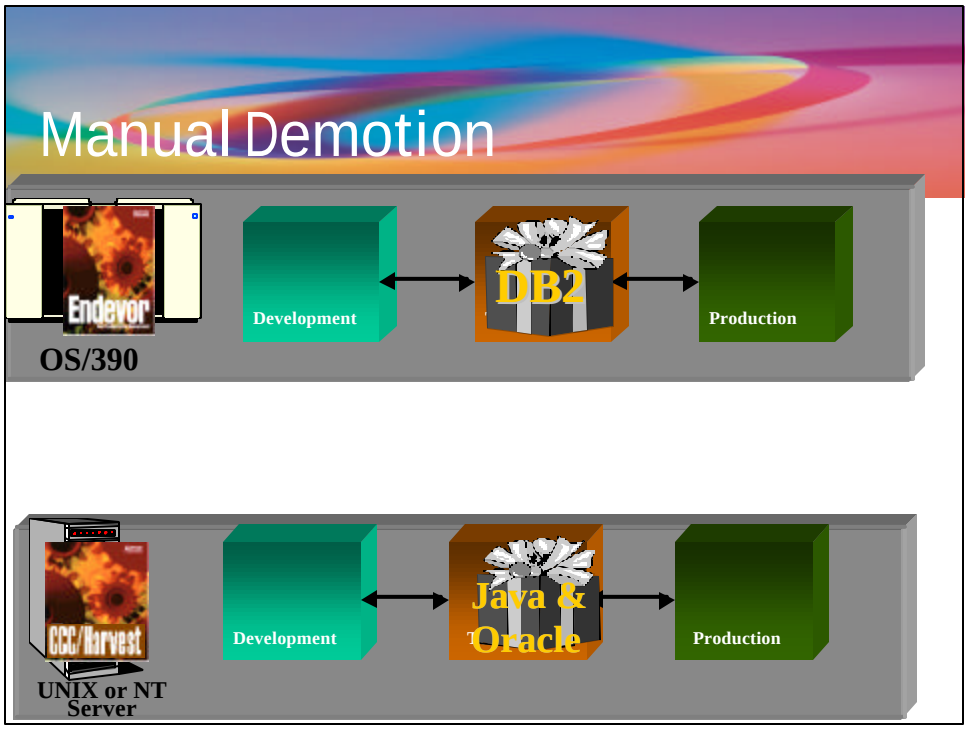

Manual Promotion
Endevor
OS/390
Development
DB2
Production
OUT OF SYNC!!!
CCC/Harvest
UNIX or NT
Server
Java &
Oracle
Test - QA
Production

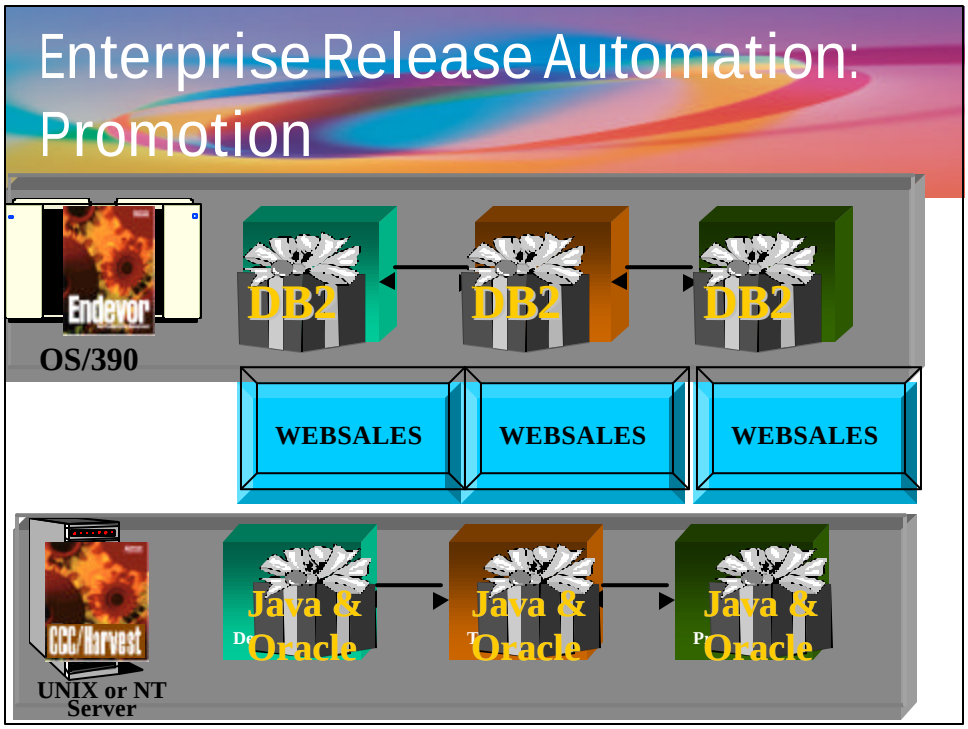

Manual Promotion
Endevor
OS/390
Development
DB2
Production
CCC/Harvest
UNIX or NT
Server
Development
Java &
Oracle
Production


Manual Promotion
Endevor
OS/390
Development
Test - QA
DB2
OUT OF SYNC!!!
CCC/Harvest
UNIX or NT
Server
Development
Java &
Oracle
Production

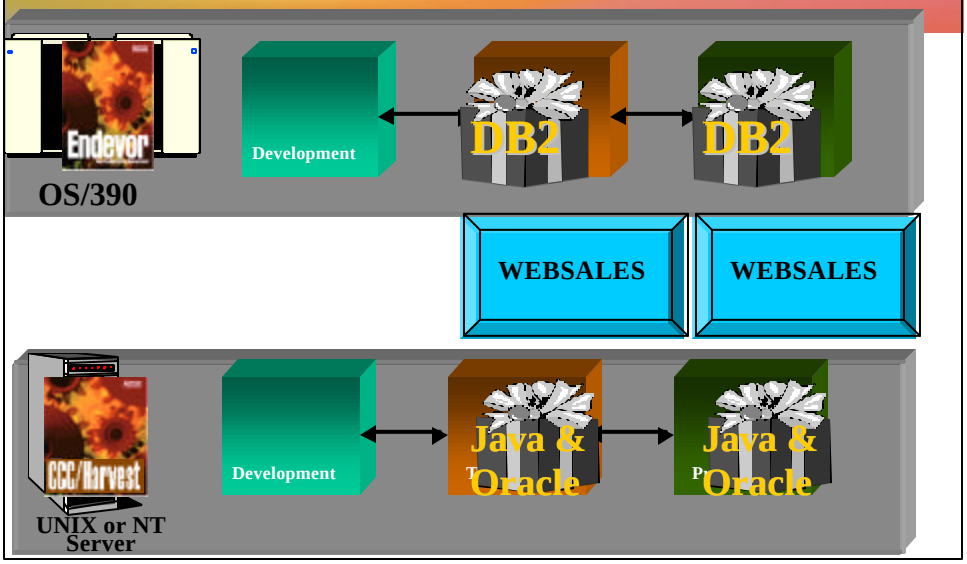

Manual Promotion
Endevor
OS/390
Development
Test - QA
DB2
CCC/Harvest
UNIX or NT Server
Development
Test - QA
Java &
Oracle


Manual Demotion
Endevor
OS/390
Development
DB2
Production
OUT OF SYNC!!!
CCC/Harvest
UNIX or NT Server
Development
Test - QA
Java &
Oracle


Manual Demotion
Endevor
OS/390
Development
DB2
Production
CCC/Harvest
UNIX or NT Server
Development
Java & Oracle
Production

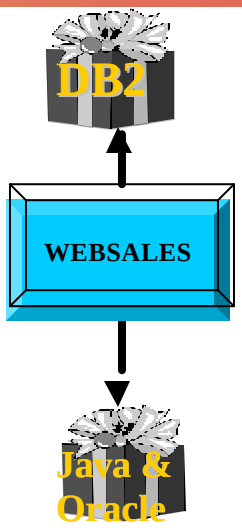

Enterprise Release Automation: Promotion
Endevor
OS/390
DB2
DB2
DB2
WEBSALES
WEBSALES
WEBSALES
CCC/Harvest
UNIX or NT Server
Java & Oracle
Java & Oracle
Java & Oracle

Enterprise Release Automation: Demotion



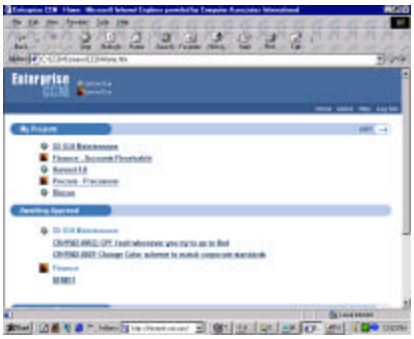
Enterprise CCM: Enterprise Packages

- Establish correlation between Endevor and CCC/Harvest packages
- Enforce approvals on all platforms
- Automate movement through the established lifecycle

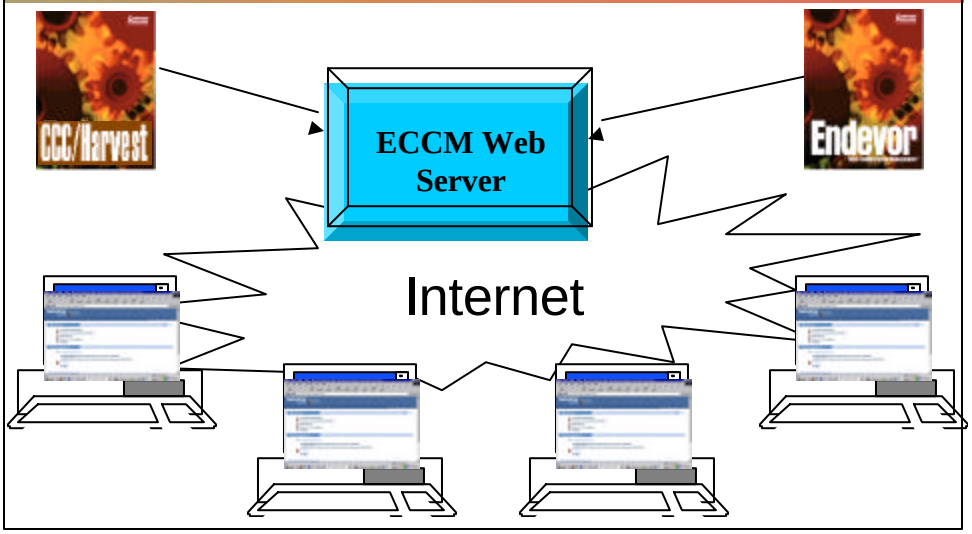


Enterprise CCM: Web Interface

- Intuitive for all types of users
- Accessible from anywhere
- Eliminates need for CCM expertise for non-technical users, e.g. approvers



Enterprise CCM: Architecture

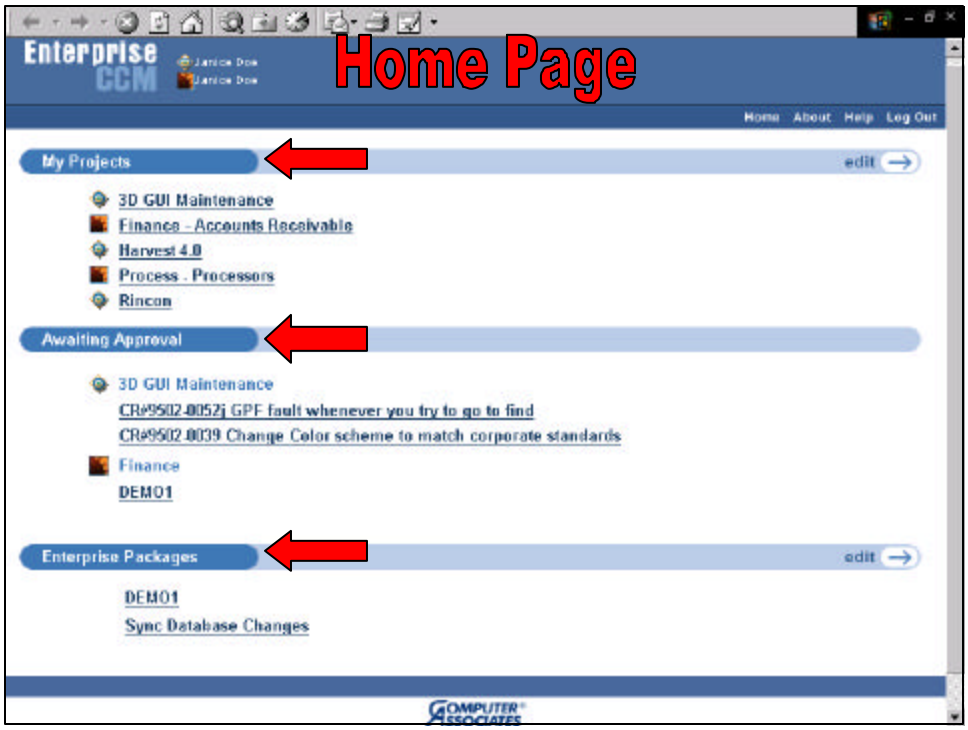


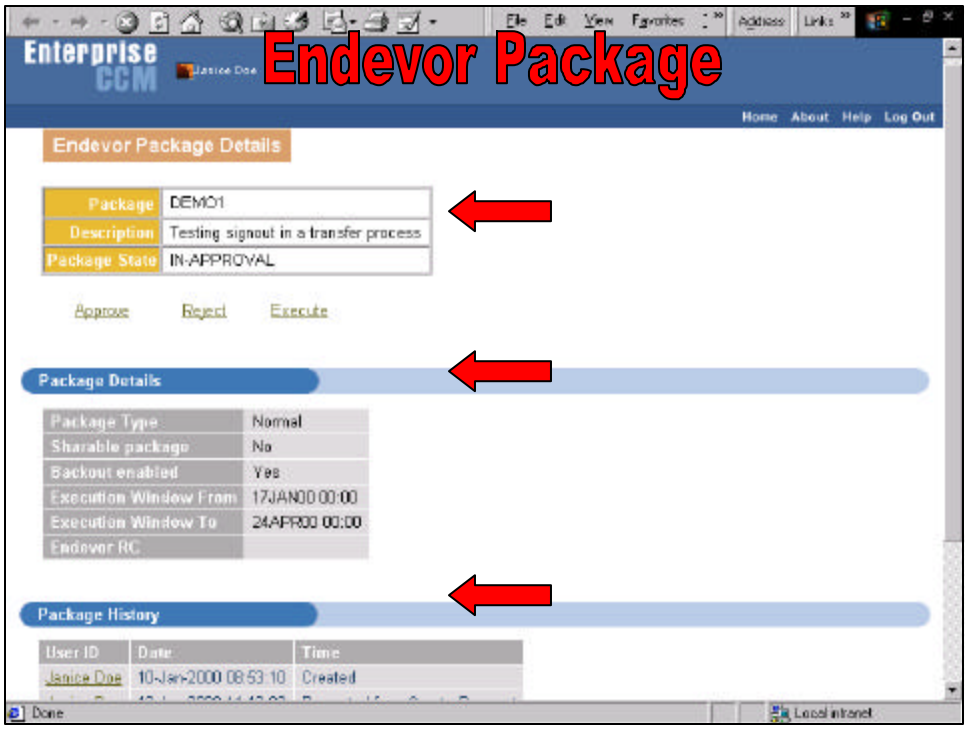
Enterprise CCM: Benefits

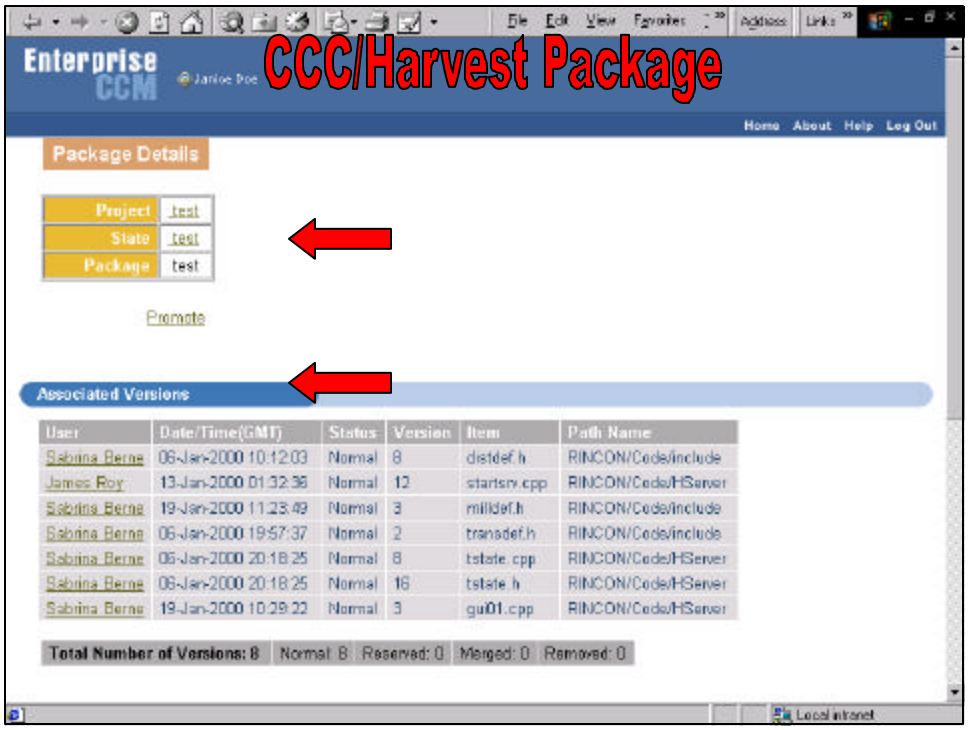
- Enterprise-wide view of the multi-tier application
- Elimination of manual errors
  - Save time and money
  - Improve quality and competitiveness
- Synchronized deployment across platforms
- Integration of tools and processes

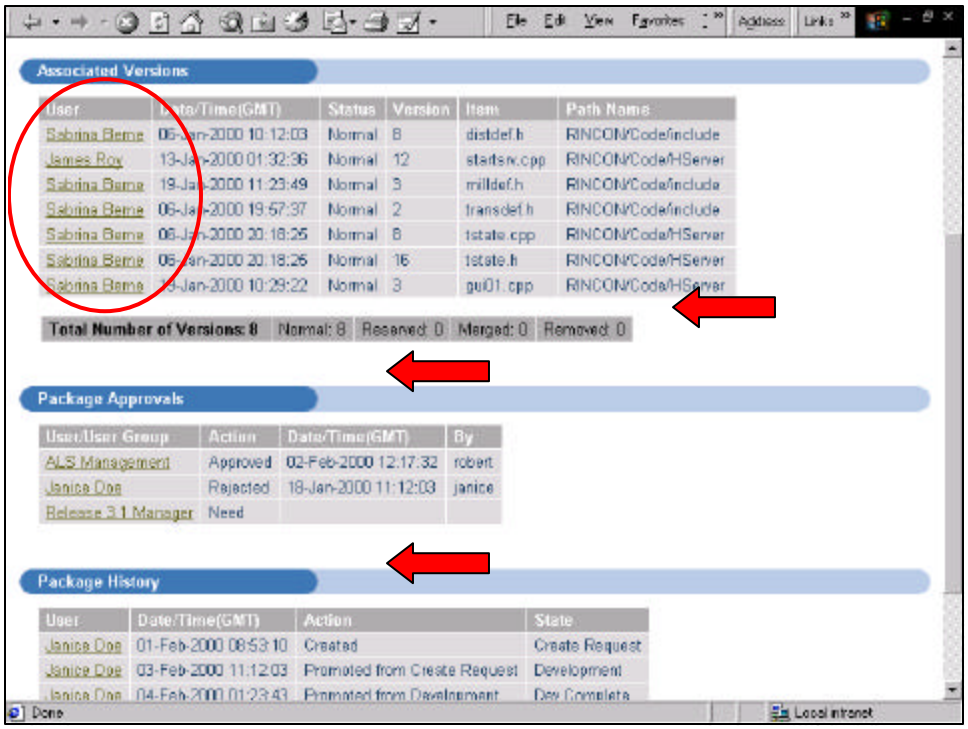
Enterprise CCM: Benefits

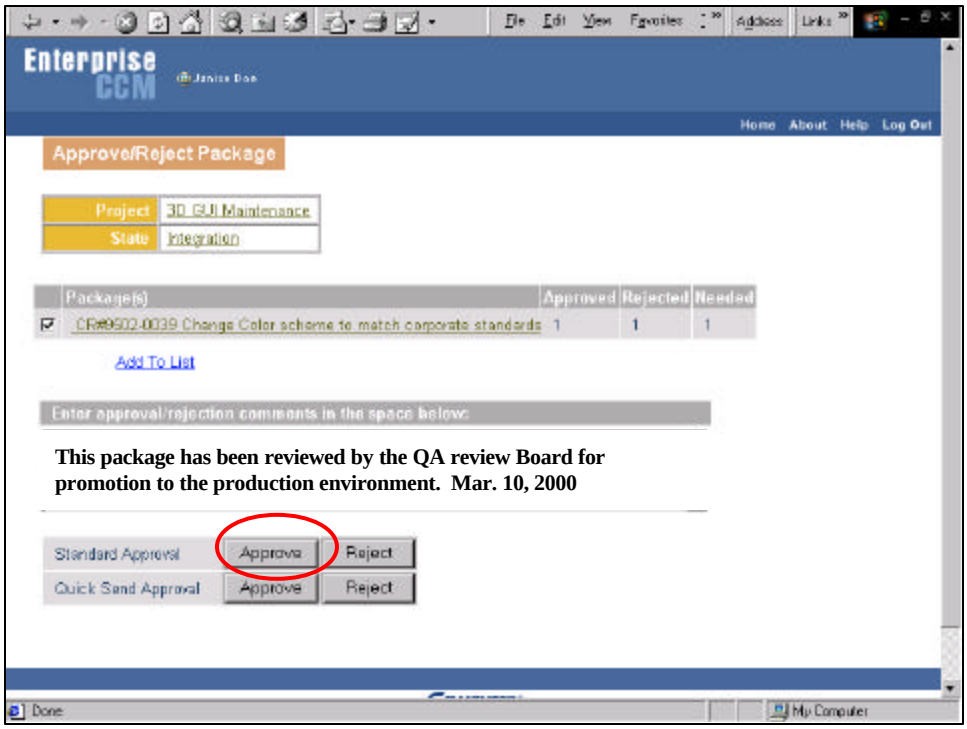
- Easy access via the Web
- Familiar and intuitive interface
  - Ease of use
  - Minimize learning curves
- Simplified administration
- Central management

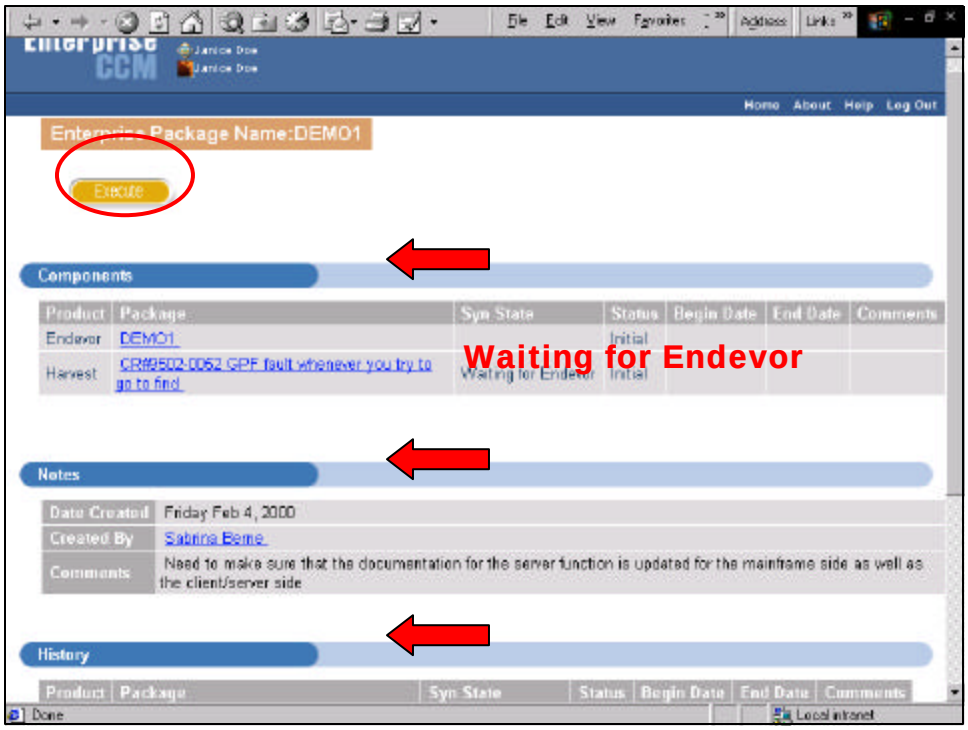

Home Page
Enterprise CCM
Home About Help Log Out
My Projects
edit
3D GUI Maintenance
Finance - Accounts Receivable
Harvest 4.0
Process - Processors
Rincon
Awaiting Approval
3D GUI Maintenance
CR#9502.0052j GPF fault whenever you try to go to find
CR#9502.0039 Change Color scheme to match corporate standards
Finance
DEMO1
Enterprise Packages
edit
DEMO1
Sync Database Changes
COMPUTER ASSOCIATES


Endevor Package
Enterprise CCM
Home About Help Log Out
Endevor Package Details
Package DEMO1
Description Testing signout in a transfer process
Package State IN-APPROVAL
Approve Reject Execute
Package Details
Package Type Normal
Sharable package No
Backout enabled Yes
Execution Window From 17JAN00 00:00
Execution Window To 24APR00 00:00
Endevor RC
Package History
User ID Date Time
Janice Doe 10-Jan-2000 08:53:10 Created
Done
Local intranet


Enterprise CCM
Janice Doe
CCC/Harvest Package
Home About Help Log Out
Package Details
Project test
State test
Package test
Promote
Associated Versions
User | Date/Time(GMT) | Status | Version | Item | Path Name
Sabrina Berne | 06-Jan-2000 10:12:03 | Normal | 8 | distdef.h | RINCON/Code/include
James Roy | 13-Jan-2000 01:32:36 | Normal | 12 | startsrv.cpp | RINCON/Code/HServer
Sabrina Berne | 19-Jan-2000 11:23:49 | Normal | 3 | milldef.h | RINCON/Code/include
Sabrina Berne | 06-Jan-2000 19:57:37 | Normal | 2 | transdef.h | RINCON/Code/include
Sabrina Berne | 06-Jan-2000 20:18:25 | Normal | 8 | tstate.cpp | RINCON/Code/HServer
Sabrina Berne | 06-Jan-2000 20:18:25 | Normal | 16 | tstate.h | RINCON/Code/HServer
Sabrina Berne | 19-Jan-2000 10:29:22 | Normal | 3 | gu01.cpp | RINCON/Code/HServer
Total Number of Versions: 8 | Normal: 8 | Reserved: 0 | Merged: 0 | Removed: 0
Local intranet


Associated Versions
User | Date/Time(GMT) | Status | Version | Item | Path Name
Sabrina Berne | 06-Jan-2000 10:12:03 | Normal | 8 | distdef.h | RINCON/Code/include
James Roy | 13-Jan-2000 01:32:36 | Normal | 12 | startsrv.cpp | RINCON/Code/HServer
Sabrina Berne | 19-Jan-2000 11:23:49 | Normal | 3 | milldef.h | RINCON/Code/include
Sabrina Berne | 06-Jan-2000 19:57:37 | Normal | 2 | transdef.h | RINCON/Code/include
Sabrina Berne | 06-Jan-2000 20:18:25 | Normal | 8 | tstate.cpp | RINCON/Code/HServer
Sabrina Berne | 06-Jan-2000 20:18:25 | Normal | 16 | tstate.h | RINCON/Code/HServer
Sabrina Berne | 19-Jan-2000 10:29:22 | Normal | 3 | gu01.cpp | RINCON/Code/HServer
Total Number of Versions: 8 | Normal: 8 | Reserved: 0 | Merged: 0 | Removed: 0
Package Approvals
User/User Group | Action | Date/Time(GMT) | By
ALS Management | Approved | 02-Feb-2000 12:17:32 | robert
Janice Doe | Rejected | 18-Jan-2000 11:12:03 | janice
Release 3.1 Manager | Need
Package History
User | Date/Time(GMT) | Action | State
Janice Doe | 01-Feb-2000 08:53:10 | Created | Create Request
Janice Doe | 03-Feb-2000 11:12:03 | Promoted from Create Request | Development
Janice Doe | 04-Feb-2000 01:23:43 | Promoted from Development | Dev Complete
Done
Local intranet


Enterprise CCM
Home About Help Log Out
Approve/Reject Package
Project 3D GUI Maintenance
State Integration
Package(s) Approved Rejected Needed
CR#9902-0039 Change Color scheme to match corporate standards 1 1 1
Add To List
Enter approval/rejection comments in the space below:
This package has been reviewed by the QA review Board for promotion to the production environment. Mar. 10, 2000
Standard Approval Approve Reject
Quick Send Approval Approve Reject
Done
My Computer


Enterprise CCM
Home About Help Log Out
Enterprise Package Name:DEMO1
Execute
Components
Product Package Syn State Status Begin Date End Date Comments
Endevor DEMO1 Initial
Harvest CR#9502-0052 GPF fault whenever you try to go to find Waiting for Endevor Initial
Waiting for Endevor
Notes
Date Created Friday Feb 4, 2000
Created By Sabina Berne
Comments Need to make sure that the documentation for the server function is updated for the mainframe side as well as the client/server side
History
Product Package Syn State Status Begin Date End Date Comments
Done
Local intranet

Summary

- Multi-tier application development and deployment
- Features, architecture and benefits of Enterprise CCM
- Tour of Enterprise CCM

Questions and Answers

QW2000 Paper 2T1

Mr. Michael Silverstein
(SilverMark, Inc.)

Automating Testing of Object-Oriented Components Using Intelligent Test Artifacts

BACK TO QW2000 PROGRAM

Key Points

- Common patterns of component/object test automation
- How intelligent test artifacts as an architectural foundation can address common patterns of component/object testing, speeding test development and reducing test maintenance
- Implementing intelligent test artifacts in Java

Abstract

Component testing is the act of subdividing an object-oriented software system into units of particular granularity, applying stimuli to the component's interface and validating the correct responses to those stimuli, in the form of either a state change or reaction in the component, or elsewhere in the system.

In the last few years, design patterns have become a currency for communicating common problems and their solutions within a context. Testing is no exception. Several publications have endeavored to document common design solutions to component test automation problems in terms of design patterns.

This article presents a set of design patterns commonly encountered when creating automated test frameworks and application domain specific test cases, and introduces the notion of ***test artifacts*** as an abstract test component implementation model.

For most types of component testing, the component test developer must eventually find answers to one or more of the following questions, and possibly the questions that their answers pose:

- How do I create a specifically configured instance of a component under test?
- How to I manage the application of a stimulus to the component under test?
- How do I organize stimuli into reusable groups?
- How to I validate the state of the component and system under test after one or more stimuli?
  - How do I manage reference objects?
    - How do I compare references objects to target objects, so that just the states that are important are compared?
  - How do I intelligently traverse the state of my object under test
- How do I manage variations on test inputs in order to drive different paths with the same test

stimuli?

- How do I verify that expected exceptions are raised?
- How do I map development activities to the tests that validate their correctness?

The presentation describes several patterns for automated component test framework and test case design, some familiar and some not, and introduces the notion of ***test artifacts*** as a component architecture for implementing tests. Examples use the **Java**ä programming language. These patterns include:

- **Test Component** - Implement a reusable component model to represent aspects of automated testing.
- **Test Sequence** - Present a complete or commonly implemented subset of a test as a discrete, reusable unit.
- **Configured Instance** - Provide an instance of an object configured to a particular state
- **Test Stimulus** - Encapsulate a stimulus/response validation operation for a component under test for singly or multiply threaded systems.
- **State Validation** - Assert that a component is in the expected state. Define this assertion as a reusable component.
- **Impression** - Express a particular view of the state of a component in a compact, but human readable form
- **Nested State** - Provide a mechanism for specifying and traversing a chain of object references to extract a specific state
- **Aggregate State** - Represent the state of a component in terms of an aggregate set or subset of the states of its subcomponents.
- **Object Variation** - Organize configured object variations

About the Author

Michael Silverstein is Lead Architect at SilverMark, Inc. (www.silvermark.com) with responsibility for technical direction for SilverMark's Test Mentor™ line of object-oriented automated testing tools for Java and Smalltalk. Michael frequently advises customers on how to design automated tests, and has been developing software for twenty years, the last eight of which using object-oriented architectures and languages, including Smalltalk and Java™.

Before SilverMark, Michael worked on a variety of software development projects at IBM, including the VisualAge development team. Michael can be reached at msilverstein@silvermark.com (919) 858-8300 x29

BACK TO QW2000 PROGRAM

Automated Testing of Object-Oriented Components using Intelligent Test Artifacts

[Session **2T1**]

Michael Silverstein
SilverMark, Inc.

www.silvermark.com
contact: msilverstein@silvermark.com
919 858-8300 x29

Copyright 2000 SilverMark, Inc. 1

Component Test Frameworks

- Level 0: Self tests, usually in static methods
- Level 1: Test cases implemented as methods in harness classes.
- Level 2: Basic execution framework (J-Unit, for example)
  - TestSuite, TestCase, TestResult
- Level 3: Execution framework includes objects that encapsulate test specific behavior, promotes task-configured reusable test components.

Copyright 2000 SilverMark, Inc. 2

Test Design Questions

- How do I create a specifically configured instance of a component under test?
- How to I manage the application of a stimulus to the component under test?
- How do I organize stimuli into reusable groups?
- How to I validate the state of the component and system under test after one or more stimuli?
  - **How do I manage reference objects?**
  - **How do I compare references objects to target objects, so that just the states that are important are compared?**
  - **How do I intelligently traverse the state of my object under test**
- How do I validate that expected exceptions are raised.
- How do I validate that expected events are fired?
- How do I manage variations on test inputs in order to drive different paths with the same test stimuli?
- How do I map development activities to the tests that validate their correctness?

Copyright 2000 SilverMark, Inc. 3

Pattern Responses to Design Questions

- **Test Component** - Implement a reusable component model to encapsulate configurable aspects of automated testing.
- **Test Sequence** - Present a complete or commonly implemented subset of a test as a discrete, reusable unit.
- **Configured Instance** - Provide an instance of an object configured to a particular state
- **Test Stimulus** - Encapsulate a stimulus/response validation operation for a component under test for singly or multiply threaded systems.
- **Nested State** - Provide a mechanism for specifying and traversing a chain of object references to extract a specific state
- **Aggregate State** - Represent the state of a component in terms of an aggregate set or subset of the states of its subcomponents.
- **State Validation** - Assert that a component is in the expected state. Define this assertion as a reusable component.
- **Object Variation** - Organize configured object variations

Copyright 2000 SilverMark, Inc. 4

Test Component

- **Intent:** Implement a reusable component model to represent aspects of automated testing, of arbitrary granularity.
- **Context:** Move common test code out of test cases and into a framework in a way that supports creation of reusable test components.
- **Strategy:** Create a hierarchy of classes called ***artifacts***, each of which embodies specific test execution behavior, as well as the properties required to configure instances for specific applications of the artifact.
  - A specification for an artifact describes a specific configuration of an artifact instance.

Copyright 2000 SilverMark, Inc. 5

Test Artifact - Properties

- **Name** – Used to identify and locate an artifact.
- **Description** – More extensive descriptive information about the artifact, either as text or a reference to some external document (URL).
- **Classification of Intent** – Specifies intent of artifact for documentation, (use case scenario, requirements, validation, invariant, etc.)
- **Expected exception** – A reference to an exception that is expected to be signaled as a result of execution of the artifact. Execution of the artifact is said to have failed if the exception is not raised.
- **Precondition action** – Actions to set and/or verify the state of the system under test prior to further execution of the artifact.
- **Invariant action** – A set of assertions that express general consistency constraints that apply to every class instance as a whole, regardless of stimulus. If specified, this is executed after execution of the artifact.
- **Synchronization** – Strategy for determining when a response to a stimulus has completed
  - Message completed/returned [synchronous]
  - Event(s) fired [synchronous or asynchronous]
  - Miscellaneous system state change [asynchronous]

Copyright 2000 SilverMark, Inc. 6

Test Artifact - Consequences

- Abstracting typical testing patterns into a palette of reusable, individually configurable component types that embody common testing behavior enables test developers to focus less on test implementation and more on the problem of exercising the system under test.
- Artifacts offer arbitrary levels of test organization beyond the traditional class, method, or statement level.
- Test artifacts support either a bottom-up or top-down test development process.
- Test artifacts benefit test tool writers because artifacts present a facade for complex behavior, and artifacts may be combined to form a complete representation of a test which may then be presented for editing by a tool user interface.
- Test development speed increases as more reusable components become available.
- Defining test artifacts in one and only one place eases maintenance because changes to test code due to changes in the system under test are localized.

Copyright 2000 SilverMark, Inc. 7

Test Sequence

- **Intent:** Present a sequence of test actions of arbitrary granularity as a discrete, reusable unit, where test actions may themselves be references to predefined reusable components.
- **Context:**
  - **A prerequisite for creating tests via composition of components is a container in which to hold them.**
  - **Fine-grained executable units enable automated testing tools to provide better definition UIs, better tracking of test execution progress, and more detailed reports.**
- **Strategy:**
  - **Create a composite *test sequence* artifact composed artifact specifications.**
  - **The execution behavior: execute each components in order**
- **Consequences:**
  - **The test sequence artifact encourages construction through composition.**
  - **Because composite artifacts are responsible for sequencing through their components during execution, they may also embody general execution rules, such as ceasing execution on failure.**

Copyright 2000 SilverMark, Inc. 8

Configured Instance

- **Intent:** Provide an instance of an object configured to a particular state.
- **Context:** Creating supporting objects is a necessary, and often distracting, evil of component testing. Supporting objects are required as:
  - inputs to individual stimuli (message parameters)
  - comparison targets (gold standard objects)
  - starting points for sequences of stimuli for testing components in a specific state

Copyright 2000 SilverMark, Inc.

Configured Instance

- **Strategy:** Create an artifact whose purpose is to return an object in a particular state.
  - Approaches:
    - **Instantiation: Create a configured instance of a class based on:**
      - **The class to instantiate**
      - **A constructor method and parameters**
      - **Additional stimuli**
    - **Materialization: Retrieve serialized object from some table/database/file, then apply additional stimuli as necessary.**
  - A configured instance may be implemented as a test sequence or test sequence subclass
    - **Subclass would have additional specification for instantiation or retrieval of object, before applying additional stimuli.**
- **Consequences:** Over time, test developers accumulate a library of supporting object instances that they may readily apply to test actions.

Copyright 2000 SilverMark, Inc.

Copyright 2000 SilverMark, Inc.

Test Stimulus

- **Intent:** Encapsulate a stimulus/response validation operation for a component under test for singly or multiply threaded systems
- **Context:**
  - The basic unit of execution of a test case is the application of a stimulus.
  - The response to a stimulus should be validated in terms of a change in the state of the component under test, or some area of the system under test, as well as an object returned directly by the stimulus.
  - In multithreaded systems, the response to a stimulus may be temporally non-deterministic

Copyright 2000 SilverMark, Inc. 11

Test Stimulus - Strategy

Create an artifact that embodies any or all of:

- A stimulus on some object, defined as one of:
  - a reference to a target object and the action to perform on the target object (message send), as well as any arguments. References to configured instance artifacts serve well for acquiring argument values.
  - the selector for a method within a supporting test class
  - a reference to an artifact
  - other executable representation of the stimulus (Java ™ inner class, Smalltalk block)
- precondition state validation
- postcondition state validation
- invariant validation
- a synchronization mechanism for suspending test execution until an asynchronous process has completed, or time limit is exceeded

Copyright 2000 SilverMark, Inc. 12

Nested State

- **Intent:** Provide a mechanism for specifying and traversing a recursive chain of object references to extract a specific state.
- **Context:** Complex objects and object clusters, especially implementations of the façade pattern, conceal recursively held collaborating objects whose individual states contribute to the whole, and whose access is often required for state validation during testing.
- **Strategy:**
  - **Implement methods on the root class to provide nested state**
    - **Adds code bloat to delivered code for the sake of testability**
  - **Implement an code that returns provides a means to specify an access scheme for a particular state of a given object. For example,**
    `return myObject.x().y().z();`
  - **Configure a composite artifact to contain a sequence of stimuli to be used to access successive states in the chain.**
    - **Less straightforward, but presents opportunity to build off of a state returned by an artifact reference.**
- **Consequences:**
  - **A configured nested state artifact is a permanent, reusable access strategy for a particular state within a component.**
  - **Maintenance is localized**

Copyright 2000 SilverMark, Inc. 13

Aggregate State

- **Intent:** Represent the state of a component in terms of an aggregate of some subset of the states of its subcomponents.
- **Context:** The recursive state of an object is the value set of instance variable values obtained by recursively expanding each instance variable value
  - A component has one *complete* recursive state, but may have many aggregate state representations, or *views*, depending on which states are of interest
- **Strategy:**
  - Populate a composite ***aggregate state*** artifact with components that, when executed, extract individual states from the object under consideration.
    - ***Nested state*** **artifacts are a prime candidate for population**
    - **Execution time behavior gathers and returns a collection of resultant execution values of each of its components**
    - **Implement comparison operators that take into account the complete set of states, like `equals()`**

Copyright 2000 SilverMark, Inc. 14

Aggregate State

- A configured aggregate state artifact acts as a reusable representation of a particular view of an object's state.
- Useful shorthand for comparing objects:

Typical aggregate state comparison

```
return customer.x().equals(referenceCustomer.x()) &
   customer.x().y().equals(referenceCustomer.x().y()) &
        customer.x().y().z().equals(referenceCustomer.x().y().z())
… etc. for each state
```

Artifact aggregate state comparison

```
(new MyArtifacts().getArtifactNamed(
  "General Customer State")).subject(customer).equals(
     (new MyArtifacts().getArtifactNamed(
       "General Customer State")).subject(referenceCustomer));
```

Copyright 2000 SilverMark, Inc.

State Validation

- **Intent:** Assert that a component is in an expected state. Define this assertion as a reusable component.
- **Context:**
  - State validation involves comparing an object's recursive state to some known, expected state, encoded as:
    - a comparison to one or more *Gold Standard* reference objects
    - simple comparisons to primitives or literals
    - a test for some known object's inclusion or exclusion within a collection type
    - a comparison between object impressions

Copyright 2000 SilverMark, Inc.

Copyright 2000 SilverMark, Inc.

State Validation - Strategy

- A common strategy: implement an `assert(boolean)` test API to log results of executing an expression that logs the success or failure of a contained expression, during test execution.
  - Simple, flexible, but not very reusable
- Use common postcondition validation action test artifact property:
  - Close, structural relationship between stimulus and response validation enhances understanding of the test. Define a validation action as:
    - **an expression that evaluates to a boolean. This is similar to the assert API discussed above**
    - **a reference to an artifact that performs the validation.**
    - **Where the same validation is performed repeatedly (invariants, for example) defining and then referencing an artifact is advantageous, especially if many state comparisons are involved.**
    - **implement an *aggregate state* artifact as a comparison wrapper, implementing `equals()`**

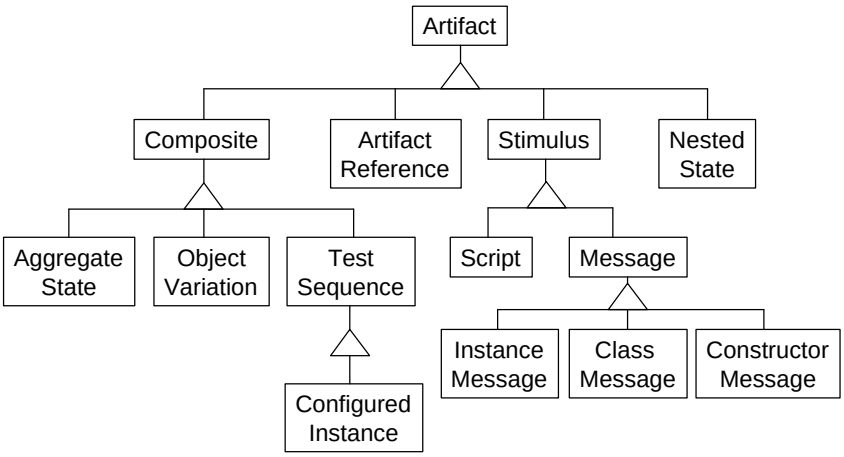
Copyright 2000 SilverMark, Inc. 17

Object Variation

- **Intent:** Organize variations of configured object instances as a single, reusable resource
- **Context:** Need to be able to pass input variations of test objects to a test for the purpose of driving execution of the different paths (data driven tests)
- **Strategy:**
  - Implement an artifact that presents an enumeration interface for sequentially accessing its elements, whereby execution of a given test element is repeated over the elements returned by the object variation artifact.
    - **Statically described references to configured object instances**
    - **Objects dynamically generated by rules**
      - **for example: an Integer interval or a calculated equivalence set**
  - For multiple variation artifacts, implement pluggable value mixing policy (one-by-one, each-with-every, etc.).
- **Consequences:** Useful variations are themselves reusable units that may shared among test developers.

Copyright 2000 SilverMark, Inc. 18

Example Artifact Hierarchy



Copyright 2000 SilverMark, Inc. 19

Example code

```
/* Return a configured instance of a Calculator class after adding an int. */
  public ConfiguredInstance calculatorAfterAdd() {
  ConfiguredInstance artifact =
      new ConfiguredInstance(“Calculator after add");
  // Set subject of step via constructor specification
  artifact.setConstructorStep(
      new Stimulus(“Default").setMessageSpec(message(Calculator.class)));
  // Apply stimulus to subject via MessageStep
  artifact.addComponent(new Stimulus(“Add number").setMessageSpec(
                                    message(subject(),"add",new int[] {123}));
  return artifact; }
```

```
/* Artifact that references an artifact */
public ConfiguredInstance calculatorAfterSubtract() {
  ConfiguredInstance artifact =
      new ConfiguredInstance(“Calculator after subtract");
  ArtifactReference reference =
      new ArtifactReference(“Calculator after Add”,MyClass,”calculatorAfterAdd”);
  artifact.setConstructorStep(reference);
  // Apply stimulus to subject via MessageStep
  artifact.addComponent(new Stimulus(“Subtract number").setMessageSpec(
                                 message(subject(),“subtract",new int[] {123}));
  return artifact; }
```

```
new MyClass().calculatorAfterAdd().getValue() // Execute by itself
```

Copyright 2000 SilverMark, Inc. 20

Component Testing with Intelligent Test Artifacts

Michael Silverstein
SilverMark, Inc.

msilverstein@silvermark.com
http://www.silvermark.com
(919) 858-8300 x29

Component testing is the act of subdividing an object-oriented software system into units of particular granularity, applying stimuli to the component's interface and validating the correct responses to those stimuli, in the form of either a state change or reaction in the component, or elsewhere in the system.

Often, component testing is performed by developers who's primary mental focus is that of the system under test and component tests to be performed. In most cases, the test case architecture, if any, evolves as an afterthought, driven mostly by the tests themselves. Some developers who have been through this process more than once write their own test frameworks.

In the last few years, design patterns have become a currency for communicating common problems and their solutions within a context. Testing is no exception. Publications such as [Binder 99, McGregor 99, Firesmith 96, Beck 94] have documented some common design solutions to component test automation problems in terms of design patterns.

This article presents a set of design patterns commonly encountered when creating automated test frameworks and application domain specific test cases, and introduces the notion of ***test artifacts*** as a test component architectural model for implementing those patterns.

This article begins by presenting several patterns for automated component test framework design, some familiar and some not, with implementation strategies. It then shifts focus from test framework design to test case design. And briefly applies the solutions discussed earlier as a basis for test case implementation.

Common component test design questions

The challenge to test framework developers is to recognize the common problems and associated implementation activities, to the extent that a scalable and easily maintainable automated test architecture can be applied as a foundation for implementing tests. Our experience has shown that for most types of component testing, the component test developer must eventually find answers to one or more of the following questions, and possibly the questions that their answers pose:

- How do I create a specifically configured instance of a component under test?
- How to I manage the application of a stimulus to the component under test?
- How do I organize stimuli into reusable groups of arbitrary granularity?
- How to I validate the state of the component and system under test after one or more stimuli?
    - How do I manage reference objects?
        - How do I compare references objects to target objects, so that just the states that are important are compared?
    - How do I intelligently traverse the state of my object under test
- How do I validate that expected exceptions are raised.
- How do I validate that expected events are fired?
- How do I manage variations on test inputs in order to drive different paths with the same test stimuli?
- How do I map development activities to the tests that validate their correctness?

Pattern responses to design questions

This section presents component test design patterns, with implementation strategies. An effort was made to keep implementation strategies language neutral, however, wherever use of code is unavoidable examples are presented in the Java™ programming language.

Pattern: Test Component

Intent: Implement a reusable component model to represent aspects of automated testing of arbitrary granularity.

Context

Generic component architectures are not new. For example, Java™ has Java Beans™, and its variants. To date, the model for component tests has been limited to units of gross execution, such as the test suite and test case [Beck 94, Fowler 99], which is useful, but falls short in the role of a general organizational and behavioral principal for automated testing. What is required is a higher level of abstraction, under which specific variants may serve as architectural building blocks for particular aspects of automated testing.

Applicability

- This pattern is indicated at many levels of testing. Test components may express test organization or behavior and are in effect, miniature, automated testing problem specific frameworks.

Strategy/Implementation

- Create a hierarchy of test component classes, called ***test artifacts***[1], each of which embodies specific test execution behavior, as well as the properties required to configure instances for specific applications of the artifact.
- Artifacts may be singular or composite. The specification for a composite artifact contains specifications for other artifacts.
    - A specification for an artifact indicates a specific configuration of an artifact instance, that is, an artifact instance whose properties are set in a certain way. The combination of an artifact type and the specific configuration of an instance of it results in a reusable component whose behavior is applicable to a certain test-related task.
    - Reusability is achieved with the creation of an artifact type that can locate a specifically configured artifact instances based on some well known identifier. An artifact reference is itself, a specifically configured artifact instance.
    - Composite artifacts execute by executing each contained artifact, in turn.
- Common artifact configuration properties include:
    - **Name** – Used to identify an artifact so that it can be located.
    - **Description** – More extensive descriptive information about the artifact, either as text or a reference to some external document.
    - **Expected exception** – A reference to an exception that is expected to be signaled as a result of execution of the artifact. Execution of the artifact is said to have failed if the exception is not raised.
    - **Precondition action** – Actions to set and/or verify the state of the system under test prior to further execution of the artifact.
    - **Invariant action** – A set of assertions that express general consistency constraints that apply to every class instance as a whole, regardless of stimulus. If specified, this is executed after execution of the artifact.
    - **Synchronization action** – An action that checks for some state transition that indicates the completion of some asynchronous transaction. In the case of an artifact that initiates an asynchronous transaction, the intent of the synchronization action would be to is to pause execution of a test until the transaction has completed.
    - **Classification of organizational intent** – such as, use case scenario [Jackobsen 93], requirements implementation, fix for known problem, etc.

[1] The name, artifact, is chosen as an alternative to the name, *test component* to avoid confusion between *test component* and *component under test*.

- The specification for an artifact's configuration should be defined one and only one place, to ensure that multiple definitions of equivalent artifacts do not become a maintenance problem.

Consequences

- Abstracting typical testing patterns into a palette of reusable, individually configurable component types that embody common testing behavior enables test developers to focus less on test implementation and more on the problem of exercising the system under test.
- Generally, automated component test frameworks offer class or method level granularity to test developers. Artifacts offer arbitrary levels of test organization beyond the traditional class or method level. One or more specifically configured test artifacts may represent a complete test. At minimum, a complete test may be as simple as instantiating a single class, or it may incorporate multiple artifacts as building blocks.
- Test artifacts support either a bottom-up test development process, where basic reusable artifacts are created first as components to be used in more broadly scoped artifacts, or a top-down development process, where broadly scoped artifacts reference utility artifacts that are not yet completely defined.
- Test artifacts benefit test tool writers because artifacts present a facade for potentially complex objects with a high degree of functionality, and places test behavior in the test framework instead of the test cases.
- As in most systems that employ a high degree of reuse, test development speed increases as more reusable components become available.
- Defining test artifacts in one and only one place eases maintenance because changes to test code due to changes in the system under test are localized.

Known Uses

There are many precedents for implementing reusable components outside of the automated component testing domain.

Pattern: Test Sequence

Intent: Present a sequence of test actions of arbitrary granularity as a discrete, reusable unit.

Context

A prerequisite for creating tests via composition of components is a container in which to hold them. Because of the sequential nature of tests, this container must be a composite of sequentially executed components. It is also extremely helpful to be able to subdivide tests into conceptual units for the purpose of enhancing readability and understandability.

Fine-grained executable units enable automated testing tools to better track test execution progress, and enable the tools to provide better test editing interfaces for users by allowing different task specific views of individual actions to be performed.

Applicability

This pattern is indicted wherever test creation through composition is to be employed, or where it is advantageous to subdivide a test into sequentially executed units.

Strategy/Implementation

Create a composite ***test sequence*** artifact that is composed of specifications for other artifacts. The execution behavior of this artifact is to execute each of its components in a well defined order under specified rules. For example, one may implement an **abort on failure** rule to provides a convenient shorthand for implementing a scheme where execution ceases on the first component failure encountered. Instead of repeatedly coding a check-and-return-if-fail after each component's execution, the artifact embodies the intelligence to manage the execution flow as specified by the rule.

Consequences

By simply extending the test artifact concept to include a composite artifact type, test creation through composition is achieved.

Because composite artifacts are responsible for sequencing through their components during execution, they may also embody general execution rules, such as ceasing execution on failure. Execution rules such as this are often used to dictate the level of granularity of the actions to be performed within the sequence.

Pattern: Configured Instance

Intent

Provide an instance of an object configured to a particular state.

Context

One of the foundations of component testing is the use of objects in a particular state as:

- inputs to individual stimuli (parameters)
- comparison targets (gold standard objects)
- test case starting state

During the process of writing component test code, creating instances of test input objects is a tangential activity that is distracting from the overall goal of

stimulating the component under test, and often where a majority of the test development effort is expended.

Unless the interface for an object is accessible exclusively via a global, static namespace, it is likely that instances of the object, and the objects that it recursively contains, must be instantiated or extracted from persisted location.

Furthermore, there are usually multiple ways of instantiating an object, so there needs to be a way of encapsulating the appropriate instantiation strategy for the component so it can be easily referenced and applied to a test. For example, Java and C++ classes may offer multiple constructors for instantiation. Smalltalk offers a much looser interface. This pattern offers a solution to encapsulating instantiation or materialization of test objects as a reusable operation.

Strategy

A good, basic approach is to implement methods in some test class that return specifically configured instances of test objects. This test class is sometimes referred to as an Oracle [Binder 99].

A further abstraction would be to implement an artifact that, given a specification, returns a configured object instance. This may be implemented as an extension of the *test sequence* composite artifact, with the additional behavior of initially instantiating or retrieving the object under consideration. Once an object is present, it may be immediately returned or first acted upon by one or more actions to be performed in sequence.

Instantiation approach

This approach assumes instantiation of an object of some class. The specification requires:

- The class to instantiate
- A constructor method. The constructor may require recursive references to other configured objects, as in [Siepmann 94].
- Additional stimuli to apply to the newly instantiated object, possibly in the form of references to one or more existing scenarios.

The artifact needs to implement behavior to instantiate the object according to the specified constructor, recursively requesting any artifacts specified for constructor parameters to provide values, execute any additional stimuli on the object, and return it.

Materialization approach

This approach assumes the object exists in some persistent state and may be retrieved via a given strategy. The specification requires:

- The location (table/database/file) of the object

The artifact needs to implement behavior to retrieve the object, most likely through some framework interface, execute any additional stimuli on the object, and return it.

Known Uses

ObjectCreator in [Siepmann 94] implements a strategy for creating instances of test objects as specified by an object description language.

Related Patterns

This pattern is in many ways similar to the Builder pattern, which separates the construction of an object from its representation. Configured Instance certainly separates construction from implementation, however, the focus of this pattern is to encapsulate any operations that are required to provide a configured instance of an object, which may include implementation of the Builder pattern, or it may simply require materializing an externally persisted object.

Pattern: Test Stimulus

Intent: Encapsulate a stimulus/response validation operation for a component under test for singly or multiply threaded systems.

Context

- The basic unit of execution of a test case is the application of a stimulus.
- The next level of organization for testing contains multiple discrete stimuli for a particular purpose – a use case scenario.
- Optionally, the response to a stimulus may be validated in terms of a change in the state of the component under test, or some area of the system under test, as well as an object returned directly by the stimulus.
- In multithreaded systems, the responses to a stimulus may be temporally non-deterministic.

Applicability

This pattern is indicated in any situation where a discrete stimulus or related sequence of stimuli need to be applied to a component, with response validation.

Strategy/Implementation

Create an artifact that embodies any or all of:

- A stimulus on some object defined as one of:
    - a reference to a target object and the action to perform on the target object (message send)

    - The selector for a method on a test class that will perform the stimulus action when requested
    - A reference to an artifact that will perform the stimulus action when requested
    - Some other executable representation of the stimulus
- precondition state validation
- postcondition state or return value validation
- invariant validation
- a synchronization mechanism for suspending test execution after executing the stimulus, until an expected state is sensed, or time limit is exceeded so that long running asynchronous transactions do not

Instances of implementors of this pattern are typically components of *test sequence* artifacts or test sequence artifacts themselves. As such, validation properties may be specified as references to validating artifacts. Precondition, postcondition and invariant validation, as well as state synchronization are all specified under the general definition common to all test artifacts, so no special work is required to implement those for this pattern.

Related Patterns

Command in [Gamma+ 95] describes a pattern for encapsulating requests. Test Stimulus differs because it also encapsulates expected response validation, and synchronization.

Pattern: Nested State

Intent: Provide a mechanism for specifying and traversing a chain of object references to extract a specific state.

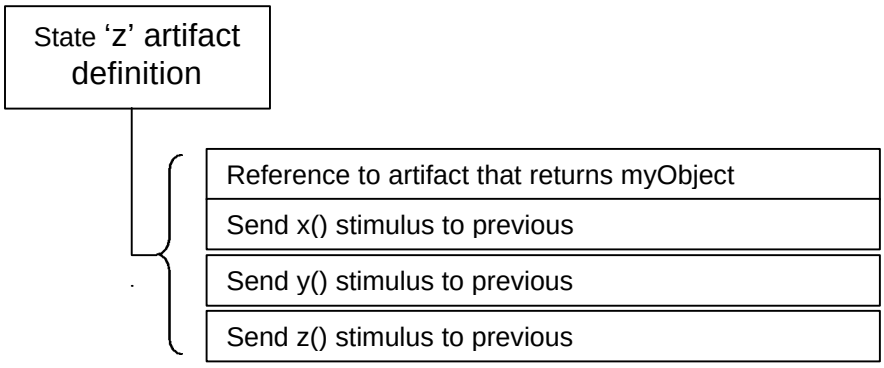
Context

Many components present a primary interface through a single object that typically follows some implementation of the façade pattern, as described in [Gamma+ 95]. If testability were the primary motivating design force, all public sub-object interfaces and states for a component would be promoted to the interface of this façade object so they were easily accessible for validation. Unfortunately, this may lead to code bloat and the need to maintain methods that do not contribute to the overall purpose of the system beyond aiding testability.

Strategy/Implementation

Implement an artifact that provides a means to specify an access scheme for a particular state of a given object. This scheme may be a reference to a method that implements a daisy chained traversal of the objects, for example, `return myObject.x().y().z();`

Another possibility would be to build on a composite artifact whose sequence of stimuli is used to access successive states in the chain. This is a little less straightforward, however it does present the opportunity to build off of a state returned by an artifact reference. For example:



In the definition for the artifact that returns state 'z', each component applies a stimulus to the result of the execution the previous component.

Consequences

A configured nested state artifact becomes a permanent, reusable access method for a particular state within a component. Tests that need to validate this state need only reference this artifact by its root artifact/name ordered pair, passing the object under consideration. If the path to the state ever changes, only a change in the artifact's configuration is needed.

Known uses

VisualWorks Smalltalk uses what is known as an access path in the ProtocolAdaptor class to describe a path to a particular object from a root object in terms of a collection of message selectors to be sequentially sent to the root object and subsequently the objects returned.

Pattern: Aggregate State

Intent: Represent the state of a component in terms of an aggregate of the states of its subcomponents. An aggregate state may or may not represent the complete, recursive state of the component and all each subcomponent.

Context

Most components, with the exception of primitives, can be described as a federation of one or more objects that are either contained by or collaborate with the component. Broadly speaking, each of these objects contributes its own state to that of the whole. The recursive state of a component is the value set of instance variable values obtained by recursively expanding each instance

variable value [Binder 99]. A component has one *complete* recursive state, but may have many aggregate state representations, or *views*, depending on which states are of interest. For example, one might choose to ignore the current time state of a component for comparisons so as to avoid mismatches due to test execution at different times.

Applicability

This pattern represents a strategy for specifying the subcomponent states of interest, traversing the subcomponents to extract those states, and representing multiple states as a single state.

This pattern is applicable when a single, representation of all or some subset of the entire recursive state of a component is required, such as for validation purposes.

Strategy

Populate a composite **aggregate state** artifact with components that, when executed, extract individual states from the object under consideration. *Nested state* artifacts lend themselves well to this purpose. The execution time behavior of this aggregate state artifact is to gather and return a collection of the resultant execution values each of its components as a flattened representation of the total aggregate state of the object under consideration.

Implement comparison operators that take into account the complete set of states

Consequences

A configured aggregate state artifact acts as a reusable representation of a particular view of an object's state. Used in conjunction with nested state artifacts, the state of an entire object tree may be captured. This type of artifact serves as a useful shorthand for comparing objects. For example, for a particular view of the state a root object and object tree one might normally code something like this in Java™:

```
return customer.x().equals(referenceCustomer.x() &&
     customer.x().y().equals(referenceCustomer.x().y()) &&

     customer.x().y().z().equals(referenceCustomer.x().y().
z())
```

… etc.

Given an aggregate state artifact instance configured for some set of states on an object of a particular type, comparison of an object and a presumably equivalent reference object is a matter of instantiating the artifact for the objects to be compared and then sending equals() to the artifacts.

```
(new MyArtifacts().getArtifactNamed(
   “General Customer State”)).subject(customer).equals(
      (new MyArtifacts().getArtifactNamed(
       “General Customer State”)).subject(referenceCustomer));
```

Note: This example syntax for instantiating an artifact is not mandatory

Known uses

Any object that externalizes referenced or contained object states through delegation

Related Patterns

- The Memento pattern captures and externalizes an object’s internal state [Gamma+ 95].
- Implementations of this pattern make use of implementations of Nested State for specifying and extracting states.
- The façade pattern [Gamma+ 95] presents a single interface for a cluster of objects.

Pattern: State Impression

Intent Express a particular view of the state of a component in a compact, but human readable form

Context

By and large, simple objects like strings and integers are much simpler to compare to one another than complex objects, and also much simpler to capture and store for later reference. Simple objects also have the added benefit of being easily recognizable by the human reader, which often proves advantageous when debugging. It becomes advantageous then, to be able to easily generate a simple representation of a view of a component’s state so that rapid comparison to reference state can be accomplished. A view of a component’s state means the union of some, but not necessarily all the instance state of the objects that comprise the component. The term ***Impression***, is used to mean a simple representation of a complex object’s state, in much the same way that artistic Impressionism conveys an impression of its subject without a detailed rendering.

Applicability

This pattern is indicated in situations where simple object state views are required, typically for comparison or debugging purposes.

Strategy/Implementation

Implement one method for each view of the states of interest. Within each method:

- Employ a strategy for extracting the required states. The aggregate state pattern describes a scheme for performing this.
- Append String representations of those states to a String or Stream, as well as any other syntactic sugar required to make the information more readable, and return.

The aggregate state artifact may be easily extended to include a toString() method that gathers its composite states in a String form. Because artifacts include a name property, it is conceivable that the names of the component states might be used in the impression's String representation.

Known uses

Implementations of impressions include **#printString** and **#asString** in Smalltalk, as well as **toString()** in Java.

Related Patterns

Implementations of this pattern may make use of Nested State for specifying and extracting states.

Pattern: State Validation

Intent: Assert that a component is in an expected state. Define this assertion as a reusable component.

Context

State validation involves comparing object state to some known, expected state. Validation of expected state may be encoded in several ways:

- as a comparison to one or more *Gold Standard* reference objects
- as simple comparisons to primitives or literals
- as a test for some known object's inclusion or exclusion within a collection type
- as a comparison between object impressions

Applicability

- Use this pattern when the state of an object at a particular moment requires validation.
- Use this pattern when scaling up to broadly scoped system-level validations from individual object validations.

Strategy/Implementation

A common strategy is to implement an *assert(boolean)* test API [Beck 99] that logs the success or failure of a contained expression, during test execution. This is a simple, but flexible solution that is applicable to many cases.

A common test artifact property discussed earlier is a postcondition validation action, which embodies a definition for some validation action to be performed on artifact execution completion. This has the advantage of tying the artifact's action to an immediate validation of its success. Having this kind of close, structural relationship between stimulus and response validation enhances understanding of the test, especially if the test framework provides user interface or results reporting that maintains the link between the two.

Possibilities for defining a validation action may be:

- an expression that evaluates to a boolean. This is similar to the assert API discussed above, which would most likely be called under the covers during execution.
- a reference to an artifact that performs the validation. In situations where the same validation is performed repeatedly (invariants, for example) defining and then referencing an artifact whose sole purpose is validation of a particular object type's state becomes advantageous, especially if many state comparisons are involved. A similar alternative, discussed above, is to implement an *aggregate state* artifact and use it as a comparison wrapper for an object under consideration and a reference object.

Consequences

A useful application of validation artifacts is to create a family of them to validate common, expected states. Consider a simple example of a ***Loan*** object with states, new, overdue, and complete, among others. Once artifacts have been created that validate the configurations that these states embody, they may be easily 'plugged in' as postcondition properties, as appropriate. When the internal representation of a state changes, it is a simple matter to update the validation artifact's specification without regard to whether it is referenced in one or one hundred places.

Collaborators

Implementors of the nested state and aggregate state patterns may used to extract state.

Pattern: Object Variation

Intent: Organize configured object variations as a single, reusable resource.

Context

When testing components at the interface level, the path from component input to output often varies depending on input values. Therefore, it is advantageous to be able to pass input variations to a test for the purpose of driving execution of

the different paths, both inter-object and intra-object. Ideally, one would be able to describe a specific set of variations for a particular class, referencing that configuration as needed. Any reference to a set of variations in a test case would automatically result in execution of that that test case across each value.

Strategy/Implementation

Design an ***object variation*** test artifact that presents an interface for enumerating, or sequentially accessing its values.

Values may be provided from references to configured artifacts, calculated values or any other values.

Any composite artifact that implements a sequential access mechanism, and whose components return useful values may serve as an object variation artifact. Components that are likely to return useful values are *configured instance* artifacts, and *stimulus* artifacts.

Another approach would be to implement an artifact that implements a sequential access mechanism (an Enumerator, for example) and generates values based on some particular strategy. Some simpler strategies are generating an Integer interval, or random values. Thanks to encapsulation, as long as an object variation artifact implements the required interface, the source of the objects is irrelevant. The concept may also be extended to support retrieval of objects from a database.

The essence of the solution is to extend the test execution framework to accept an object (object variation artifact) that implements a sequential access mechanism as an execution modifier, whereby execution of a given test element is repeated over the elements returned by the object variation artifact. The variation artifact implements a strategy for generating or retrieving objects, in some cases based on its own particular configuration, In the example of the Integer interval, that may configuration may be an upper and lower bound. For database retrieval, it may be the database, table, and some SQL.

The final implementation consideration concerns the case where multiple variation artifacts are specified. In this case, the elements provided by each of the variation artifacts must be mixed according to some consistent and desirable rule (one-by-one, each-with-every, etc.). The recommended approach is to encode the rules in a hierarchy of policy objects to be used by the execution framework to govern retrieval of variation artifact elements. The advantage to encoding this behavior in a separate class is that instances may be plugged in to the execution framework as appropriate to the given situation, as an execution behavior modifier.

Consequences

Useful variations are themselves reusable units that may be shared among test developers.

Even though any artifact that implements a sequencing interface may serve as an object variation provider, it is important to note that care should be taken that the objects that are returned are appropriate for use and of the correct type.

Related Patterns

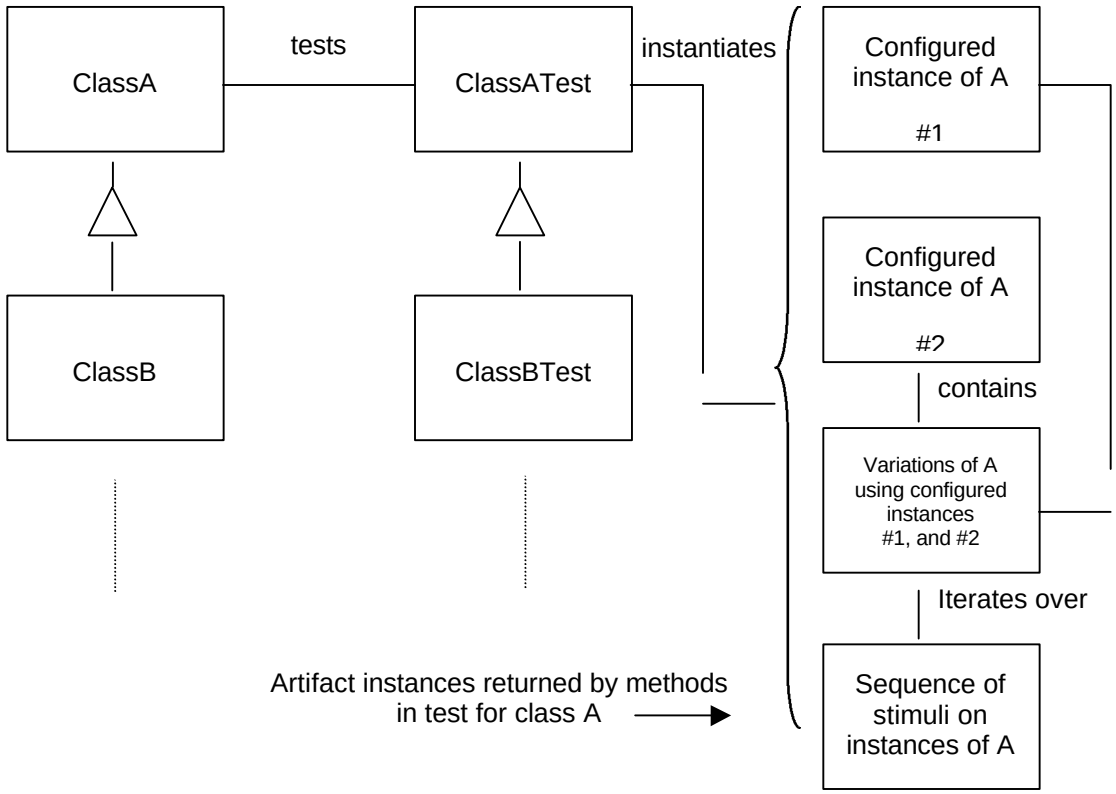
The Oracle pattern [Binder 99] describes approaches for providing test data to tests.

Applying artifacts

This article introduced several test artifact types as responses to common automated component test design patterns. Test artifacts may be used as an architectural foundation for implementing automated component tests. Test artifacts are implemented by classes whose instances embody particular execution behavior. The behavior for an artifact instance is affected by the configuration of its properties.

In practice, artifacts are instantiated, configured and returned by test methods within classes organized by purpose. Traditionally classes known as test suites [Beck 99, Binder 99], provide specifications for test cases and supporting methods. This class-based organization is applicable to artifacts as well, where each artifact is represented by an instance method.

For unit tests, a suite class may appear in a hierarchy designed to mimic the hierarchy of the classes under test [McGregor+96]. For example, consider object, **A**. A parallel test suite class $\mathbf{A_T}$ may be defined for the purpose of testing **A,** with subclasses of $\mathbf{A_T}$ following along subclasses of **A**, as appropriate. Using this architecture one is encouraged to implement methods that return artifact instances configured specifically for the class under consideration.

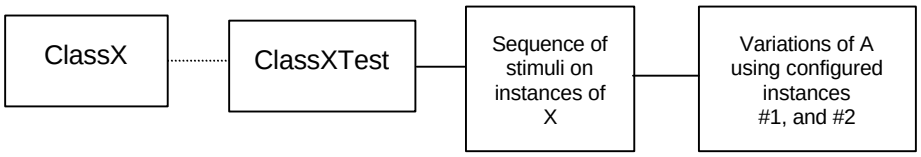


In the above diagram, **ClassA** and **ClassB** are classes under test. **ClassATest** and **ClassBTest** are test suites arranged in a hierarchy that parallels the classes they test. **ClassATest** implements methods (not indicated) that each return a configured artifact instance. The test artifacts are:

Test Artifact name	Description
"Configured instance of A #1"	A ***configured instance*** artifact - configured to, when executed, return an instance of ClassA in a particular state.
"Configured instance of A #2"	A ***configured instance*** artifact configured to, when executed, return an instance of ClassA in a particular state that is assumed to differ in some useful and interesting way from the above instance
"Variations of A"	A ***variation*** artifact - provides object variations and is configured to refer to and enumerate the two configured instances above.
"Sequence of stimuli on instances of A"	A ***test sequence*** artifact - represents a sequence of stimuli on some instance of ClassA, that is, a unit test case. It is configured to take as input the "Variations of A"

	artifact, which means that the sequence of stimuli are performed on each of the two instances of ClassA defined above.

The above is a very simple, and in some ways contrived test, but it does show potential relationships between classes and artifacts. A more likely usage would be to employ variations of **ClassA** as input to constructors and other methods on some other **ClassX**. For example,



(From ClassATest above)

In the above example, a “Sequence of stimuli on instances of X” (a test case for **ClassX**), is passed, variations of **ClassA**. Though not shown here, the test case for **ClassX** uses instances of **ClassA** as parameters for constructors and other message sends on **ClassX**.

As a side note, for cluster or subsystem tests, it is less reasonable to organize according to the structure of the objects under test, and more likely that a suite class will appear in a flatter test hierarchy that takes a more functional perspective of the system under test.

Regardless, as a matter of policy, a well designed suite should always be targeted toward testing some component under test, [Dwyer 99] whether it is a single class, cluster, or subsystem.

Effectiveness

The approach described here is most effective when applied to tests for more than just a handful of classes. Our experience has shown that the value of building tests upon a framework of reusable test components such as the artifacts described here, increases as the scope of the tests and the number of reusable artifact definitions grows.

It is also important to note that tests and test components are development work products in themselves and should be designed along with the rest of the system under test. A common question developers should ask themselves as they design code, and that should certainly be asked at reviews, is “how do I prove that this works?” It is also important to publicize and share reusable test components, especially *configured instance* and *object variation* artifacts, as they become available.

Sample Code

```
/* Return an artifact that returns a configured instance of a
 * Calculator class after adding an int.
 */
public ConfiguredInstance calculatorAfterAdd() {

ConfiguredInstance artifact =
new ConfiguredInstance(“Calculator after add");
```

```
// Set subject of step via constructor specification
artifact.setConstructorStep(
new Stimulus(“Default").setMessageSpec(message(Calculator.class)));

// Apply stimulus to subject via MessageStep
artifact.addComponent(
new Stimulus(“Add number").setMessageSpec(message(subject(),
                                                  "add",
                                                  new int[] {123}));
return artifact;  }
```

Example 1

```
/* Return an artifact that references an artifact. Reference
 * an artifact that provides a Calculator in a state of having been
 * had a value added as a ‘constructor step’. Then subtract a
 * value from it.
 */
public ConfiguredInstance calculatorAfterSubtract() {

ConfiguredInstance artifact =
    new ConfiguredInstance(“Calculator after subtract");

// Reference existing artifact
ArtifactReference reference =
      new ArtifactReference(“Reference”,MyClass,”calculatorAfterAdd”);

artifact.setConstructorStep(reference);

// Apply stimulus to subject via MessageStep
new Stimulus(“Subtract number").setMessageSpec(message(
                                                  subject(),
                                                  "subtract",
                                                  new int[] {123}));

return artifact; }
```

Example 2

```
/* Instantiate and execute an artifact to retrieve its value.
new MyClass().calculatorAfterAdd().getValue()
```

Example 3

The first example shows a method that returns an artifact that describes an instance of Calculator configured with the default constructor, with an ‘add’ stimulus applied. The result of evaluating the constructor specification is subsequently accessible by `subject()`, which is assumed to be implemented in the method class’ hierarchy .

The second example shows a method that returns an artifact that is composed of a reference to the first artifact, as a subject, with a subtract stimulus applied to it.

The third example simply shows instantiating an artifact by invoking a method that returns it, and then sending grtValue() to execute the artifact and return its value.

Conclusion

Widespread adoption of component architectures in the software industry is helping increase developer productivity and software quality. Applying the same architectural principles and design for reuse to component tests as the components under test is a sensible approach.

This article has described a component architecture for automated component testing frameworks, based on a family of objects called test artifacts that move common testing behavior, frequently implemented within automated component tests, from the tests themselves to the test artifact objects. The benefits of this approach are

- Less time is spent implementing component tests because the framework implements common testing behavior
- The test structure suggested by the artifacts themselves guides test developers down well defined implementation paths, making tests consistent and providing test developers with design momentum.
- Because artifacts can be used to completely describe tests, they may serve as a basis for test tool implementation, and targets for automated test generation.

References and further reading

[Beck 94] Kent Beck, *Simple Smalltalk Testing.* The Smalltalk Report 4(2):16-18, October 1994

[Binder 99] Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Object Technology Series. 1999

[Dwyer 99] Graham Dwyer and Graham Freeburn, *Business Object Scenarios: a fifth-generation approach to automated testing*. In *Software Test Automation : Effective Use of Test Execution Tools. Edited by* Mark Fewster and Dorothy Graham Addison-Wesley Pub Co. 1999

[Firesmith 96] Donald G. Firesmith, *Pattern Language for Testing Object-Oriented Software,* Object Magazine, Vol. 5, No. 8, SIGS Publications Inc., New York, New York, January 1996, pages 32-38.

[Gamma+ 95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing. 1995

[Fowler 99] Martin Fowler, *A UML Testing Framework. Software Development Magazine*. April, 1999

[Fewster 99] Mark Fewster, Dorothy Graham *Software Test Automation : Effective Use of Test Execution Tools.* Addison-Wesley Pub Co. 1999

[Hetzel 84] Bill Hetzel, *Software Testing*, QED Information Sciences, Inc. 1984

[Jeffries 99] Ronald E. Jeffries, *Extreme Testing*, Software Testing and Quality Engineering 1(2):23-26 March/April 1999

[Jorgenson 94] Paul C. Jorgenson and Carl Erickson,. *Object-Oriented Integration Testing*. CACM, 37(9):30-38 September 1994

[Kaner 93] Cen Kaner, Jack Falk, Hung Quoc, *Testing Computer Software*, International Thompson Computer Press 1993

[Marick 95] Brian Marick, *The Craft of Software Testing*, Prentice Hall. 1995

[McGregor 94] John D. McGregor, Testing Object-Oriented Systems Tutorial Notes In *The Conference on Object-Oriented Systems, Languages and Applications, October* 1994

[McGregor 96a] John D. McGregor and Timothy Korson, *Integrating Object-Oriented Testing and Development Processes*. CACM, 37(9):59-77 September 1994

[McGregor 96b] John D. McGregor and A. Kare, *PACT: An Architecture for Object-Oriented Component Testing*, in the Proceedings of The Ninth International Software Quality Week in San Francisco, California, SR Institute, San Francisco, 22 May 1996.

[McGregor 99] John D. McGregor *Test Patterns: Please Stand By,* Journal of Object-Oriented Programming, 12(3):14-19, June 1999

[Murphy 94] Gail C. Murphy, Paul Townsend, and Pok Sze Wong, *Experiences With Cluster and Class Testing*. CACM, 37(9):39-47 September 1994

[Myers 79] Glenford Myers, *The Art of Software Testing*, John Wiley & Sons, Inc. 1979

[Poston 94] Robert M.Poston, *Automated testing from Object Models*. CACM, 37(9):48-58 September 1994.

[Siegel 96] Shel Siegel, *Object Oriented Software Testing, a Hierarchical Approach*. John Wiley & Sons, Inc. 1996

[Siepmann 94] Ernst Siepmann, A. Richard Newton: TOBAC: A Test Case Browser for Testing Object-Oriented Software. ISSTA 1994: 154-168

QW2K Paper 2A1

Mr. James Andrews
(The Open Group)

Automated Conformance Testing for IT & T Product Certification

BACK TO QW2000 PROGRAM

Key Points

- Methods for automatec conformance testing
- Certification of information and telecommunications technology against industry standards
- Product certification of operating systems, wireless telecommunication, and internet technology in practice

Presentation Abstract

This talk will outline the principles of conformance testing and the methods by which automated test suites may be develooped to assess the conforming of implementation of open standards.

The talk will further outline the various approaches to certificatio nof IT&T technology and how automated testing can be used as teh basis of product certification.

This will be illustrated with examples of real world experience in the product certification of the UNIX opearting system, the Wireless Application Protocol, and the network computer known as the Open Brand.

About the Author

Mr. Andrews is the Conformance Quality Manager of the Open Group. He was responsible for the definition of the Open Group's Test Suite development procedures and thereafter for the Operation of the Open Group's Open Brand program.

Mr. Andrews has 20 years experience in softawre quality assurance and 15 years experience iun IT&T software conformation testing and certification. He is a lead assessor for Natlas for the assessment of IT&T test laboratories.

Mr. Andrews lectured extensively on conformance testing and certification and has been translated into several European languages, Russian and Japanese.

BACK TO QW2000 PROGRAM

Automated Conformance Testing for IT & T Certification

James Andrews

Conformance Quality Manger

The Open Group

THE Open GROUP

Development Testing

- Development or QA testing.
  - Test to look for bugs.
  - Product is its own standard.

THE Open GROUP

Conformance Testing

- Testing for conformity.
  - Test cases traceable back to the open standard.
    - Though they may become the standard themselves for proprietary specifications.
  - Test restricted only to the standard.
  - No assumptions on how standard implemented.
  - Avoid subjective judgments.
    - Automated verdict assignment.
  - Development testing.
  - Final certification.

THE Open GROUP

Conformance Testing Versus Quality Assurance Testing

- Conformance testing
  - Black box
  - Functional
  - Top down
  - Portability
  - Interoperability
- QA testing
  - Code bugs
  - Performance
  - Robustness
  - Quality of implementation

THE Open GROUP

Conformance Test Suite Design a Few Criteria

- Design of test suites to be objective
  - Repeatable
  - Reproducible
  - And impartial
- Design of test cases
  - The use of assertions
  - Independence, set up and clean up
  - Parameterized
  - Portable
  - Verdict assignment: Pass, Fail, Unresolved, Untested

THE Open GROUP

TETware Test Harness

- Support for POSIX **assertion-based testing**
- Builds, executes and cleans up test suites
- Test scenarios can be defined using a powerful scenario language

THE Open GROUP

TETware Test Harness

- Test parameters can be specified using a flexible configuration variable mechanism
- Configuration information and test results are recorded in a journal
- Support for the standard POSIX result codes is built in - user-defined results are also supported

THE Open GROUP

Technology Fields Where Conformance Testing Has Been Applied?

- TETware based tests exist for
  - Operating system tests
  - Networking API tests
  - GUI tests
  - Network protocol tests
  - Object oriented testing
  - Data management tests
- Alternative test frameworks have been used for
  - Web browsers
  - Network computers
  - Wireless web devices

THE Open GROUP

Test Suite Lifecycle

- Assertion design and review
- Beta test
- General release
- Maintenance release
  - Clear bugs
  - New tests
  - Beta review
  - (6 months overlap)

THE Open GROUP

Test Assertion Design

- Maps the test to the standard.
- ‘Shall, must.’
- ‘May, should.’
- As a tool to debug the specification.
- As a tool to debug the tests.
- As the basis of objective assessment of results.

THE Open GROUP

Certification Value Add

- Objective independent review
- Simple criteria for procurement
- Single sheet
- Continuing conformity
- Register of products
- Lab accreditation

THE Open GROUP

Interpretations Process

- Agreed errors in test suite
- Errors in specification
  - Permanent
  - Temporary
- Independent review
- Public list
- Industry review
- Appeal

THE Open GROUP

Conformance Statements

- Statement of options Implemented
- Means of implementation
  - Eg limits
- Scope of certification
  - Derived conformity
- Test suite parameterization

THE Open GROUP

Trademark License Agreement

- Standards of quality
- Warranty of conformity
- Continued conformity
- Use of certificate and mark
- Appeals
- Removal of certificate mark
- Right of audit

THE Open GROUP

Assessment

- Conformance Statements
  - Self consistent, correct
  - Test suite parameterization
- Referenced interpretations
- Test report self consistency
- Scope and identity
- Legal agreement

THE Open GROUP

Certification in Action

- The Single UNIX® specification
- Graphical user interface
  - Desktop style
- 35,000 conformance test cases
  - Six test suites
- Evolution
  - Certification ready

(R) UNIX is a registered trademarks of The Open Group in the US and other countries.

THE Open GROUP

UNIX Certification

- UNIX® 98 System
- UNIX® 95 System.
- UNIX® 98 workstation.
- UNIX® 98 server.

The operating system of choice for internet servers.

THE Open GROUP

Wireless Application Protocol

- Interoperability
  - Protocol
  - End-to-end
  - Multi-party
- Rendition
- Functional correctness
- Relationship of The Open Group to the WAP Forum

THE Open GROUP

Futures

- FIX –A protocol for exchanging financial information
- Directory services- LDAP
- Security – CDMA
- Realtime and Embedded Systems Testing
- Other open specifications
  - Using the UNIX® System experience elsewhere

THE Open GROUP

Conclusion

- Market lead
- Consumer confidence
- Openness
  - Impartiality
  - Automation
- Framework
- Limitations
  - Quality of service
  - Value-add

THE Open GROUP

QW2K Paper 2W1

Mr. Ted Fuller
(Agency.com)

Notes From The Front Lines: How to Test Anything and Everything on a Web Site

BACK TO QW2000 PROGRAM

Key Points

- Identifies Internet-specific Quality issues you need to address.
- Discusses details you don't want to miss before launching a web site.
- Explains how to apply existing Quality practices to prevent problems that might otherwise show up in testing.

Presentation Abstract

Some of the best training in web site Quality comes only with experience. This presentation will share the lessons you won't find in any book, based on 35 years experience (in Internet Time) testing the full spectrum Internet, intranet and extranet websites.

The focus of this paper will be on practical lessons and tips and tricks, but we'll also discuss how to apply traditional Quality practices on Internet projects.

After attending this presentation, you will approach Internet projects with confidence, knowing that you will be using familiar tools and skills, and simply applying them to a new medium.

About the Author

Ted Fuller is a veteran of Quality Assurance in interactive media with AGENCY.COM. He has five years experience testing web sites, retail and tradeshow demos and kiosks, and computer based training.

Past experience has served him well in QA. He honed his communication skills as a technical writer, writing documentation and help files for products ranging from software, personal computers, and hydroelectric generation system modeling systems, to web site content management systems, client-server applications and database configuration tools. Internet deadlines don't faze him, after five years fighting forest fires in California and Oregon.

BACK TO QW2000 PROGRAM

Notes From the Front Lines: How to Test Anything and Everything on a Web Site

Ted Fuller
Quality Control Specialist
AGENCY.COM

©2000 Ted Fuller AGENCY.COM

Today's Program

- Lesson # 1:
  Testing web sites is completely different from testing any other software
- Lesson #2:
  Testing web sites is no different from testing other software, especially software with a GUI
- Lesson #3:
  Nothing in your QA experience prepares you for working on a web site project
- Lesson #4:
  Everything in your QA experience prepares you for working on a web site project

Lesson #1:
Testing web sites is completely different from testing any other software

- Everything's New
- Are you Testing the Latest Version?
  - Quality Process in Place
  - Browsers
- Changes to Software and Infrastructure
  - Structural Changes to Code
  - Hardware Changes

Lesson #1:
Testing web sites is completely different from testing any other software (continued)

- Site Index Page and Search Functionality
  - Appealing Indices
  - Relevant Searches
- Browser Idiosyncrasies
  - Confirm Functionality Support
  - Browser Archives
  - AOL's Browser
  - Configuration and Platform Testing
- Someone Still Has to Review Every Page

AGENCY.COM

Lesson #2:
Testing web sites is no different from testing any other software (especially GUI software)

- Tips & Tricks
  - Common Elements
  - Test Procedures
- Automated Tools
  - Link Checkers
  - Code Validators
  - Transactions and Other Functionality

AGENCY.COM

Lesson #3:
Nothing in your QA experience prepares you for working on a web site project

- Two Challenges
  - Time
  - Content

AGENCY.COM

Lesson #4:
Everything in your QA experience prepares you for working on a web site project

- You Know More Than You Think
  - Backend Functionality
  - Existing Quality Practices
  - Integration
  - Frontend Interface

Notes From the Front Lines:
How to Test Anything and Everything on a Web Site

By: **Ted Fuller**
Quality Control Specialist
AGENCY.COM
tfuller@agency.com
©2000 Ted Fuller AGENCY.COM

Some of the best training in web site Quality comes only with experience. This presentation will share the lessons you won't find in any book, based on 35 years experience (in Internet Time) testing everything from computer based training, retail and tradeshow demos and kiosks, to all colors, shapes and sizes of Internet, intranet and extranet web sites.

Lesson #1:
Testing web sites is completely different from testing any other software.

Everything's New

The World Wide Web is built and viewed using all new technology. Browsers, plug-ins, java, flash, quicktime, mp3. Developers are doing things with code no one imagined two, five, ten years ago. How can we test it when they're out on the edge developing and using new tools and technology at a breakneck pace that don't have any benchmarks or standards?

You test it by defining the intended functionality, features and performance. Anything may be possible, but it isn't necessarily what the client wants. Make sure developers and clients understand and agree on what they need, what they want, what's feasible, and what the ultimate goal is. Help them establish those definitions by having them explain how to measure the features and functions, and identify the characteristics that indicate satisfactory performance. Does this sound familiar? It should: the team needs to define requirements and specifications, and agree on how to determine that those requirements and specifications have been fulfilled. These are fundamental Quality practices.

That being said, there are some testing issues unique to web projects.

Are You Testing the Latest Version?

Quality Process in Place: Development teams have multiple programmers, typically young workers who may be unfamiliar with good development and Quality practices. Development teams must use version control software. Changes are too easy to make. Testing needs to be done on a dedicated environment. We're all used to software evolving over the life of the project, but on the web we're talking hourly evolutions. If the test environment isn't managed effectively, the resulting chaos will bog down the testing process, and impact the project schedule and quality.

Browsers: Along with making sure your processes and procedures maintain version control and protect your test environment, you need to make sure your browsers are showing you the latest version as well. You need to keep your cache and cookies clean. It's not enough to select the option to "compare cache version to network version Every Time." You need to empty the cache, clear the history and regularly delete the coookies.txt file. Otherwise you risk inaccurate (not to mention embarrassing) regression test results.

<u>To clear cache, go to a web site outside the scope of your test project, then:</u>
In **Netscape Navigator:** Edit/Preferences/Navigator branch: Clear History; Advanced branch: Cache/Clear Memory Cache; Clear Disk Cache.

In **Internet Explorer:** View/Options (v4) Tools/Options (v5) General tab: Temporary Internet Files: Delete Files; History: Clear History.

To delete the cookies file:
Search the computer's hard disk drive for the file cookies.txt. Delete it.

Changes to Software and Infrastructure

Structural Changes to Code: Whenever a major change is made to the directory, you need to do regression testing on all the links. A new directory, or different directory or filenames, changes the path, which breaks all the links.

Hardware Changes: Whenever the project changes location, whether moving to a new port on the same box, or moving from one physical box to another, you need to do thorough regression testing. Recheck everything to verify that all the links, features and functions that worked on the original server still work on the new server. Broken links are the obvious issue, but there's also the risk to functionality because of different machine configurations. Unless the boxes were obtained and configured simultaneously, and religiously maintained and upgraded in parallel, they will not be identical, and your software is at risk.

Site Index Page and Search Functionality

Two of the common features found on nearly every web site are the ones that sometimes get the least attention until the launch deadline approaches. Typically the Site Index page and Search functionality are the last things coded. While they're common components of most projects, they're rarely included in design and copywriting efforts. They're reminiscent of how Online Help is approached. A standard necessity, but nothing that gets the attention it needs until late in the process.

Appealing Indices: Good sites have Site Index pages that use the same graphics and visual navigation cues users see on the Home Page and main sub-section pages. This helps them associate the links on the index with the content they've accessed from there.

Relevant Searches: Likewise, the Search page needs the site's look and feel, and includes instructions and examples. More importantly, the results page is easy to interpret, with enough information to give the users the context of the result. These are just basic requirements for Search. It's a specialty in the industry. We're helping one of our major clients fine tune their search function so that it brings back relevant, accurate, informative results.

Browser Idiosyncrasies

Browsers are a new technology unique to the web. Be grateful we're past the early versions with their inconsistencies and instability. The version 4 browsers from Netscape and Microsoft were the first where the features and functions matched up pretty well. The version 5's will be even more similar. Even so, you need to understand all browsers are not created equal (especially on different platforms). The same web site files can look very different on different browsers. Text, tables, fields and buttons don't display consistently. Spacing and position of elements is inconsistent. You need to see it to believe it (and report it).

Confirm functionality support: One of the challenges of the web is that the technology keeps changing, and the development tools and the browsers are rarely in step. Webmonkey is a great resource site for developers. The Browser Chart (http://hotwired.lycos.com/webmonkey/browserkit/), provides a quick reference to what technologies each of the major browsers supports.

Browser Archives: Although the major browsers have converging feature sets, you can't ignore the old versions. You don't want to aggravate users who would rather keep their stable, functional systems rather than upgrade with every new release. Even more importantly, the current non-US versions are one- to two versions behind, which may include a significant portion of your audience. How do you find old versions of browsers: Go to a browser archive: http://www.browser.com

AOL's Browser: America Online's proprietary browser is actually built on the Internet Explorer platform (although probably not the next version, since AOL owns Netscape now). But it has some idiosyncrasies of it's own. How important is it for your site to support AOL? That depends on how much you want to impress the 22 million members who use it. To find out more about AOL's proprietary browser, go to: http://webmaster.info.aol.com/ AOL's version of the Browser Chart is accessible by clicking the Browser Info link on the left, and the Computability text link on the page that displays.

Configuration and Platform Testing: While the field of major browsers is relatively narrow, there is a broad spectrum of computer platforms used to surf the web. One of the key elements of a users web experience is performance, and much of that depends on their personal system: Chip speed, RAM, Video memory, modem speed, bandwidth available. You can't test every possible combination, but it is possible to evaluate performance on a variety of systems and configurations. But you don't want to do it yourself. The setup and maintenance is a full-time job, as evidenced by the number of test lab vendors in the Exhibit Hall. You want a couple of machines, not only for browsers but for different hardware, so you check that chip and modem speed and RAM don't limit the user experience. Once you confirm the site is stable, pass it to a lab for configuration testing, as well as performance testing. Like fine tuning Search functionality, performance testing is a specialty. You need to be aware if the site is slow, or bogs down when the internal network is busy, it will be much, much worse out on the real web world. But for realistic load testing, go to a lab equipped and trained to do it right and offer tuning recommendations.

Someone Still Has to Review Every Page

Because content tone, images, and look and feel, are such important parts of the user experience, it's critical that every page gets reviewed for these criteria. Consistency is a subjective measure, but critical to prevent confusion, which is fatal to audience satisfaction. You need to check the display of graphics: are they complete, clear, and the correct images?

If you have trouble accepting this notion, think of it as if it were a print project. It's been proofread and copy edited by countless people throughout production and at every milestone of development. The final milestone is the galley or proof. You see a sample of the first impressions, checking every page with a critical eye to make sure everything's in its proper place. And even when you give the go ahead, you still check it as comes off the press, and during the press run, to make sure it's right.

Lesson #2:
Testing Web sites is no different from testing other software, especially software with a GUI.

Tips & Tricks

Here are some issues to remember and test, and surefire ways to break a website.

Drop down lists: Try selecting a heading or blank line; Is the text too long for box width? Any items missing from list?

Forms: Do the text fields hold enough text (try: Elizabeth Montgomery in name; San Francisco in city; Elizabeth_Montgomery@anynetwork.com), What happens when the user enters bad data or formats? Some symbols (#$%) and Punctuation marks (":,?/) are reserved characters that require special handling in fields (especially an apostrophe in SQL).

Authentication: If users can log on, they need to be able to log off. What if they forget their password? You also need to include password maintenance functionality, or instructions on how to make changes.

Site Map: Does it match main navigation elements? Are all the various menu elements listed? Any missing elements? Do all the links work?

Email links: Live address? Right address? Correct distribution when received?

External links: Poorly defined? (link to a periodical should go directly to the intended article; don't force users to search for it; Out of date? External sites change. Out of date links annoy users.

Printing forms: Check on all browsers (QW2000 Speaker Data Sheet). Better to use PDF.

Common Elements: The supporting documents for this presentation include an appendix: Web Site Elements/Features Test Procedures. It's a list of software features, functions, and elements, and how to test them. It includes:

compatible hardware, software and platform issues;
runtime, utility and support software;
conflicts with other software;
user registration, log-in and record keeping;
menu issues;
navigation issues;
special effects, animations and videos

and more.

Look at that list closely. Did you notice that nothing on the list is unique to web sites? Testing any or all of these issues is a part of testing any software. I've included it in the presentation for people new to software testing, and new to GUI's in particular.

For those of you who have tested GUI software, it's important to recognize you're testing the same functionality on the web; it's just in a different software application.

Automated Tools

For basic brochure ware sites, link checkers and code validators should be used. For more complex sites that handle transactions and other functions, you should consider automated test tools. In both cases, it's tools are important, but they need to be used correctly.

Link Checkers: You should use a Link Checker to automate the process of clicking every link. You have to understand though, that the link checker isn't perfect. Just as a spell checker won't catch when you type the wrong ward, a link checker can't tell you that the link is going to the wrong place. If it only tests items coded as links -- what about items that are supposed to be coded as links, but aren't? What about a link that may be functional (it points to a working URL), but is it the right target? (http://www.whitehouse.com doesn't go where you think). Relative links may not stay stable throughout server promotion and deployment; what if linkages are inconsistent? The lesson here is that you need to analyze the results.

Code Validators: Your developers should use a code validator as part of their unit testing. A code validator compares the code to the established standard (such as they are) and flags violations. While developers should follow the standards to ensure performance on the browsers that are supposedly built to match the code standards, often interface and functional requirements require workarounds. These workarounds are fairly common, but you and the developers do need to be aware of them to make sure they don't impact performance.

Transactions and Other Functionality: The key to automated tools that develop test scripts are the time required to set up the scripts, and maintenance to keep them current with the evolving web site. They require a huge amount of customization, parallel to the coding effort. You have to balance this time investment with the value of the resulting automated tests. If the site structure will remain stable, automated tests are critical for thorough regression testing. Again, this is a specialty in the industry; and there are vendors in the Exhibit Hall.

Lesson #3:
Nothing in your QA experience prepares you for working on a web site project

Two Challenges

You face two major challenges in web projects: Time, and inexperience with the major components of web sites: Marketing, graphics and the world of the web.

Time: Internet time is unreal. There are two or three presentations on the topic this week at QW2000. Most companies are playing catch-up: if they don't have a site up yet, they need it now. If they do have a site, it's the one they put up in a big hurry to say they had one, and it desperately needs to be updated. Either way, most companies are in a huge hurry to get a site online. It's not as bad as the daily deadline at a newspaper, where your talking hours and minutes. But you have to understand, we're usually talking days, weeks if we're lucky. Most of our projects are measured in weeks, but in order to deliver on time, we require that the client review our deliverables and return comments in 24 hours. If they can't, we can't launch on time.

The impact on testing is something we're all familiar with: Schedule slippage and compression, resulting in time budgeted for testing getting eaten up by development. When that happens, risk assessment and mitigation is crucial. Basic site navigation and any mission critical functions get highest priority. If there are too many functions that are not ready for primetime, it is relatively simple to keep them inactive at launch, by removing the links that take the users there. When they are finished, tested and ready, they can be phased in and promoted as a new feature. This is an important option, compared to off the shelf software, where purchasers expect a complete, fully functional package that doesn't ship needing an immediate update.

If launching the site in phases isn't an option then you need more testers. We talked about outsourcing previously regarding platform and configuration testing. They are also an excellent resource when you have more test tasks than time. It's the easiest way to get help from qualified people. You don't need to teach them how to test, or how to file a defect report. You just brief them on the site's objectives and the specific tasks they're executing, and you're set. Two important issues: You need to make sure your site is ready for hired testers: it's a waste of time and money for them to report broken links that keep them from getting to the areas they need to test. And you need to be very explicit in what you want tested and how.

Content: The other major challenge in testing web sites is the critical importance of marketing, graphics and the "satisfying user experience." If you aren't comfortable assessing these subjective aspects, that's understandable. But if your role in QA is to help ensure a web site's success, you need to take steps that the site meets high quality standards in these areas. Enlist experts in each field, and have them review the site as part of your quality process. Why is this necessary? Because the quality of a web site goes far beyond functionality.

It doesn't matter if we're talking about a public site to promote your company, or a site your customers come to for information and service, or a site investors and analysts visit for business information, or a single site that serves all three audiences: Of course it needs to function correctly. But even assuming it has bulletproof functionality, if it's confusing to navigate, or hard to use, or information is hard to find, or it doesn't meet the audiences needs, your audience will go somewhere else, perhaps a competitor, to find what they're looking for.

If you think of your web site as a product, the marketplace where it's available is incredibly large, and filled with others competing for your audience's attention. You've only got a matter of seconds and a few clicks to establish the quality, functionality and usefulness of your site, before your audience will jump to another one. There's no doubt functional quality is as important on the web as in other software. But simply addressing functionality risks doesn't ensure your site will be successful. Success on the web is based on complete, accurate functionality in providing easy access to useful information in a timely manner.

Lesson #4:
Everything in your QA experience prepares you for testing a web site project

You Know More Than You Think

While the Internet and World Wide Web have revolutionized the way people communicate, the way companies conduct business, and the way products are marketed, sold and delivered, it's important to understand that revolutionary Internet projects require only evolutionary adaptations to their existing Quality practices.

Backend Functionality: No adaptations necessary here, except to understand relationship within structure of a web site. Backend modules still need to be bulletproof, facilitating isolation of defects from separate web site components.

Existing Quality Practices: Specifications and Documentation, Test Plans, Automated Test Scripts, Defect Reporting and Tracking, Change management, etc. These don't change. They are just being applied to a new medium, and involve some different staff (content owners vs. typical SMEs) who may not have experience with established development and Quality practices.

Integration: Connecting the web Frontend to the mainframe Backend functions. Proper documentation, developed well, early in the process and followed meticulously, is critical. This includes component contracts, data dictionaries, and specification of development and staging platforms and a promotion plan.

Frontend Interface: This is where Quality professionals new to the web may balk, saying that this is up to the content owners (Subject Matter Experts), or that it's too subjective to be measured and verified. But because a good web site is the result of successful coordination of content and technology, the Quality efforts can't ignore the content. A closer look at the Interface components helps illustrate why the Quality team needs to include them in their scope.

Documentation: The project documentation should include interface specifications and requirements, so that the client and development team have a good understanding of what's expected. A sitemap outlines the content and its structural relationship. The screen deck shows which navigation elements are on each page of the web site.

Information Design: At AGENCY.COM our Information Architecture includes interface design, logical organization of the information to be delivered, and intuitive navigation design for ease of use.

Content: Analyze the content, make sure it is user-focused, that it is what the user wants and needs. Do the titles make sense, is the information in a logical location? Leave style & tone to marketing department. Focus on factual errors. Watch for poor grammar and typographical errors; they reflect poorly on the web site and developers.

A clearly defined QA process throughout the project, and comprehensive documentation, strategies that work so well in other projects, are your best tools for delivering Quality on a web project.

Appendix: Web Site Elements/Features Test Procedures

This section documents the real-world experience I have acquired in five years of Quality Control, testing multimedia training projects, demo CDs, and websites. It addresses how to manually test project features and functions item by item without aid of automated testing tools.

This document is intended as a learning guide for novice or ad hoc testers, as well as reference for the development team regarding what's involved in testing various features and functions. It was written the majority of testing was done late in the development cycle.

Function	Multi-platform Functionality
Description	If the project needs to work on more than one platform or browser, then it needs to be tested on each specified platform or browser. The scope of multi-platform functionality needs to be clearly stated in the specifications, since it may involve configuration testing that needs to be outsourced.
Lessons learned	Check performance on different platforms early and often. Various versions of Windows may have incompatible system files (.dll etc.). Netscape Navigator and Internet Explorer perform differently. Likewise, different versions of the same browser perform differently, and have different capabilities. Thus, different versions of browsers should be treated as individual platforms.
Test Strategy	Run software on the least common denominator platform, being sure to test all features and functions.

Function	Runtime, Driver, Plug-in Functionality
Description	Need to know how files are being delivered and executed on target systems. Files can be delivered as packaged requiring a runtime, or as a standalone executable. If engines/runtimes and drivers need to be included, the delivery and installation must also be tested. If Plug-ins are required, need URLs for download sites. Instructions are needed for all.
Lessons Learned	
Test Strategy	Test each section of software that relies on Runtime, Driver or Plug-ins.

Function	Splash Sequence
Description	The Splash sequence sets the tone for the entire project. While it has little impact on the functionality of the software, it can make a critical first impression.
Lessons Learned	Test under field conditions to identify any performance issues.
Test Strategy	Have the entire team review the Splash Sequence together to confirm it's operating as intended and consensus is reached on changes.

Function	User Records Functionality
Description	Depending on project specifications, user records can be used to track progress, test scores, identify sections visited, measure elapsed time, bookmarking (when users logon, offers them a shortcut to the last section they visited). Records may be kept on the user's system, or transferred to a central server. Records are generally used in real time to mark segments as viewed, as well as a lasting usage record. Records for repeat users are updated.
Lessons Learned	On multi-platform projects, make sure each component of User Records is compatible with the each platform.

	Test that end of year-end/millennium issues won't impact performance. Registration, Logon, Bookmarking sections should be approached as components of the deliverable that need to be scripted and graphically designed as part of the whole project.
Test Strategy	Use different identities to create different records. Take detailed notes on navigation/path as project is tested. Compare to user records. Depending on project specifications, determine how many test records need to be created (check project specifications for anticipated load) and identify what aspect of records needs to be tested. (see related test topics below) Develop a plan for what records to create. Test that existing records are updated correctly by logging on, visiting a new section and exiting. Confirm that record accurately documents this progress. For complex systems, it may be useful to create & save a set of preliminary records in order to restore system after performing tests that will update records. If the record is transmitted to a central server, check that the user instructions work under field conditions. Set up an internal simulation (transmission to internal server), test transmission and confirm data integrity. When successful, conduct live test transmission to client's system under field conditions.

Function	User Registration
Description	Refer to project specifications for specific details. In general: Each time the site is visited, users are prompted to logon, with New Users given the opportunity to register. The registration segment should be able to prevent duplicate records from being created. New Users complete a form to establish user ID info and password. When the form is submitted a record is created. As users move through the software, a temporary record notes user activity. When the user exits, the initial record is updated, and referred to the next time the user logs on. Depending on the project specifications, users may be able to modify registration info for an existing record. This is a complex function that would need extensive testing.
Lessons Learned	Test on each platform.
Test Strategy	Check that the Registration segment has instructions and that the form is easy to complete. Navigation should be intuitive, with explanatory prompts. Take notes on data entered and compare to record saved. Confirm with programmer and compare to project specifications that the record is formatted correctly.

Function	User Logon
Description	Refer to project specifications for specific details. In general: Each time the software is launched, users are prompted to logon using the ID and password they entered at Registration. Generally, users identify from a list or enter their ID, then enter their password. The Logon system checks that the ID and password match existing registration info. Refer to the project specifications for details on how the system should respond to invalid logons.
Lessons Learned	
Test Strategy	Check that the Logon segment has instructions and that the form is easy to complete. Navigation should be intuitive, with explanatory prompts. After logging on, confirm that the software accessed the correct user record by comparing to navigation/path notes.

Function	Bookmarking Functionality
Description	Based on User Records functionality, Bookmarking records the last section visited when users exit, so the next time users logon, they are offered a shortcut directly to that section.
Lessons Learned	Project specifications need to identify how far bookmarking will drill down. Default is Topic Menu level. Confirm that Bookmarking functionality is explained to users.
Test Strategy	Based on the project specifications, exit the software from each point that should have Bookmark Functionality. Logon as the same user. At the bookmark prompt confirm that the system displays the correct section.

Function	Menu Marking
Description	Menu marking helps users understand where they are, where they've been, and where they still need to go. The visual cues make it easier to understand the structure of the program and an idea of how much information is available.
Lessons Learned	Menu marking should be approached as a component of the deliverable that needs to be scripted and graphically designed as part of the whole, with consideration for complexity of code required. Confirm that Menu Marking is explained to users.
Test Strategy	The items on each menu need to be tested for each state. This includes dependencies of higher level menus based on state of sub-menus. For example: check that starting each topic posts an Open icon on menus/submenus. Use those Open icon records as the starting point to check that completing a topic posts a Completion icon on submenus, but leaves an Open icon on the main menu until the final topic is completed. Likewise, check that when each topic is the final topic completed on a submenu, a completion icon is posted on the main menu. In projects that feature menu marking and user records, menu marking is a handy way to confirm that a logon has accessed the correct record.

Function	Exit Sequence
Description	An Exit sequence is the counterpart to a Splash sequence. Rather than getting dumped straight back to the desktop when Exit is clicked, there is usually a graceful exit. Refer to the project specifications for details, but generally: there is a confirmation window, and then an exit screen that may contain disclaimers, copyrights, development credits, fade out images.
Lessons Learned	Exit Sequence should be approached as a component of the deliverable that needs to be scripted and graphically designed as part of the whole project.
Test Strategy	Exit from all sections of the software. If a single sub-routine can't be used throughout, check for inconsistent performance within each section.

Function	Navigation – Menus
Description	Menus are used to navigate through the structure of a project. Thus their functionality is critical to the success of the project. They are important to users so that they get a clear understanding of the scope of the site and how to get to the information they are looking for. Menu Marking functionality is an important cue for indicating to users where they've been.
Lessons Learned	The different state of menu items should be designed. Default "inverse" images are not acceptable. MS Internet Explorer 3 doesn't support JavaScript, which is typically used so that menu items change appearance when the mouse is rolled over the element. In projects with linear user paths, the behavior needs to be defined for the first screen, the final screen, and whether users are able to navigate from start-to-finish and finish-to-start.
Test Strategy	Test menus by using them throughout the testing process. This is a basic functionality that needs to be well-designed, well-programmed and well-tested. Understand the relationship between main menus and submenus, especially in terms of menu marking.

Function	Special Effects (Animations, Videos)
Description	Special effects add pizzazz to projects…if they work. They need to work under 'normal' field conditions, as well as under more stressful situations, especially if the user does something when the special effect is running.
Lessons Learned	Make sure required drivers/runtimes/plugins links are provided & installed. Test on target platform under field conditions as soon as possible.
Test Strategy	Run the segment with the special effect in each round of testing. Do abusive testing (ie clicking during the special effect) to ensure performance under stressful conditions.

Function	Dealer Locator Functionality
Description	Dealer Locator functionality allows users to find any particular product or service provider. It can be a closed search, where the user enters a city, state or zip code, and the system returns the matches for that descriptor, or an open search, where the user navigates through a list of providers, typically organized by geographic region.
Lessons Learned	Know the structure and format of how the dealer info was provided to us, and how it was entered. That will inform you of how much proofreading and structural analysis is required.
Test Strategy	Ideally, each possible search/matching criteria would be tested. But depending on the complexity of the locator list, this may not be feasible. At least test matches in each category

QUALITY WEEK 2000.

Automated Conformance Testing for IT & T Certification

By James Andrews

Conformance Quality Manager – The Open Group

March 2000

© The Open Group 2000

Conformance Testing versus Quality Assurance Testing.
Every organization that cares about the quality of its products undertakes testing. However only organizations concerned with meeting Open, that is Public standards may need to undertake conformance testing.

In many respects a proprietary product is its own standard. Provided that the product meets planned levels of quality and its end user documentation is consistent with the functionality it delivers it may be of lesser importance whether it meets its original design objectives, as defined in a functional specification. The primary thrust of testing for proprietary products is to assure the quality of the implementation in terms of fitness for purpose.

Contrast this with the requirements to be certified against an open standard. Here the primary thrust of the testing activity is to determine the correctness of the implementation under test with respect to the specific requirements of the standard. That is, the implementation conforms to the standard rather than the 'quality' of the implementation per se.

This is a rather different mindset to the type of testing that developers undertake to ensure the quality of the ongoing coding. Conformance testing is by definition black box, in nature. It does not concern itself with how a standard has been implemented only that each and every requirement of the standard has been realized in the implementation. In other words if bugs are present in the code which have no impact of the functional requirements of the standard they are not the concern of conformance testing. In particular open standards seldom concern themselves with performance or reliability. In an extreme case a product may be slow and unreliable but still pass a set of conformance tests since these aspects of a products concern the quality of an implementation rather than its conformance to a standard.

So why is conformance testing important? To understand this one needs to consider the purpose standardization. It is driven by a desire for commoditization, for products to be interchangeable and interoperable. We demand that a telephone is capable of receiving and making calls regardless of who supplies it, and there must be an open standard that enables manufacturers to design devices which achieve this. Though there remains great scope for manufactures to compete on quality and on richness of features beyond the basic standard for connectivity, a telephone that does not conform to this base standard can only communicate with identical telephones. Not only will such a telecommunication device fail to find favor with consumers its very presence will damage the market in telecommunications as a whole by undermining consumer confidence. Thus not only is conformance testing important in telecommunications, it is so important that both users and suppliers demand it as the basis of certification and as a market entry requirement.

In this paper I will concentrate on the science and art of conformance testing with its complementary service of certification, and the role of these as an enabler in the IT & T market

Testing for Conformance

It is the basis of conformance testing that the tests can be traced back to the standard from which they are derived. They must test the standard and only the standard, and they should not make assumptions as to the means by which the standard is implemented. Open standards are by their nature designed to facilitate multiple implementations. Conforming implementations should therefore be interchangeable.

A proprietary standard is a product standard. Conformance tests can be written against such a standard but if they cannot be traced back to the specific wording of the standard then they effectively replace it as a measure of conformance. If this is the case the tests have ceased to be a conformance test suite and have instead become a Quality Assurance test suite. They test the implementation rather than its specification. For proprietary standards it is not unknown for a test suite to be labeled a ‘conformance’ test suite. However once a set of tests has migrated from being directly traceable to the base standard from which they are derived they cease to be a ’conformance test suite’ and become simply a set of tests that must be passed for trademark purposes. In this case the standard itself has lost much of its value as a means to facilitate development of multiple implementations, since what is required to conform has ceased to be publicly defined. The standard, which gave raise to these tests, cannot be regarded as open, nor the tests be considered as a conformance test suite. I stress this not to say that such a situation is somehow wrong but to highlight the rigor that must be applied in the development and maintenance of true conformance tests.

The nature of a conformance test suite is to limit each test to a realization of an explicit requirement of the standard. Requirements should not be implied; therefore the conformance test suite should not presume the availability of functionality beyond the specification under test. Furthermore while a set of tests cannot normally be exhaustive in testing all the functionality (and in particular all combinations of functionality) they must avoid subjective judgments. It is often said that a set of tests cannot prove the correctness of an implementation; they can only identify bugs. Nevertheless this is the purpose of a conformance test suite is a proof of correctness, an implementation either conforms or it does not. This inextricably links automation to conformance testing. Wherever possible subjective evaluations by the test engineer should be avoided. Where the status of a test result cannot be readily determined this should be evident from the outcome of the test. This is most readily achieved by automatic assignment of test results in the form of an appropriate set of verdicts. A test passes, fails or has neither outcome. The verdict assigned in such indeterminate cases

may be termed “Inconclusive” or “Unresolved”. It is often the role of an independent certification authority to use objective processes to assess the meaning of the ‘fail’ or “unresolved” verdicts on the overall conformance of the implementation. Automated conformance test suites facilitate multiple independent developments of conforming implementations by the objective allocation of consistent test verdicts directly traceable to the requirements of the standard under test. This is the philosophy used by The Open Group for certification against open standards.

A conformance test suite can have two uses. It may be used internally within vendors during the development testing as the primary top down, black box test technique, typically moving forward in the development lifecycle up to final functional testing. It may also be used externally as the basis for the award of a certificate and possibly a trademark, for tested conforming implementations of the standard.

Conformance test suite design - a few criteria

It is the nature of a conformance test to act as the arbiter of whether an implementation conforms. Central to this is the concept of objectivity. The outcome of the test should be determined by the standard alone as verified by a test method traceable back to that standard. There are three qualities that make a test method objective:

- Repeatability
  This means that if repeated on the same implementation, on a different occasion, the test will produce the same results.
- Reproducibility
  This means that if repeated in a different location, on the same or functionally equivalent implementation, the test will produce the same results
- Impartiality
  This means that the test suite does not favor one conformant product over another. Which is to say that in order to pass the test the implementation need only conform to the specification, that no functionality outside of the specification is required, and that the implementation of the specification is not otherwise constrained.

In order to meet these qualities each test must itself have certain qualities. In the case of The Open Group test technology it must be:

- Based on assertions
  The use of test assertions is a way of establishing and maintaining a traceabillity between the test method, and the specification from which it is derived. A test assertion is a semi-formal definition of the test purpose. Mandatory test assertions are derived from each statement in specification where the word ‘shall’ is used. Conditional assertions are derived form each statement in the specification where ‘may’ or ‘should’ is used.

- Independence, Set up and clean up
  The outcome of each test shall not depend on the outcome of another test. This implies that each test shall undertake its own set up and clean up.
- Parameterizable
  Where the specification permits optional behavior or behavior is implementation defined this shall be reflected in the selection of test cases and where appropriate in the values for testing.
- Portable
  The test should operate on any conformant implementation of the specification; a test should not presume a specific operating environment other than that mandated by the specification itself. Where functionality is required beyond the specification in order for an individual test to run it should, where possible, be provided by the test suite itself.
- Verdict assignment, Pass, Fail, Unresolved, Untested
  A consistent verdict assignment should be reported. If the verdict cannot be resolved to a Pass or Fail this should be reported as Unresolved. When a test is not selected or cannot otherwise be run (for example there is no known test method for the assertion) the test should report 'untested'

Fortunately many of these aspects of test case and test suite design can be themselves automated

TETware Test Harness

TETware is a set of software utilities developed by The Open Group to act as a plug and play environment for assertion based test cases. It automates much of the work of test management and report writing so that test engineering effort can be concentrated where it is most needed, in the design of test cases. Some of the features of TETware are:-

- Support for POSIX assertion-based testing
  TETware manages the selection of conditional tests, which are concerned with optional functionality. Furthermore TETware includes the test assertion (and method) in the report generated for each test case in order to facilitate debugging of the implementation for the failed tests.
- Builds, executes and cleans up test suites
  TETware manages both pre and post test conditions to ensure that each test is independent of other tests. The result of a given test is not dependant on what has gone before. In addition TETware provides utilities to build, configure, install and run the test suite as a whole.
- Test scenarios can be defined using a powerful scenario language
  TETware enables differing scenarios to be run depending on whether the purpose of the test run is for debugging, regression testing or a formal certification run. It is a feature of TETware that users can add additional tests, select a subset of tests or repeat individual tests for development purposes. By

this means users of TETware can also make use of conformance tests for performance evaluation.

- Parameterization
  Test parameters can be specified using a flexible configuration variable mechanism. TETware handles the use of parameters to tune the testing to the specific implementation, for example to select optional tests and to set implementation dependant limits and values.
- Configuration information and test results are recorded in a journal
  TETware supports a standardized test reporting format that includes the test parameters, the individual test assertions with the associated result verdicts, and time stamps. These are written into in a file known as the journal. TETware also provides utilities to parse this journal into a test summary report in which the overall test outcome of test campaign can be easily viewed, for example total numbers of test in each category of, PASS, FAIL, UNRESOLVED, and UNTESTED. These test results can also be further broken down into test groupings.
- Test result verdict assignment
  Support for the standard POSIX result codes is built in, - user-defined results are also supported. In addition to these POSIX test verdict assignments users can define additional test result codes for development purposes.
  In addition to these POSIX test verdict assignments users...

In what Technology Fields has conformance testing been applied?The techniques of automated conformance testing have been used in a wide variety of Information and Telecommunications technology fields (IT&T). Much of the philosophy of conformance testing was developed in the telecommunications field during the deregulation from 1980 onward and in particular for the Open Systems Interconnection (OSI) initiative. Ironically the confusion engendered by multiple testing and certification programs associated with OSI may have compounded the over complex standards base, engendering a relative failure of this technology in the market place. More recently conformance testing was used much more successfully in the field of operating system components for example the UNIX system. Today conformance testing is now used widely across all IT&T industry sectors.

Conformance testing based on The Open Groups TETware include

- Operating System tests
  In particular POSIX and UNIX systems.
- Networking API tests
  NFS, XTI, and sockets.
- GUI tests
  The X Window System and Motif
- Object oriented testing
  The OMG's Corba

- Data management tests
  LDAP

The Open Group also has a test framework for rendition tests, which include

- Web Browsers
- Network Computers
- Wireless web devices

Other technologies associated with conformance testing with which I am familiar, but which are not based on The Open Group test frameworks include

- Leading Programming languages
  - C & C++
  - Fortran
  - Cobol
- OSI protocols still in use.
  - X25
  - X400

Test suite lifecycle

It may perhaps be useful to look at a lifecycle for the design and implementation of conformance tests.

- Assertion design

  The development of a conformance test suite starts with capturing the specifications in a series of individual test purposes know as assertions. This forms a semi formal definition of the test suite. In an analysis of the specifications those statements that use the term 'Shall' and 'Must' are identified. The language around such statements is then broken down into distinct stand-alone test assertions. For example the sentence "On calling of the set function the value of the variable %name shall be set with %value unless the %value, contains unescaped reserved characters, in which case the variable shall be set with the string 'invalid'.

  This might give rise to two obvious assertions

  1. On calling of the set function the value of the variable %name shall be set with %value
  2. If %value contains an unescaped reserved character the named variable shall be set with the string 'invalid'.

  However there is also a third assertion which the test design may consider implied.

If %value contains an escaped reserved character the named variable shall be set with the %string with the reserved character unchangedThe design of assertions in this way is both a science and an art. Clearly the specification fragment could be implemented

in a single multiple part test but splitting it in this way makes the reason for a pass or fail more obvious and aids debugging of the implementation. It may be that no test method can be identified for certain specific assertions. In this case they would be marked 'Untestable'. In the example these assertions would be mandatory, but language such as 'may' 'either' or 'implementation defined' will also give rise to test assertions, but in this case these assertions will be conditional on the support or otherwise of optional behavior in the specification

- Assertion review

  It should be noted that the third assertion requires diligence in identification since it is implied rather than explicit. In fact a test result of fail in this case may well give rise to a challenge to the test method. A fail would not necessarily represent a non-conformance since the specification fragment is actually silent on the behavior of the function when escaped reserved characters are in %value. The value of this third assertion is in having drawn attention to the fact that the specification is imprecise in this area. If assertions are written and reviewed in parallel with the review of the specification itself, this can be a valuable tool in the review process, and may help to identity ambiguities or errors in the underlying specification. For maximum value assertions should be reviewed early in the test suite development lifecycle and prior to the writing of test code. In this way the quality of both the specification and the final test suite are improved in parallel prior to developers expending effort on coding an implementation and of test engineers in writing test code.

- Beta test

  Once the test assertions have been written and been subject to internal testing there is no substitute for exposing the test suite to real life implementations in a 'beta' test. The implementations used for the beta test need not be complete or even conformant to the specification. Such implementations will nevertheless identify errors in the test code, the test assertions and the specification from which they were derived. Again the ideal time to undertake beta testing is in parallel with the development of implementations. In this way the quality of the implementations and the test suite are improved in parallel. An ambiguity in the specification as captured by the example third assertion, may well be identified during beta test, when perhaps an implementer presumes that the escaped character in the variable string should be ignored.

- General release

  General release is when a test suite becomes sufficiently stable to enter into a formal maintenance phase, typically this may be measured by number of bugs outstanding, bug rate, an industry consensus decision or a combination of these. General release should logically mark the point at which a conformance test suite is deemed adequate to be the measure of conformance for implementations of the specification. Then it can form a basis of certification if desired. After general release should an implementation fail the example third

assertion, this could be a block to certification. A formal method must exist to resolve the meaning of this Fail result in respect to conformance. The Open Group calls these 'formal interpretations'. I shall talk more about how such false Fails relate to certification later in this paper.

- Maintenance release
  After general release at some point there will be a number of errors identified in the test suite. These could be errors in the test code, the test assertion, or the specification itself. At this point a maintenance release may be appropriate, and this typically takes place every six months. A maintenance release is designed to clear bugs, but also new tests may be added to 'plug' holes in test coverage. Such a maintenance release should again be subject to a beta review. In most cases a maintenance release will clear bugs and reduce the number of false fail verdicts and thus be 'easier' to pass. However a development team that is in the midst of debugging their implementation prior to certification may find the introduction of a new test suite adds confusion to any regression testing. For this reason it is typical to have a period of overlap when both the new and previous releases of the test suite are valid for certification purposes. The Open Group has a policy that this overlap shall be 6 months.

Certification Adding Value to Conformance Testing

I have discussed the value that conformance testing can bring to the development of conforming products and to the specification from which they are derived. I have also stated that conformance testing forms the basis of product certification. So what benefits does product certification bring? And how does it add value to conformance testing? If we are to presume that there will be multiple implementations of a standard then certification increases the Quality Assurance delivered by conformance testing and furthermore brings significant marketing advantages to vendors of conforming products.

- Objective independent review
  A set of test results does not in itself indicate conformity. Even conformant products may have results other than pass for some tests. Where options give rise to conditional tests this must be reflected in the correct parameterization of the test suite. A certification body can provide an unbiased independent review of the test results, as a distinct activity to the testing itself. This is particularly important in the case of self-testing by a development team when 'mindset' may engender errors to occur in the testing process.
- Simple criteria for procurement
  Conformance test results may constitute many mega bytes of data, which it is impractical for a procurer of a product to review. Furthermore the options within a specification may add complexity for assessing the potential to mix and match interoperating implementations from different suppliers. Certification can put test results into a simple statement of conformance and

available options. Purchasers of products need this to make procurement decisions. The availability of a simple certificate of conformance when combined with the perception of independent assessment, may be a powerful market enabler for the sale of conformant products

- Single Sheet
  A simplification of the conformance test result into a single page certificate may bear all the key information a potential purchaser needs to take a procurement decision. The certificate can be used by vendors in marketing materials and may even be shipped with the product. By these means certification has benefits to both the vendors of products and the users who will procure them.
- Register of products
  Complementary to the issue of a certificate is an entry on a searchable register of conformance products. Procurers may use this register to build complex systems of heterogeneous but conforming products from different suppliers.
- Continuing Conformance
  Type certification makes a statement about a product's conformance at a single instance of time. Some certification programs back certification by trademark law to deliver a warranty of continuing conformance. This significantly adds to the confidence built by the testing activity. The Open Group operates a certification program called The Open Brand that uses a legal contract called the Trademark License Agreement to deliver such a warranty.
- Lab accreditation
  Some certification programs go beyond requiring the use of a standard conformance test suite as a basis for certification. They also consider the environment in which the conformance testing takes place. By assessment of the test facilities quality system, the requirements of repeatability, reproducibility and impartiality can be ensured for the testing process itself. Again my The Open Group provides this service for some customers, and indeed it is my particular field of expertise.

In a quality assurance test suite, a bug is just a bug. In a conformance test suite it can determine whether a product is awarded or denied certification. The process, by which an applicant for certification can dispute a result other than 'Pass', must also be repeatable, reproducible and fair. A customer may apply to have a 'Fail' or 'Unresolved' result waived for certification purposes. Such a waiver must only be given if the result does not represent a non-conformance with the standard. The Open Group calls such an request to disregard a non-pass result for certification purposes a formal 'interpretation. The process by which the merits of a request for interpretation are reviewed must also be objective, with respect to certification. While I shall discuss The Open Group approach the principles apply also to other certification programs.

A non-pass result may be the result of a non-conformance in the implementation under test, but it may also be caused by a false fail path in the test code or an error or ambiguity in the specification.

Test Suite Deficiencies

The Open Group terms errors in the test suite that result in a false fail, a test suite deficiency . Once it is agreed that an interpretation request constitutes a test suite deficiency this can be referenced in support of applicant for certification by any vendor whose implementation produces the exact same behaviour as evidenced by the test journal. A test suite deficiency exists for the life of the version of the test suite which gave rise to it as the test suite error will be normally corrected in its next release. In the uncommon event that such an error cannot be corrected in the next release for technical reasons, then a test suite deficiency may apply to more than one specified version of the test suite.

Permanent Interpretations.

Errors in the specification from which the test suite is derived are classed as permanent interpretations or temporary interpretations. A permanent interpretation ordinarily spans all versions of the test suite and applies to the life of the specific version of the specification used for certification. This reflects the policy of The Open Group that products that are deemed conformant by virtue of the award of certification should not be later deemed non-conformant due to a subsequent clarification of the specification. Thus a permanent interpretation relaxes a specification never and tightens it. Such errors in a specification are published as corrigenda and normally are corrected in its next release.

Temporary Interpretations

A temporary interpretation is awarded when there is believed to be an error in a specification to which The Open Group defers and over which it has no control, an example is an international (ISO) standard. The temporary interpretation is awarded as an interim measure while the superior standards body considers the matter. If it is confirmed that The Open Group decision to grant an interpretation is correct then it becomes a permanent interpretation. If the determination is that the Open Group decision is incorrect the vendor of the certified product is given a period of time (No more than twelve months) to correct the implementation in line with the Standard Body's formal ruling.

The process of determining the outcome of an interpretation commences with an independent request for analysis by an expert. The Open Group's Conformance Quality Manager then further reviews this analysis and ensures any issues identified are resolved prior to the final ruling. If the matter is straightforward the ruling may be made on the basis of this twofold review. However if there is an element of doubt as to the correct ruling, the applicant's organisational and product details are removed. The

interpretation is then subject to industry review by the applicant's peer vendors before the final outcome is determined. Irrespective of the means by which the Open Group arrives at its ruling the applicant has a legal framework for formal appeal, though to date no such appeal has been invoked.

Agreed test suite deficiencies, permanent interpretations and temporary interpretations are published to all users of the conformance test suite by means of the World Wide Web. Thus granted interpretations can be referenced in support of an application for certification or can aid in the development of conformant products. Permanent interpretations normally result in corrigenda to the associated specifications comprising the standard and are published on the Open Group web site.

Conformance Statements

Conformance Statement Questionnaires are the means that The Open Group uses to define a product precisely, which conforms to the standard. This constitutes a formal description that is itself standard. Other certification programs have equivalent documents, for example the Wireless Application Protocol certification program uses the term Implementation Conformance Statement (ICS). A Conformance Statement complements the single page certificate by defining precisely how the standard has been implemented and provides additional information for procurers choosing between conformant products or designing systems composed of a variety of conforming products.

- Statement of options provided
  Specifications forming standards very often have functionality, which is optional for conforming implementations. Purchasers, when particular options are important for their application, may use the Conformance Statement to determine which certified products support each option.
- Means of implementation
  Specifications forming standards may allow some aspects of conformance to be implementation defined, for example certain resource limits. In the compiler world a standard may define a value, for the minimum number of nested loops that may be supported. Implementations may support a greater number of nested loops than that specified by the standard as the minimum. Purchasers intending to port a legacy application that exceeds the standards minimum number of nested loops may wish to know the implementation dependant maximum. Similarly it is common for certain error codes to be optional and users who may have applications that make use of such codes, may need to know what is supported by a given implementation. It should be stressed however that applications designers are advised not to assume the presence of functionality or limits beyond that mandated by the standard. This will ensure portability and interoperability between different conformant implementations.
- Scope of certification

A number of certification programs allow for derived certification of implementations that may differ in precise configuration from the actual implementation tested. For example operating systems may run on a series of hardware platforms without change. The Open Group has the concept of binary compatibility. In the Conformance Statement the supplier can define a series of product identifications, which are supported unchanged by the tested implementation, and for which a warranty of conformance is extended. Thus the Conformance Statement defines the exact identity of the implementation tested and any other products also covered by the certificate of conformance. It must be stressed that for certification to be extended beyond the tested implementation in this way, the software must be identical at the binary level as well as at the source level.

- Test suite parameterizationThe optional functionality, other implementation defined features and values are defined fully in the Conformance Statement. In many cases this is directly related to options within the associated conformance test suite. Thus the Conformance Statement forms part of the audit trail, which enables test results to be reproduced and repeated.

Those interested in viewing real life examples of Conformance Statements associated with the Open Brand certification program may do so at:
http://www.opengroup.org/csq/browse.mhtml

Trademark License Agreement The Trademark License agreement is a legal contract which customers of The Open Group certification program sign. It not only defines and restricts the usage of The Open Brand trademarks but also underpins the certificate itself with real commitments by the supplier to its customers.

- Standards of Quality
  The Trademark License Agreement defines a set of Standards of Quality, which are the devices by which conformance is established and maintained. These include:-
  - The use of standardized conformance test suites
    Where standardized test suites are defined for the product standard the vendor must provide satisfactory test results both on application for certification, and thereafter on demand
  - The requirement to provide and maintain a Conformance Statement
    The Conformance Statement must be maintained up to date during the life of the registration of the certified product
  - The Open Group’s rights to audit continuing conformance of certified products
    A requirement on the vendor to provide reasonable evidence of continued conformance on demand, not limited to test results
  - Notification of any known non-conformance

A vendor's obligations to bring any known non-conformance to the attention of The Open Group

- Continued conformance
  The obligation of the vendor to ensure any maintenance, or other subsequent releases, of the product remain in conformance within the scope of the products registration
- Time limit to clear any known non- conformance arising after certification
  The vendor's obligations to rectify a breach of the standards of quality, in a timely fashion
- Removal of certification for breach of conditions.
  The legal framework for removal of certification if a breach of the standards of quality is not rectified in the specified time.
- Changes to register product identity
  Procedures for a change of the product name of a registered product that is otherwise unchanged
- Anonymous review of interpretation requests
  The right to industry review of interpretation requests and associated confidentiality rights
- Formal appeals.
  .

- Warranty of conformance
  The vendor of a certified product is required to publicly "warrant and represent" that the product is in conformance with the standard. This warrant is absolute and goes beyond the requirement to pass specific test suites. In this respect The Open Group certification program provides greater confidence than that delivered by a 'type' certification program.
- Continued conformance
  Not only must a product be in conformance at the point of certification but also it must remain so thereafter, this includes any subsequent releases within the exact scope of this commitment as chosen by the vendor.
- Use of certificate and mark
  A definition of what may be claimed concerning the certification of the product and the trademark associated with this certification.

Assessment of Certification Applications

Conformance Statements

Conformance Statement are assessed for self-consistence and correctness. While options may be permitted by the specifications forming the standard, some options may be mutually exclusive, or may in other ways constrain each other. The assessment process ensures that the optional support of functions claimed in the Statement is valid. Similarly the specification may permit a range of values and the actual value

entered will be checked to ensure that it is within range of values defined as valid within the specification. At The Open Group we use Conformance Statement assessment procedures designed to meet the requirements of an ISO 9000 series quality standard. By this means we can ensure that the assessment of Conformance Statements is repeatable, reproducible and impartial. The Conformance Statement is also crossed-checked against the test suite parameters which have been selected by the test engineer to ensure consistency between the options for which conformance is claimed and the tests that have been selected.

Referenced interpretations

Interpretations may have been referenced by the applicant in support of a waiver of test results other than pass. Such referenced interpretations are also checked for validity. It is ensured that the interpretations cited are applicable to the specific tests and the version of the test suite used. It is also ensured that the test result is consistent with the example on the interpretation and that when the test has failed or did not complete, that it did so for the same reason as that described on the referenced interpretation.

Test report self consistency

The test report is checked as an entity, for correctness, completeness, and self-consistency. The setting of certain parameters should result in certain test cases being run. It also ensurs that all the tests have been run. Where appropriate it is ensured that the test report is produced in a single test run rather than being composed of several incomplete runs. Finally there is a cross check that all the tests which fail to report ‘pass’ are covered by valid references to approved interpretations or test suite deficiencies.

Scope and identity

The Conformance Statement, test results and the application forms are checked to ensure that the identity of the tested product is precisely defined and is consistent across all documents associated with an application for certification. Similarly the statements concerning binary compatibility (derived certification) and thus the scope of the applicants commitment to continuing conformance are evaluated to ensure the certificate bears an appropriate definition. A vendor may commit to the conformance of the specific versions of a product tested or may commit that subsequent version will also fall under this commitment. The Standards of Quality in The Trademark License Agreement define the implications of this choice. In any event, if there is a significant change to the functionality of a version of the product that may impact its conformance, this is considered a new product and the vendor is required to apply for a new certification. (Releases of a purely maintenance nature to clear bugs are normally included in the scope of certification.)

Legal agreement

The application forms are checked to ensure that a Trademark License Agreement is in force between the applicant and The Open Group for the products for which certification is applied.

UNIX

It was X/Open and more latterly The Open Group which has overcome the fragmentation of the UNIX operating system and contributed to its continued market lead as the back office application of choice. Standardization and associated testing and certification was the basis of this achievement

The UNIX system was envisioned as a single operating system to provide a standard application and user environment, however market forces caused diversification of UNIX system platforms perpetuating hardware and skills dependences for applications. Today a focus on conformance testing and certification has resulted in a standard and stable definition of a core application-programming environment – the UNIX Standard base; known as the Single UNIX specification. The addition of a Graphical User Interface termed the Common Desktop Environment with associated conformance testing and certification has provided PC style usability to top end systems. The UNIX system is the operation system of choice for the majority of Internet servers. The UNIX system continues to evolve in order to add innovation but does so in parallel with conformance testing and certification with the added advantage that the specification is designed for conformance assessment from the onset. High quality robust versions of the UNIX system are released at the same time as the general publication of the application portability environment, and the availability of conformance testing and certification. There are 35,000 Test cases available for the UNIX system family of products distributed over six test suites. Certification is available against four product standards.

- UNIX 98 - the mark for systems conforming to Version 2 of the Single UNIX Specification
- UNIX 95 - "*the standard base*" for Version 1 of the SUS
- UNIX 98 Workstation
  - UNIX 98 plus the Common desktop environment
- UNIX 98 Server
  - UNIX 98 plus Internet Server capabilities
  - Adds interoperability services to the UNIX 98 APIs in support of internet/intranet services

Within the product standards associated the UNIX trademark there is an option to support Realtime functionality. Future work is planned to address certification of realtime embedded systems using much of the certification tools used for testing the UNIX system."

Wireless Applications

The Open group has been chosen by The Wireless Application Protocol Forum (WAPF) to be its Certification authority for wireless web devices. These devices include cellular phones palmtops and other small telecommunication devices with text and graphic capability. Thus the expertise used in the certification of UNIX systems is applied to a diverse technology area being modified to meet the specific needs of the WAPF.

•Interoperability

Unlike the UNIX system certification that is mainly concerned with the portability of software applications between the UNIX system platforms of different suppliers, WAP certification is primarily concerned with the interoperability of telecommunications devices conforming to the WAP profiles, again from different suppliers. Conformance testing consists of end-to-end testing, and protocol testing in a multi-party interoperability scenario. In addition the correct rendition of text and graphics is also validated.

Many of the processes of the UNIX system certification also apply to the WAP certification program. The key differences are.

- The WAP Forum owns the specification and the trademark associated with the certification program. The Open Group operates the processes associated with certification on behalf of the WAP forum.
- Customers need not apply for interpretations in support of an application for certification. Problem reports against the test suite are classified as test suite deficiencies according to whether they may be used as a waiver for a non-pass result.
- In order to be awarded a certificate applicant must run the conformance test suite against at least two other supplier's products.

Futures

I can only talk about some of the areas that my own company is researching into. These include

FIX - A Protocol for exchanging financial information.

The Open Group is currently a member of a consortium of financial and telecommunications companies doing work in this area. FIX is an open protocol for real time exchange of electronic messages for communicating securities transactions between two parties. The Open Group is researching plans to provide certification services in the following areas:

a) Protocol Certification - certify correctness of protocol implementation relative to specification
b) Line of Business Validation - validate correct execution of protocol interactions required by higher-level business functionality (business scenarios)

c)

Directory Services

There is no single directory of Internet addresses. Various web browsers and mail servers have islands of Internet address directories but these are on diverse platforms running non-compatible database managements systems in which the data is held. The Open Groups Lightweight Directory Access Protocol LDAP is a means of enabling these directories to interwork and thus forming a worldwide network of distributed Interwork directories. Certification of applications making use of the LDAP protocol is a means to ensure their interoperability.

•Security

One of the areas where lack of consumer confidence is most definitely affecting the growth in the area of the financial transactions over the Internet. There are two main concerns:

- Security
  There are fears that access to sensitive information including trading and account identity can be 'hi jacked' on route via the Internet. News of recent co-ordinated attacks on key Internet retailers has done little to alleviate these fears. Certification of secure Internet trading systems may promote rapid market growth, provided that the specifications upon which these systems are built deliver the required security and confidentiality.
- Transaction determinism
  Financial trading is a real time activity, and it is crucial that a transaction takes place in a specified duration of time and that it is confirmed to have taken place at a specific time. The Internet is not connectionoriented protocol which means that a given data exchange can take place in anything from a few milliseconds to several minutes. Tracking the progress of transactions across public infrastructure is therefore required.

Another area where The Open Group is working on security is in Common Data Security Architecture CDSA. This Open Group specification is of a layer between the operating environment and the various security services to promote interoperability and portability of secure applications. Certification of the CDSA Application Programming interfaces or API's is intended to accelerate market take up of secure systems and growth in secure applications.

Other Open SpecificationsIn principle the techniques applied to the Open Groups existing certification programs can be applied to any open standard as evidenced by the application of skills acquired in the UNIX system program, to the wireless applications. Certification of IT & T product is a task that is never complete. New areas emerge at the leading edge of technology so there is a role for standardization and associated conformance testing and certification to 'oil' the wheels of market acceptance. This prevents the damaging market fragmentation that can occur if competitive forces are left unchecked. The role of an independent vendor neutral

organization such as The Open Group is to promote co-operation between product vendors to forge consensus on common standards and to act as the objective impartial arbitrator of conformance to these standards.

Conclusion.

Market lead

The impetus for conformance testing and certification is the market. Unless the certification scheme meets the need of the market then vendors will have little incentive to summit to the external quality assurance associated with a certification program. It is market pull that supports such a program. However there are also clear marketing advantages to vendors. A fragmented market of non interoperating competitive products will grow more slowly than one where agreement on common standards and a measure of conformance can engender consumer confidence. From a vendor perspective standardization and certification are of value when they deliver such confidence to users and help more products be sold. Logic suggests that selling a greater $ value of products is rather more important long term than selling a greater number of products than ones competitors. Selling this way does little to generate profit for the company! Thus vendors will co-operate on the definition and policing of Open Standards if it can be shown that this gives medium term benefits in potential earnings. There was a time when most nations had more than one airframe manufacturer, in an unregulated market. Today there are two major manufacturers worldwide but the total value of the commercial airline business is beyond the dreams of the air entrepreneurs of that past era. Furthermore independent certification and regulation has helped the consumer confidence necessary for such growth. We are beginning to see similar trends as there is an explosion of applications making use of the standardized services provided by hardware platforms and operating systems from a variety of vendors.

A few of the key words used in this presentation

<u>Impartiality</u>

Key to consumer confidence in conformance testing and certification by both users and vendors is a perception that the process is objectivity and impartial.

<u>Automation</u>

To the extent that automation as is practical in the conformance testing and the evaluation of the results, it is perhaps the most straightforward way to deliver impartiality and objectivity into these processes.

<u>Framework</u>

An important aid to efficient automation of conformance testing is a standard test framework such as The Open Group's TETware product. This automates many of the requirements of conformance test design, and development, by providing the services

required for test build run and test reporting. This frees engineering resources for the most important task of designing good tests.

Limitations

Conformance testing and certification is not however a panacea. With certain exceptions it does not address the quality of the conforming implementation in terms of performance or robustness. However this is in itself not a limitation rather an opportunity for competitions in value add features to stimulate innovation.

QW2K Paper 2M1

Mr. Joel Fleiss
(VeriTest)

The ABCs of Managing a Software Testing Project

BACK TO QW2000 PROGRAM

Key Points

- Test Management
- Test Methodology
- Test Guidelines

Abstract...

This presentation discusses a methodology which changes test management from an art to a science.

There are numerous books and articles on project management and software testing, but none provide a step-by-step methodology to simplify the management task. If you judge a software testing project that is 50% over budget, significantly late, or does not fully meet the requirements as a failure, then very few technical projects (less than 20% according to the Standish Group) are a success.

Like any Project Manager, a Test Manager's job is to maximize the use of the allocated resources in achieving the client's goals. If you want to minimize the number of software testing projects in your organization which exceed their original budget, improve your on-time record, and assure that your products fulfill their requirements, then this presentation will provide you with a better understanding of what steps to take in order to improve your organization's testing success rate.

The following are some of the questions that will be answered in "The ABCs of Managing a Software Testing Project" presentation:

- Why do projects fail?
- How do we create a test environment where projects are likely to succeed?
- What are the major software testing methodologies?
- What are the different testing goals?
- In a typical software project, what percentage is testing? What should it be?
- Is the percentage of testing changing for today's e-commerce, GUI applications?
- What class of software tools exist to support software testing?
- What are some of the key documents which should be created to support testing?

- What should a test management methodology consist of to assure project success?
- How do we simplify managing the test functions of project?

Author Bio...

Mr. Fleiss was the President of two software firms for over two decades. His initial firm, Proprietary Software Systems, Inc. provided software tools (compilers, assemblers, link editors, debuggers and real-time operating systems) to major defense contractors. His second firm, Quality Information Systems, Inc. developed and marketed a document management system for managing large document repositories. Mr. Fleiss was responsible for the product design and managed the implementation of both organization's products.

Mr. Fleiss graduated from the University of California at Los Angeles. After college he became a software engineer developing compilers, macro assemblers and operating systems. Recently he wrote a book on managing technical projects for a major financial institution. He is currently employed by VeriTest and supports their overall project management and marketing efforts.

BACK TO QW2000 PROGRAM

ABCs of Managing a Software Testing Project

- Why projects fail?
- How do you establish a successful test environment?
- What are the major software test methodologies?
- What are the different testing goals?
- What percentage of product development/maintenance is testing?
- Has e-commerce and GUI applications changed this percentage?
- What class of software tools support testing?
- What are some of the key documents used to support testing?
- What test management philosophy should exist to assure success?
- How do we simplify managing the test functions of a project?

ABCs of Managing a Software Testing Project

Why Test Projects Fail

- Lack of employing a viable methodology.
- Requirements not clearly defined.
- Inappropriate schedule.
- Unstable requirements.
- Lack of experienced Test Manager.
- Lack of experienced Test Engineers.
- Lack of motivation.
- Lack of commitment.

ABCs of Managing a Software Testing Project

Major Software Test Methodologies

- Unit Testing
- Component Testing
- Functional (Bach) Testing
- Stress Testing
- Compatibility Testing
- Standards Testing
- Logo Testing
- Qualification Testing (Preliminary/Formal)

ABCs of Managing a Software Testing Project

Establishing a Test Environment

- Corporate commitment.
- Test laboratory with proper computer systems.
- Appropriate software test tools.
- Test matrix.
- Quality Assurance procedures.
- Problem Report system.

ABCs of Managing a Software Testing Project

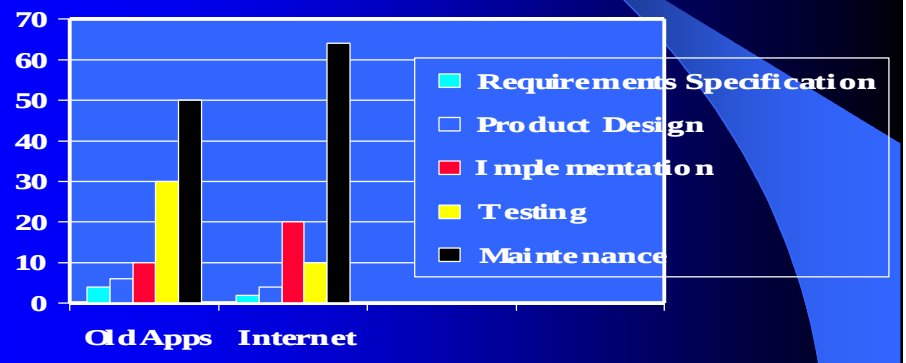
Testing Goals

- Thoroughness
- Timeliness
- Reasonable Costs
- Documented Anomalies
- Documented Work-Arounds

4/3/00 5

ABCs of Managing a Software Testing Project

Testing Percentages



4/3/00 6

ABCs of Managing a Software Testing Project

Test Tools

- Software Instrumentation
- Ghost Images
- Scripting Languages
- Software Emulators
- Debuggers
- Problem Reporting Systems

ABCs of Managing a Software Testing Project

Testing Documentation

- Test Plan
- Test Procedures
- Test Report
- Test Data
- Test Matrices

ABCs of Managing a Software Testing Project

Test (Project) Management Concepts

Planning	Organizing
Coordinating	Directing
Monitoring	Controlling
Supervising	Motivating

ABCs of Managing a Software Testing Project

Simplifying Test Management

- Documented Methodology
- Technically Knowledgeable Test Engineers
- Documented QA/CM Procedures
- Online Problem Report System
- Experienced Test Manager
- Comprehensive Test Document Templates
- Corporate Commitment

The ABCs of Managing a Software Testin

Introduction

Managing a testing project is not an exact science. Very rarely are two testing projects identical, nor is there an accepted testing methodology practiced by multiple organizations. There are multiple reasons why some testing projects run like clockwork, while others seem to drag on forever with little or no meaningful results. Too often Test Managers run projects by the seat of their pants instead of performing tasks in an orderly, documented manner. They spend the majority of their time putting out fires (current disasters). Testing projects are often complex, non-linear activities. A major function of a Test Manager is to provide the paradigm that maximizes the use of his/her resources in achieving the project's goals. It is the purpose of this paper/presentation to provide a methodology that will simplify the task of managing a software test project.

Why Can't Software Testing be Comprehensive?

Almost all software products (applications) can never be fully tested. This is often a difficult concept to grasp for the non-software oriented individual. The reason is a matter of numbers. If a software program has 100 different functions, then it is not a major task to create 100 test cases to verify the operation of each function. If you now say that not only must you verify that each function works perfectly by itself, but also you need to combine it with all the other functions to verify it still works. So you then create 10,000 tests (100 x 100). Daunting, but still possible. Now you say to really thoroughly test the program you need to at least test each function with 3 other functions and therefore create 100,000,000 tests (100^4). If each test takes 15 minutes to create, then we are looking at over 3,000,000 people days. This does not even count the time of running the test. If each Test Engineer's day costs as little as $400, then we have a $1,200,000,000+ test project for just the test creation. A program with 100 functions is a very simple, small application. Thus, we will always be burden with latent defects in software products, even products used by millions of users, like Microsoft Windows and Word.

Why Most Testing Projects Fail

Like other aspects of the software development cycle (e.g., design, development and maintenance), testing projects fail if you judge failure as not being:

- Within 40% of the estimated schedule.
- Within 40% of the estimated cost.
- At or near the anticipated quality.

The problem is that test projects are rarely delivered on time, within budget and have the required quality. The major reasons why projects fail are shown in the following table:

Number	Reasons Why Test Projects Fail
1	Lack of employing a viable methodology (test plan, procedures, test scripts, test matrix).
2	Requirements not clearly defined.
3	Unstable requirements.
4	Inappropriate schedule.
5	Lack of corporate commitment.
6	Lack of exhaustive verification (testing).
7	Inexperienced/unqualified Project Manager.
8	No experienced/motivated staff.
9	No measurability.
10	No or minimal testing tools.
11	Lack of appropriate equipment.

Project Failure Reasons

The majority of software testing projects, especially small ones (less than $100,000), use the Test Manager's intuition to determine what needs to be done to fulfill the testing requirements. This lack of using a viable methodology is the primary reasons why so many projects fail. Unrealistic expectations, changing requirements, impossible schedules, internal politics (lack of commitment), new technology and financial cuts are reasons Test Managers often state why their projects have failed. Yet the main reasons are not knowing what to do, how to do it and when to do it. It is the Test Managers responsibility to assure that the testing requirements are realistic, clear and not haphazardly changed on a day-to-day basis. Selecting the "right" team and having a realistic schedule are also critical to a test project's success.

The ABCs of Managing a Software Testin

Testing Methodologies

Software testing covers a broad spectrum of activities. The following table describes the more common testing activities.

Methodol	By Who	Description
Unit	Programmer	Testing of small functional units within the application.
Component	Programmer	Testing of multiple functional units comprising a component (e.g., a subroutine or function) within an application.
Functional	Test Engineer	Testing major functionality of application by rapidly determining major features. For a GUI application, this might be testing the menu commands for expected results.
Stress	Test Engineer	Testing limitations and performance.
Compatibility	Test Engineer	Testing compatibility of application with hardware/software products.
Standards	Test Engineer	Testing whether the application supports specific standards.
Logo	Test Engineer	Verifying the application fulfills the requirements of the logo program.
Preliminary Qualification	Test Engineer	Testing using test plans and test procedures as a precursory to formal qualification testing.
Formal Qualification	Test Engineer and QA.	Formal testing using Test Plans and Test Procedures with Quality Assurance witnessing expected results.

Test Methodologies

Establishing a Test Environment

A critical initial step in testing is the establishment of a test environment. The initial step is to obtain a commitment from the individual responsible for the budget (your boss, your boss' boss, etc.). It is important to obtain this commitment in writing, so when the resources are required there are no lengthy delays in obtaining approval. The next step is establishing the test laboratory with the proper computer systems. This is often an overlooked task, and few software engineers understand the importance of logistics and a comprehensive test environment. The test environment should include office space to support the appropriate:

- Computer systems.
- Peripherals.
- Support software systems.
- Software tools.
- Network/Internet Connectivity.
- Expertise.

Testing Goals

It is not the goal of a testing project to build a better mousetrap; rather its primary goal is to determine the problems that the product currently exhibits. Problems can be in terms of actual anomalies, poor documentation, confusing interfaces, incompatibilities, lack of responsiveness, etc. Testing activities goals include:

GOAL	DESCRIPTION
Thoroughness	Are all functional aspects of the product tested in all possible ways? How many combinations of functions were tested together?
Timeliness	How long did the testing take relative to the schedule?
Costs	How expensive was the testing? Did it exceed the testing budget?
Anomalies	Did the test case uncover a reasonable number of anomalies? Were the documents clear for the software engineer to understand and was supporting evidence provided?
Work-Arounds	Were work-arounds provided for anomalies? Did they satisfy the problem as a temporary solution?

Testing Goals

Changing Percentages

In software developments initial decades (1950s through the 1980s), testing was always the major resource/time consumer when you considered design, implementation and testing. With the popularization of the PC and even more so with the popularity of the Internet coupled with the rush to market, the percentage of time spent testing as compared to product design and implementation has decreased dramatically. Recognizing that

testing is a slow, meticulous process and at best a triage as to what level of testing to perform, businesses in their rush to market have minimized the testing effort.

The net effect of this is that we now have products that constantly exhibit malfunctions. With a less sophisticated user population, many are left confused as to how to work around all the anomalies even our most popular products exhibit (e.g., Microsoft Word). Where in the 1950s through the 1980s testing represented more than 60% of the product development cycle, for most Internet products its lucky if testing represents 30% of the development effort. Who suffers from this change in percentages? Obviously, it is the user.

E-Commerce and GUI Applications

The 1990s brought two new activities that are critical to today's software business, e-commerce and GUI (graphical user interface) applications. GUI applications complicate the testing process because it is difficult without a powerful emulation tool to thoroughly test a GUI application in a minimal amount of time. Because of the constant human interaction GUI applications employ, it is very time consuming to verify applications validity. E-Commerce has popularized concurrent processing. Similar in concept to real-time operating systems, this activity is inherent with unique problems because of the added "time" factor of the order of events. In applications pre-1990 using real-time operating systems, these systems were subject to tremendous testing efforts to verify their algorithms under varying circumstances. Now days it is not uncommon for a complex web-application to be developed in N months and tested for a few weeks prior to it "hitting the market." A major culprit here is greed to capitalize on the investment as soon as possible.

Software Support Test Tools

Software Engineers have always developed tools to simplify the task of designing, creating, testing and maintaining software. The exact software tools used for an application depend on the general functionality of the product. The following table briefly describes some of the tools commonly used during the testing phase of a software application.

TOOL	DESCRIPTION	FEATURES
Instrumentation	Instrumentation tools modify the program to include code to facilitate debugging, testing and timing. This can be a special program that modifies the source to include calls to specific routines or a compiler that performs a like operation.	Should facilitate tracing an applications path, variable modification, memory utilization, etc.
Ghost Images	One of the scientific method's major objectives is to control the environment. Ghost images allow us to capture the environment at a known state so that we minimize the number of changing variables.	Should simplify restarting from a known environment.
Scripting Languages	Scripting languages became popular with the advent of GUI. They eliminate the time consuming task of using the mouse and keyboard to designate specific sequences of user inputs.	A scripting language should be able to emulate anything you can do with the mouse and keyboard. It should also contain facilities to compare the current window against an expected results window and report the results.
Software Emulators	An emulator simulates the action of another product. Most software emulators will emulate a piece of hardware, be it a PC or a peripheral.	The emulator should perform the operation as near identical as the product would perform it. A major problem with emulators is time. Most emulators cannot perform their operation as rapidly as the product being emulated can, and thus may introduce or not discover specific time related problems.
Debuggers	Programmers use debuggers to help solve problems.	Debuggers usually allow the programmer to view variables at specific locations within a program, trace the path of a program, time how long a specific set of instructions takes, etc. More

TOOL	DESCRIPTION	FEATURES
		powerful debuggers provide these features on a conditional basis.
Problem Reporting Systems	Problem Reporting Systems allow users to record problems.	A problem reporting system should allow for the problem to be stated, the environment in which the problem occurred, step-by-step procedure for recreating the problem along with whatever data is required, work-arounds if possible and contact information.
Measurement Tools	Measurement tools facilitate timing, counting and usage.	The ability to time specific regions of code, to count how often certain code is used, and to determine the percentages different parts of the application are consuming.
Complexity Analysis	An analysis of the source program can usually provide a reasonable estimate as to the number of problems expected and the difficulty in correcting the problems.	Complexity analysis usually involves a combination of the nesting depth of the program (loops, conditionals, switches), use of recursion, iteration, lack of comments, etc.

Testing Documentation

Documentation is a critical element of any project. The testing documentation is critical, because it provides a plan for how the testing is to be accomplished along with documented evidence of the testing completed.

The specific format and title used for the test documentation is not as important as the content being provided. The following table contains the key testing documentation that should be an integral part of any software development effort.

TITLE	DESCRIPTION
Test Plan	This is a design documentation that describes the testing environment, standards to be used (if any), responsibilities, strategy, test design (including verification method) and a matrix (see Test Matrix below).
Test Procedures	This is a detailed description of each test, with sign-offs and a step-by-step procedure for how each test is to be accomplished. It should contain the test matrix with expected results and the associated test data to be used for each test case.
Test Matrix	The test matrix is probably the most critical element of the testing process. It should contain the following information: • Test Identifier • Test Description (what is being tested) • Associated Test Data • Expected Results • Test Results (PASS, FAIL, OTHER) • Reference Documents (e.g., problem report identifier)
Test Data	This is the actual test data (input files, scripts, etc.).
Problem Reports	This document is used to describe anomalies. It should include the following information: • Test Identifier • Problem Description. • Test Environment • Tester's Name and Contact Information • Problem Status • Verification Method • Work Arounds • Special Comments
Test Report	This document summarizes the testing activity. It should include a completed test matrix along with a summary of all problem reports.

Test (Project) Management Concepts

CONCEPT	DESCRIPTION
Planning	A critical part of any technical effort is the planning phase. The planning phase output should be a Test Plan that describes in general terms what should be done and how it should be accomplished. Prior to commencing any task, it is important to have a clear definition of what the effort entails. The goal of nearly all test projects is to: (1) deliver the testing results on or ahead of schedule, (2) deliver the testing results at or better than the specified quality, and (3) deliver the testing results at a cost within or lower than the estimated budget. Verbal directions or instructions are often interpreted differently, especially as time passes. A documented (written or electronic) test plan simplifies communication for what is needed. **With a stable test plan, the project proceed in an orderly, predictable and precise manner.**
Organizing	Organizing and using the resources provided for a testing activity should be done at the start of the project and at any point in the project where changes occur which effect the schedule.
Coordinating	Making sure the appropriate communication exist between the different interfaces of the test project and coordinating their activities is a major function of the Test Manager.
Directing/Empowering	Creating a project plan with delineated tasks and informing the Test Engineers is critical to maintaining schedule commitments. It is often necessary to empower staff members so that they can accomplish their tasks in the proposed time frames. Management's commitment is crucial to the future success of a test project. We could accomplish commitment for all team members through a series of pep talks, or from a speech from one of our charismatic leaders, or through the actions of our leaders or through a written policy. Although all of the above methods may temporarily motivate test team members to commit, a more specific, recordable measurement of commitment is still needed. Often we assign a staff member authority without documented approval. Some may question why a project should have a documented commitment. There are often discussions about the requirements and deliverables in complex testing projects. Commitment without concrete action and documented proof is not commitment. It's marketing hyperbole. Any experienced Test Manager has probably heard such statements as: • "I never read the test plan." • "I never committed to that requirement." • "I never promised that ." • "What you mean you needed the proper resources to finish the task on time?"
Monitoring	Status reports, time sheets, walkthroughs, periodic meetings (kickoff, status or specific technical topics) along with occasional one-on-one meetings are the basic methods to be used for managing a test project. A critical part of a project is the monitoring of each task's progress. Some might call this proactive listening. It provides the data needed for the Project Manager to make rational decisions to effectively employ his/her resources. It also encourages problem solving at all levels. If a team member knows that their problems will be acted upon when they communicate their needs, then this knowledge will likely increase the probability that they will communicate their problems and suggestions for project improvement. A key part of a Project Manager's monitoring activities is learning how to filter out noise. Intelligent listening and reviewing team member's activities is not a simple activity. The Project Manager must be able to discern what is germane to the project's success. This facilitates eliminating what has no consequences relative to improving future performance and profitability or the current project success. Time management is a vital aspect of project management and individual productivity.
Controlling/Supervising	It is up to the Test Manager to control the activities of his/her staff. This includes making sure that each staff member is working on the appropriate assignments. To accomplish this, it is critical that each staff member have more tasks at any one time than they can accomplish, and that the tasks are prioritized.
Motivating	Motivating staff members to perform as best as they possibly can is a difficult challenge. What motivates one individual may have no or the opposite effect on another staff member. The reason for motivating your staff is to improve productivity. An initial step is helping team members personalize their goals and objectives; as the Test Manager, you need to be cognizant of what motivates your staff. The results will lead to earlier deliveries, less project costs, improved quality and most important to your corporate bean counters, more profits. There are numerous ways to

	motivate your staff.

Simplifying Test Management

In the previous sections we discussed some of the aspects of managing a test project. This section discusses the "what and how" of managing a test project.

The following table contains a step-by-step process for managing a test project. It should be noted that some of the steps are iterative.

STEP	TITLE	DESCRIPTION
1	Definition	Create a test plan that describes what is to be tested, where and with what it will be tested and how it will be tested. This usually should be derived from a requirements document. If a requirements document is not available, then a user guide or user reference manual is a reasonable substitute.
2	Implementation Plan	Create test procedures detailing how each test will be conducted and documented. A comprehensive test matrix with projected results is a critical aspect of this step.
3	Schedule	Create a project plan specifying the tasks, schedule and resources to be used for the testing.
4	Implement	Monitor that the testing staff is implementing the test procedures within the constraints of the schedule.
5	Report	Document all testing and problems.

QW2000 QuickStart 2Q

Mr. James Bach
(Satisfice, Inc.)

The Heuristic Approach to Testing

BACK TO QW2000 PROGRAM

Presentation Abstract

Heuristic testing is the systematic use of guidelines, checklists, and other empirically and theoretically derived tools to help testers design effective tests.

Heuristic testing focuses on how testers think, and since it minimizes documentation it's particularly useful in rapid, chaotic, or ambiguous situations that increasingly characterize modern software development. Heuristic testing focuses on skills and ideas, rather than procedures and metrics. This talk examines the principles, skills, and tools of heuristic testing.

About the Author

James Bach (http://www.satisfice.com) is founder and principal consultant of Satisfice, Inc. James cut his teeth as a programmer, tester, and SQA manager in Silicon Valley and the world of market-driven software development. He has worked at Apple, Borland, a couple of startups, and a couple of consulting companies.

Through his models of Good Enough quality, exploratory testing, and heuristic test design, he focuses on helping individual software testers cope with the pressures of life in the trenches and answer the questions "What am I doing here? What should I do now?"

BACK TO QW2000 PROGRAM

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Heuristic Software Testing

James Bach, Satisfice, Inc.
james@satisfice.com
www.satisfice.com
(540)631-0600

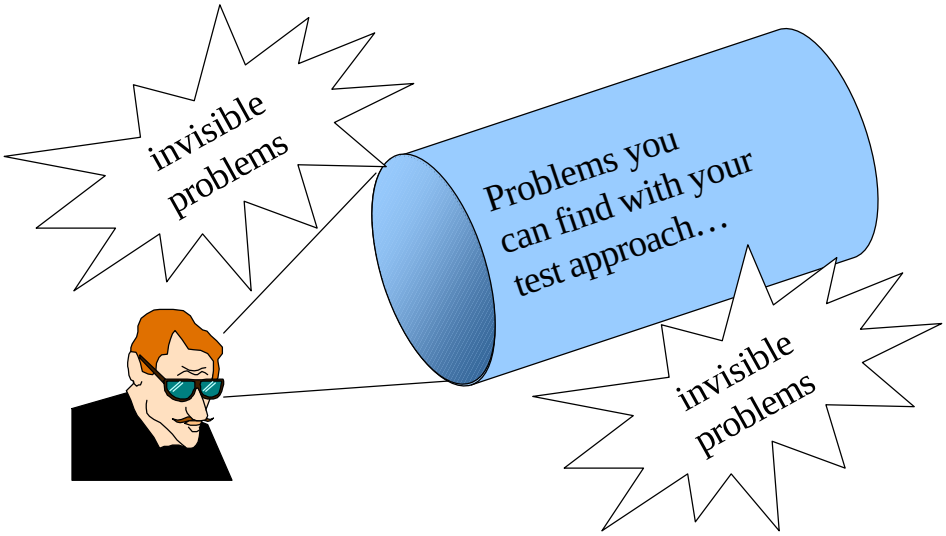
One way to conceive of testing...

Testing is the **art**
of evaluating the **invisible**
against the **ambiguous**
so as to avoid the **unthinkable**
happening to the **anonymous**

This definition reflects how it *feels* to do testing under typical working conditions.

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Tunnel-Vision is Our Great Occupational Hazard



Heuristics: Ways to Escape From Tunnels

- **adjective:**

  "serving to discover."

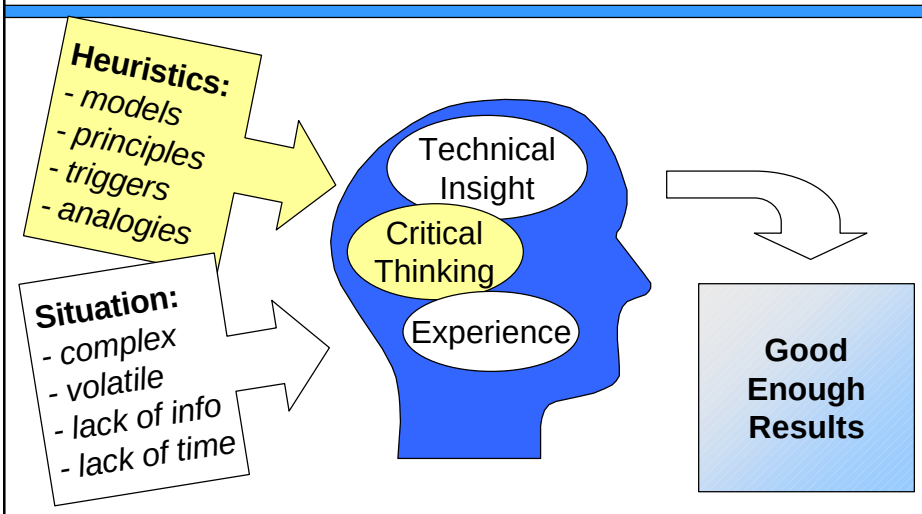
- **noun:**

  "a useful method that doesn't always work."

"Heuristic reasoning is not regarded as final and strict but as provisional and plausible only, whose purpose is to discover the solution to the present problem."
- George Polya, *How to Solve It*

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Heuristic Testing is a Personal Skill for Any Situation



Why Not Rigor and Structure? We May Not Have a Choice...

- Staff
  - Not enough people
  - Wrong skills
- Risk
  - Unknown risk
  - High risk
- Support
  - Lack tools
  - Lack information
  - Lack influence
  - Lack popularity
- Time
  - Not enough
- Product
  - Too invisible
  - Too volatile
  - Too complex
  - Too brittle
  - Too unreliable
  - Not controllable

What works when nothing else can?
Our minds.

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Testing is NOT a Sub-Field of Computer Science

The study of testing goes back at least to Aristotle.
Before there was software to test, there were *theories* to test.

- **Epistemology** *(hypotheses and logic)*
- **Cognitive Psychology** *(thinking and perception)*
- **Economics** *(allocation of scarce resources)*
- **Decision/Game Theory** *(rational decisions)*
- **Forensics/Rhetoric** *(persuasive arguments)*
- **General Systems Theory** *(rapid analysis of systems)*
- **Probability Theory** *(estimations of risk)*

Epistemology Lesson: Abductive Inference

- Abductive inference is the process of finding the best explanation for a set of data.
- Abductive inference is not the same as logical deduction. Abductive inferences from factual data are not necessarily true.
- Good abductive inference involves generating and evaluating alternative hypotheses.
- One good approach to abductive inference is *conjecture and refutation*…

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Epistemology Lesson: Conjecture and Refutation

- We never know product quality for certain.
- We conjecture about quality. To conjecture is to explore plausible realities.
- A good conjecture is *falsifiable*: we can imagine facts that would refute the conjecture.
- A conjecture can never be proven, only corroborated. Corroborating evidence is most interesting when gained as the result of a genuine attempt to refute.
- Good testing is a serious attempt to refute and corroborate conjectures about product quality.

General Biases

- **Beliefs:** Ideas or mental models that we treat as true. We tend to persevere in our beliefs. We are often not aware of the beliefs that influence our thinking.
- **Prominence:** We give greater weight to prominent evidence. We treat recent, disturbing, or familiar evidence as more prominent. It's hard to see omissions and subtleties.
- **Assimilation:** We assimilate evidence into our preconceived beliefs or explanations. We ignore contrary evidence. We treat things that look alike as if they are alike.
- **Self-Interest:** We look harder for conclusions that favor our self-interest. We practice wishful thinking.

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Some Domains of Thinking that are Subject to Bias

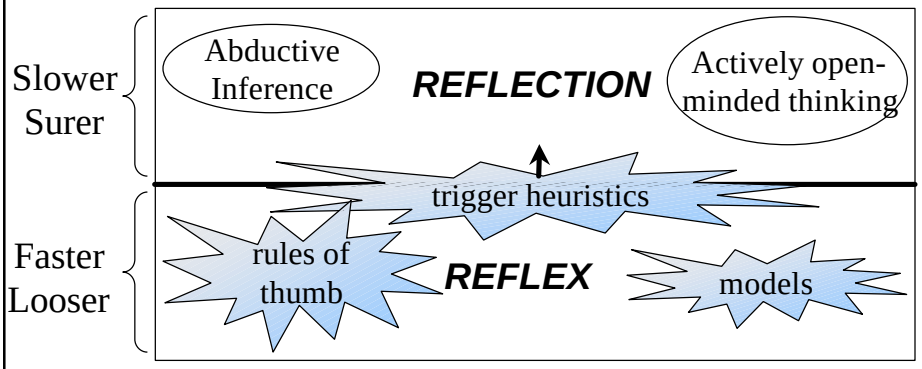
- Perceiving phenomena
- Investigating causality
- Judging probabilities
- Judging severity of problems
- Judging how other people think or feel
- Deciding what to do
- Judging success and failure

Some Everyday Heuristics

- It's dangerous to drink and drive.
- A bird in hand is worth two in the bush.
- Nothing ventured, nothing gained.
- Sometimes people stash their passwords near their computers. Try looking there.
- Stores are open later during the Holidays.
- If your computer is behaving strangely, try rebooting. If it's very strange, reinstall Windows.
- If it's a genuinely important task, your boss will follow-up, otherwise, you can ignore it.

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Heuristics Help You Rapidly Make Sense of a Situation



How Heuristics Differ from Other Procedures or Methods

- Heuristics are known to be wrong, at least some of the time.
- No one can say for sure when a heuristic will work.
- Heuristics aid or focus an open-ended problem-solving or solution-searching effort.
- Heuristics can substitute for complete or rigorous analysis.

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Using Heuristics

- **Awareness:** Practice noticing the heuristics by which you test.
- **Alternatives:** Learn diverse and opposing heuristics.
- **Triggers:** Practice bringing them to mind when you need them.
- **Know-When:** Question the dynamics of each heuristic, so that you know when it's best to apply it.

Heuristics are Biases

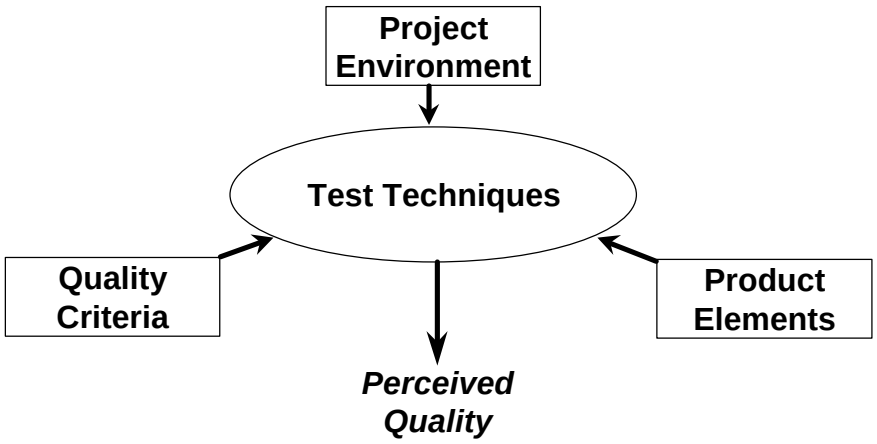
- **Beliefs:** A heuristic is a belief. Heuristics may affect us in ways we aren't conscious of.
- **Prominence:** Heuristics tend to stick in our minds. They become reflexes.
- **Assimilation:** We tend to assume that our heuristics fit every situation. Especially if our situations don't change that much. We tend to distrust heuristics that come from other people.
- **Self-Interest:** We want our heuristics to be right, so we don't always look hard enough for their flaws.

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

A Heuristic Model of

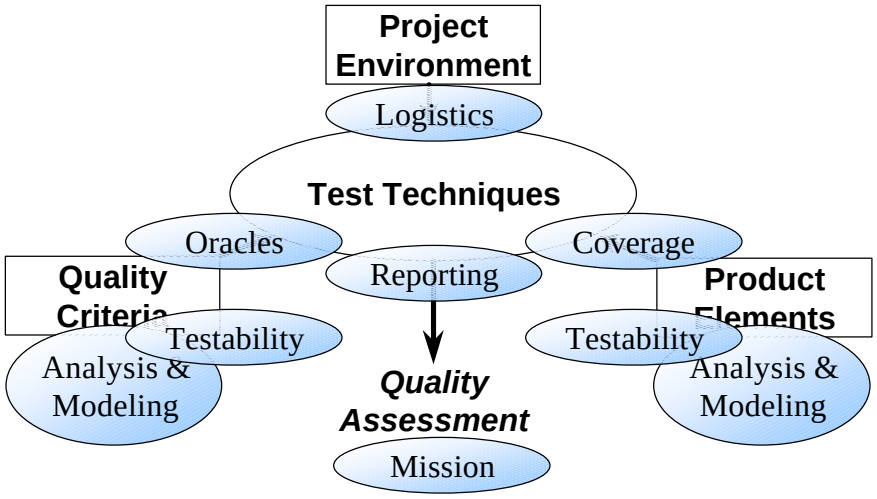
Testing

All Product Testing is Something Like This



Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Major Themes in Test Design and Execution



Testing Activities

- Analyze the product.
- Model the test space.
- Select what to cover.
- Determine test oracles.
- Configure the test system.
- Operate the test system.
- Observe the test system.
- Evaluate the test results.

A test technique is a recipe for performing these tasks.

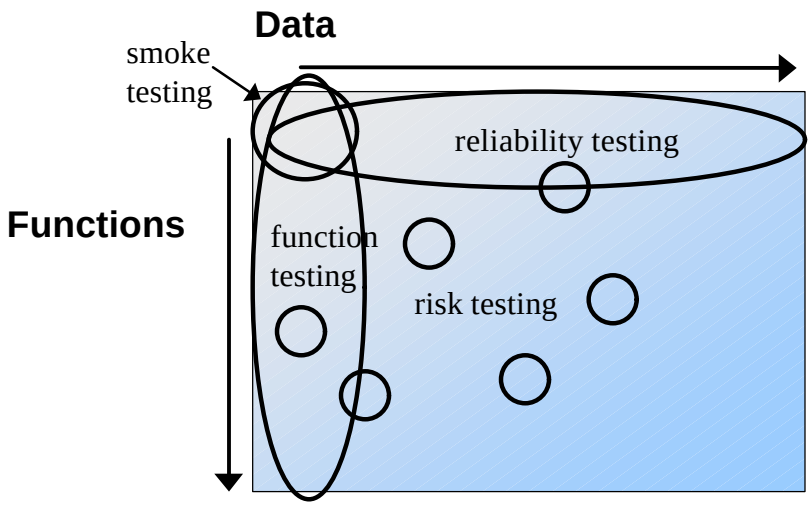
Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Coverage

Product coverage is the proportion of the product that has been tested.

- **There are as many kinds of coverage as there are ways to model the product.**
  - Structural
  - Functional
  - Data
  - Platform
  - Operations

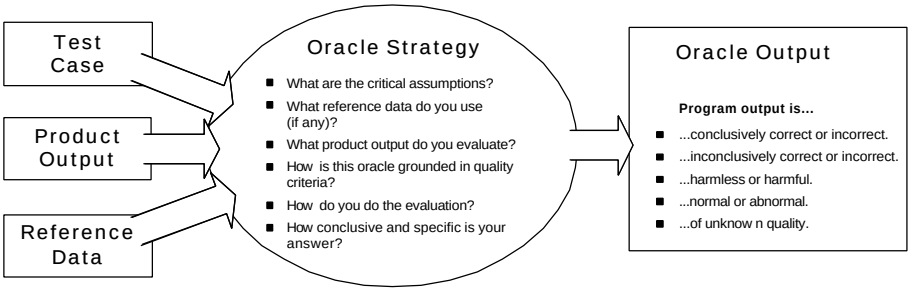
Coverage Heuristic: The Function/Data Square



Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Oracles

An oracle is a strategy for evaluating whether a test has passed or failed.



Quality Criteria

- **Importance:** how critical is the fulfillment of this expectation?
- **Source:** where does it come from?
- **Authority:** how does it relate to the stakeholder?
- **Testability:** how can you know whether it is fulfilled or violated?
- **Applicability:** to what product elements does it apply?

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Quality Criteria: Specifications

A specification is any reference that can be used to evaluate quality.

- **Explicit:** a reference that is formally acknowledged.
- **Implicit:** a reference that is *not* formally acknowledged.

Oracle Heuristic: Consistency

- **Consistent with Purpose:** Function behavior is consistent with its apparent purpose.
- **Consistent within Product:** Function behavior is consistent with behavior of comparable functions or functional patterns within the product.
- **Consistent with Comparable Products:** Function behavior is consistent with that of similar functions in comparable products.
- **Consistent with Claims:** Function behavior is consistent with what people say it's supposed to be.
- **Consistent with History:** Present function behavior is consistent with past behavior.

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Oracle Trigger Heuristic: "It Works"

"...it works."

really means

"...it appeared to meet some requirement to some degree."

Which requirement? Appeared how?
To what degree? Perfectly? Just barely?

What was the oracle?

The Universal Test Procedure

"Try it and see if it works."

- Learn about it
- Model it
- Speculate about it
- Configure it
- Operate it

Models

- Know what to look for
- See what's there

Coverage

- Understand the requirements
- Identify problems
- Distinguish bad problems from not-so-bad problems.

Oracles

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

General Test Techniques

- Domain testing
- Stress testing
- Flow testing
- User testing
- Regression testing
- Risk testing
- Claims testing

I call these techniques "general" because they are common, widely applicable, and can be combined with many other techniques.

Domain Testing

Key Idea: *"Try ranges and options"*

Summary:
- **A domain is a range or set of potential data.**
- **Identify each domain.**
- **Analyze limits and properties of each domain.**
- **Identify combinations of domains to test.**
- **Coverage strategy:**
  e.g., *exhaustive, typical, boundaries, random, invalid, etc.*

Good for: all purposes

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Stress Testing

Key Idea: *"Overwhelm the product"*

Summary:
- **Select test items and functions to stress.**
- **Identify data and platform elements that relate to them.**
- **Select or generate challenging data and platform configurations to test with:**
  - **e.g., *large or complex data structures,* high loads, long test runs, many test cases**

Good for: performance, reliability, and efficiency assessment

Flow Testing

Key Idea: *"Do one thing after another"*

Summary:
- **Define test procedures or high level cases that incorporate multiple tests.**
- **Dont reset the system between tests.**
- **Incorporate the element of time.**
- **Combine with other techniques:**
  - **e.g., *user flow, stress flow, risk flow***

Good for: finding problems fast
(however, bug analysis is more difficult)

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

User Testing

Key Idea: *"Strive for realism"*

Summary:
- **Identify categories of users.**
- **Determine what each category of user will do, how they will do it, and what they value.**
- **Get real user data, or bring real users in to test.**
- **Otherwise, systematically simulate a user.**
- **Do not assume that you are a user.**

Good for: all purposes

Regression Testing

Key Idea: *"Repeat testing after changes"*

Summary:
- **Identify what product elements changed.**
- **Identify what elements could have been impacted by the changes.**
- **Coverage strategy:**
  e.g., *recent bug fixes, past bug fixes, likely elements, all elements*

Good for: managing risks related to product enhancement

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Risk Testing

Key Idea: *"Find big bugs first"*

Summary:
- **Risk is the product of likelihood and impact.**
- **Determine what makes a bug "big."**
- **Analyze risks using experience with similar products and advice from customer.**
- **Focus testing on highest potential risks.**
- **Use test results to refine the risk analysis.**

Good for: making best use of testing resources; leveraging experience

Claims Testing

Key Idea: *"Verify every claim"*

Summary:
- **Identify specifications (implicit or explicit).**
- **Analyze individual claims about the product.**
- **Ask customer to clarify vague claims.**
- **Verify each claim.**
- **Expect the specification and product to be brought into alignment.**

Good for: simultaneously testing the product and specification, while refining expectations

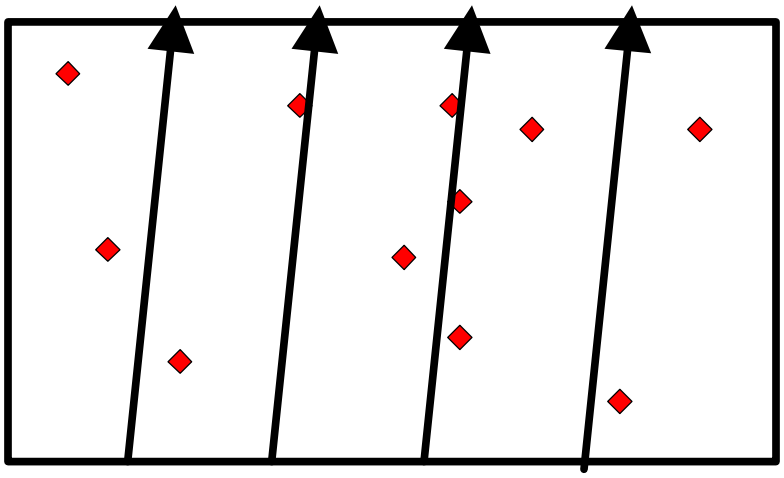
Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Test Strategy Heuristic: Diverse Half-Measures

- There is no single technique that finds all bugs.
- We can't do any technique perfectly.
- We can't do all conceivable techniques.

Use **"diverse half-measures"--** lots of different points of view, approaches, techniques, even if no one strategy is performed completely.

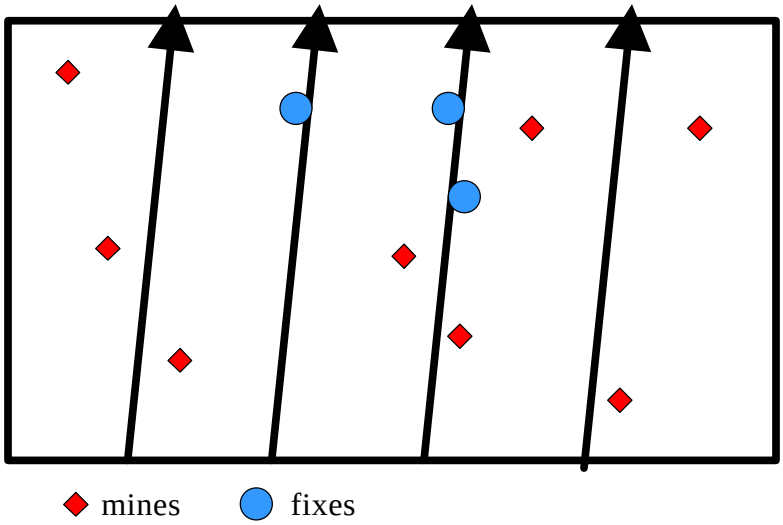
Test Strategy Heuristic: Clearing Mines



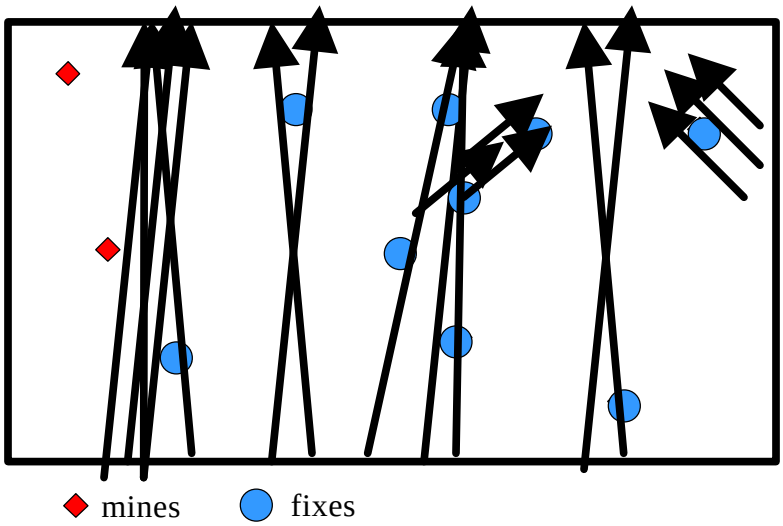
mines

Copyright © 1995, Software Testing Laboratories, Inc., all rights reserved
James Bach, jamesb@stlabs.com

Totally repeatable tests wont clear the minefield



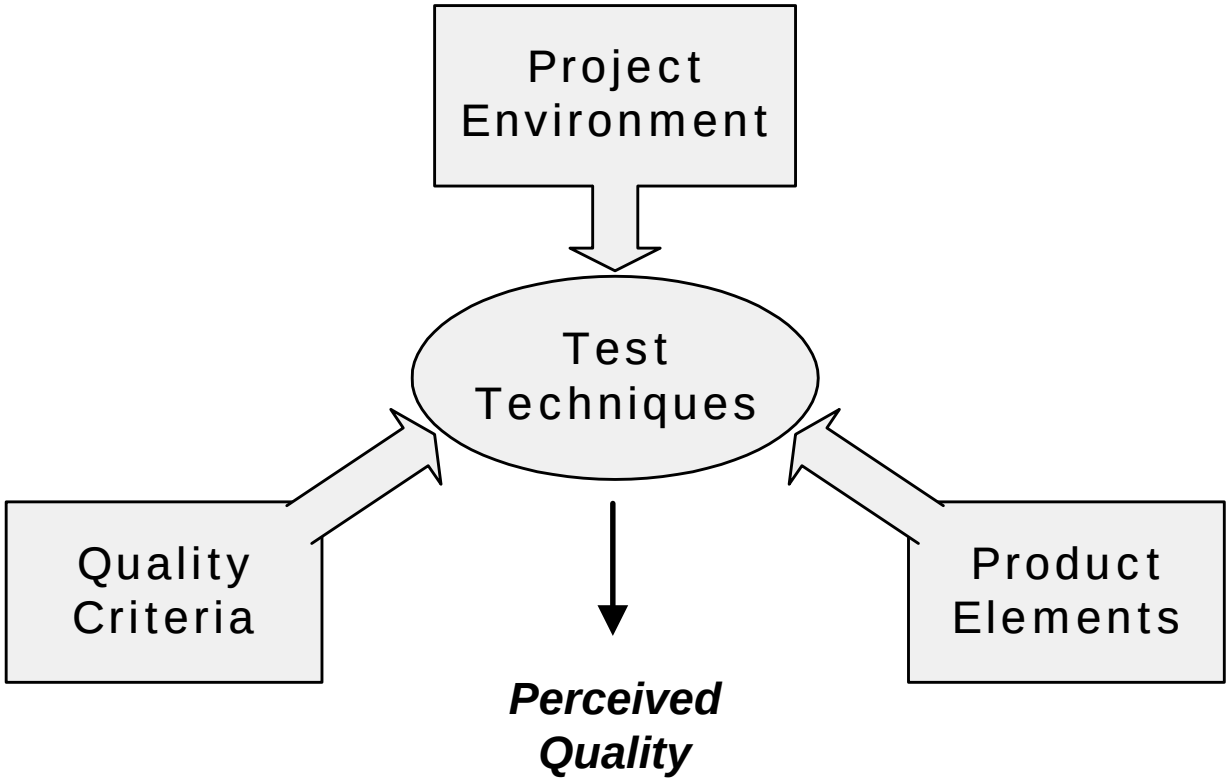
Variable Tests are Often More Effective



Designed by James Bach
james@satisfice.com
www.satisfice.com
Copyright 1996-2000, Satisfice, Inc.

Version 2.5
2/11/2000

Heuristic Test Strategy Model



The **Heuristic Test Strategy Model** is a set of patterns for designing a test strategy. The immediate purpose of this model is to remind testers of what to think about when they are creating tests. Ultimately, it is intended to be customized and used to facilitate dialog, self-directed learning, and more fully conscious testing among professional testers.

Project Environment includes resources, constraints, and other forces in the project that enable us to test, while also keeping us from doing a perfect job. Make sure that you make use of the resources you have available, while respecting your constraints.

Product Elements are things that you intend to test. Software is so complex and invisible that you should take special care to assure that you indeed examine all of the product that you need to examine.

Quality Criteria are the rules, values, and sources that allow you as a tester to determine if the product has problems. Quality criteria are multidimensional, and often hidden or self-contradictory.

Test Techniques are strategies for creating tests. All techniques involve some sort of analysis of project environment, product elements, and quality criteria.

Perceived Quality is the result of testing. You can never know the "actual" quality of a software product, but through the application of a variety of tests, you can derive an informed assessment of it.

Project Environment

Creating and executing tests is the heart of the test project, and is what most people think of as "software testing." However, there are many factors in the project environment that are critical to the successful completion of a testing project. In each category, below, consider how that factor may help or hinder your testing. Be aware of what's going on in the project. Try to exploit the resources you have available, while minimizing the impact of constraints.

Customers

Anyone who is a client of the test project.

Information

Information about the product or project that is needed for testing.

- **Availability:** Do you have all the information from the developer and others that you need in order to test?
- **Volatility:** Is that information current? How are you apprised of new or change information?

Team

Anyone who will perform or support testing.

- **Size:** Do you have enough people to complete all planned testing within the desired time frame?
- **Expertise:** Do you have people with the right knowledge to satisfactorily complete the planned testing?
- **Organization:** Are the testers coordinating their efforts and pulling for the same goal?

Budget

Money needed to purchase testing resources and materials.

Equipment & Tools

Hardware, software, or documents required to administer testing.

- **Hardware:** Do we have all the equipment and platforms we need in order to execute the tests?
- **Automation:** Are any test automation tools needed? Are they available?
- **Probes:** Are any tools needed to aid in the observation of the product under test?
- **Matrices & Checklists:** Are any documents needed to track or record the progress of testing?

Processes

The tasks and events that comprise the test project.

Schedules

The sequence, duration, and synchronization of events.

- **Testing:** When will testing start and how long will it take?
- **Development:** When will builds be available for testing, features added, code frozen, etc.?
- **Documentation:** When will the user documentation be available for review?

Test Items

The product to be tested.

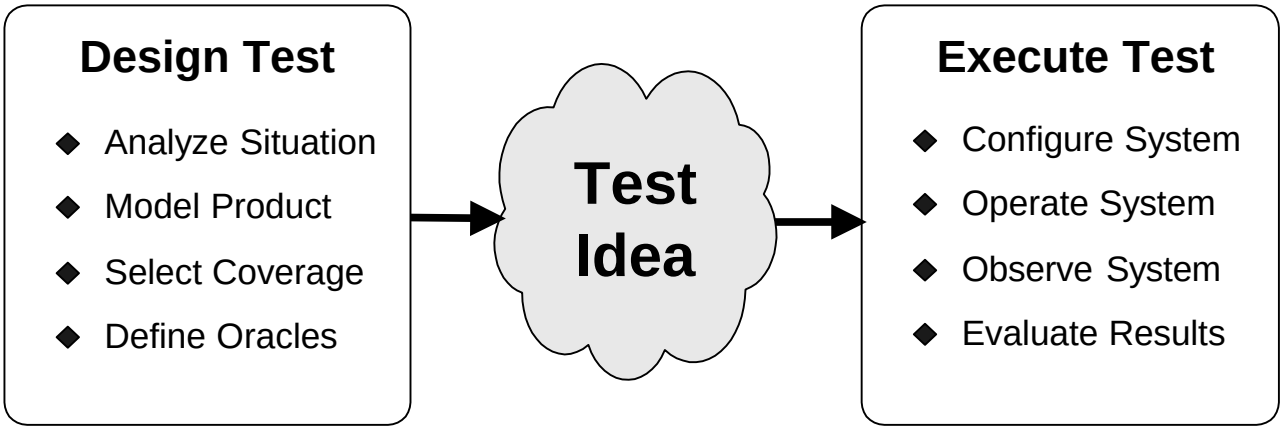
- **Availability:** Do you have the product to test?
- **Volatility:** Is the design and implementation of the product constantly changing?
- **Testability:** Is the product reliable enough that you can effectively test it?

Deliverables

The observable products of the test project.

- **Content:** What sort of reports will you have to make? Will you share your working notes, or just the end results?
- **Media:** How will you record and communicate your reports?

Test Technique Elements



Test Idea: Any idea that guides test execution. It is not necessarily fully articulated, recorded,or maintained.

Analyze Situation: Identify, examine, and clarify information about project environment, quality criteria, and product elements.

Model Product: Construct a test model-- a depiction or representation of something you intend to "cover" with your tests. Example: for risk-based testing, your model might be a list of risk areas.

Select Coverage: Determine what you will cover in the test model. In other words, decide what parts of the product must be tested, and what parts you will leave alone.

Define Oracles: An oracle is a strategy for knowing whether a test passes or fails. Without one or more oracles, you aren't really testing. Define your oracle strategy for your tests.

Configure System: Prepare the product and platform for testing. Assure that the system is in the proper state to commence the test.

Operate System: Provide the input and control to the system necessary to carry out the test. For some kinds of testing, this may require special tools.

Observe System: Monitor the output, or any other relevant effect of the operation of the system. This may require special tools in order to observe hidden variables or subtle effects.

Evaluate Results: Use your oracle strategy to evalute test results. Perform additional testing as needed to confirm your evaluation.

General Test Techniques

A test technique is a way of creating tests. There are an indefinite number of interesting techniques. The list below shows seven *general* techniques. These techniques were chosen because each of them is conceptually simple, unique, and often incorporated into other techniques. Each one can be performed in a quick and dirty way, or much more formally. They can also be combined with each other (e.g. risk-based domain testing, user-oriented regression testing).

Domain — *Try ranges and options*

1. Identify domains to test.
2. Analyze the limits and properties of each domain.
3. Identify combinations of domains to test.
4. Apply selection strategy:

   e.g., exhaustive, typical, boundaries, random, invalid

Stress — *Overwhelm the product*

1. Select test items and functions to stress.
2. Identify data and platform elements that relate to them.
3. Select or generate challenging data and platform configurations to test with:

   e.g., large or complex data structures, high loads, long test runs, many test cases, low memory conditions

Flow — *Do one thing after another*

1. Define test procedures or high level cases that incorporate multiple tests connected end-to-end.
2. Don't reset the system between tests.
3. Incorporate the element of time.
4. Combine with other techniques:

   e.g., user flow, stress flow, risk-based flow

User — *Strive for realism*

1. Identify categories of users.
2. Determine what each category of user will do, how they will do it, and what they value.
3. Get real user data, or bring real users in to test.
4. Otherwise, systematically simulate a user (be careful—it's easy to think you're like a user when you're not).

Regression — *Repeat testing after changes*

1. Identify what product elements changed.
2. Identify what elements could have been impacted by the changes.
3. Select what to test, such as recent bug fixes, past bug fixes, new code, sensitive code, or all code.

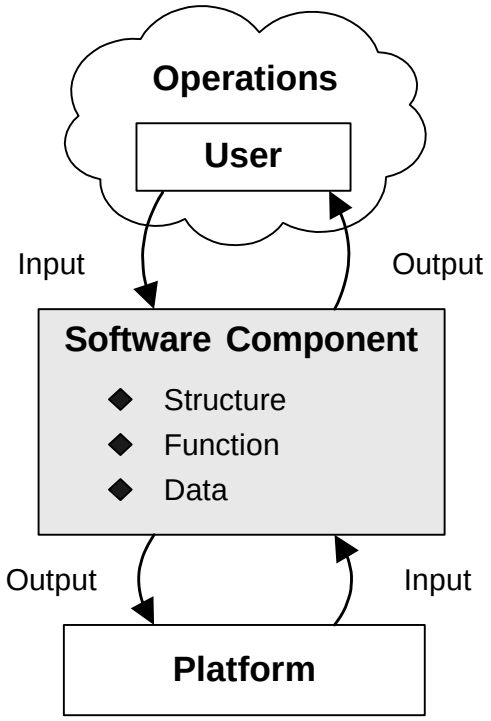
Risk — *Find big bugs first*

1. Analyze project factors, product elements, and quality criteria in order to identify sources of risk.
2. Focus testing in areas of highest potential risks.
3. Use test results to refine the risk analysis.
4. Be careful not to completely neglect low risk areas—your risk analysis might be wrong.

Claim — *Verify every claim*

1. Identify reference materials that include claims about the product (implicit or explicit).
2. Analyze individual claims, and clarify vague claims.
3. Verify each claim.
4. If you're testing from an explicit specification, expect it and the product to be brought into alignment.

Product Elements Model



Operations include the **user**, how the user interacts with the product, and anything else going on in the user's business, social, and physical environment that could affect the product.

All non-trivial **software components** consist of some **structure** (e.g. files, lines of code), they all perform some sort of **function**, and they all exchange **data** with some sort of user (which may be another progam) and some sort of platform.

The **platform**, broadly speaking, is anything the software depends upon in order to function. This includes the hardware and operating system, of course, but may also include other external programs or configuration parameters.

Product Elements

Ultimately a software product is an experience or solution provided to a customer. Software products have many dimensions. So, to test well, we must take care to examine a variety of those dimensions. Each category of elements, listed below, represents an important and unique aspect of a product. Testers who focus on only a few categories are likely to miss important bugs. This list is a generic starting point, but in any particular domain of technology you should be able to construct a more specific checklist of elements to examine.

Structures

Everything that comprises the physical product.

- **Code:** the code structures that comprise the product, from executables to individual routines.
- **Interfaces:** points of connection and communication between sub-systems.
- **Hardware:** any hardware component that is integral to the product.
- **Non-executable files:** any files other than programs, such as text files, sample data, help files, etc.
- **Alternate Media:** anything beyond software and hardware, such as paper documents, web links and content, packaging, license agreements, etc..

Functions

Everything that the product does.

- **User Interface:** any functions that mediate the exchange of data with the user.
- **System Interface:** any functions that exchange data with something other than the user, such as with other programs, hard disk, network, printer, etc.
- **Application:** any function that defines or distinguishes the product or fulfills core requirements.
- **Error Handling:** any functions that detect and recover from errors, including all error messages.
- **Interactions:** any interactions or interfaces between functions within the product.
- **Testability:** any functions provided to help test the product, such as diagnostics, log files, asserts, test menus, etc.

Data

Everything that the product processes.

- **Input:** any data that is processed by the product.
- **Output:** any data that results from processing by the product.
- **Preset:** any data that is supplied as part of the product, or otherwise built into it, such as prefabricated databases, default values, etc.
- **Persistent:** any data that is stored internally and expected to persist over multiple operations. This includes modes or states of the product, such as options settings, view modes, contents of documents, etc.
- **Temporal:** any relationship between data and time, such as the number of keystrokes per second, date stamps on files, or synchronization of distributed systems.
- **Invalid:** any data or state that should trigger an error handling function.

Platform

Everything on which the product depends.

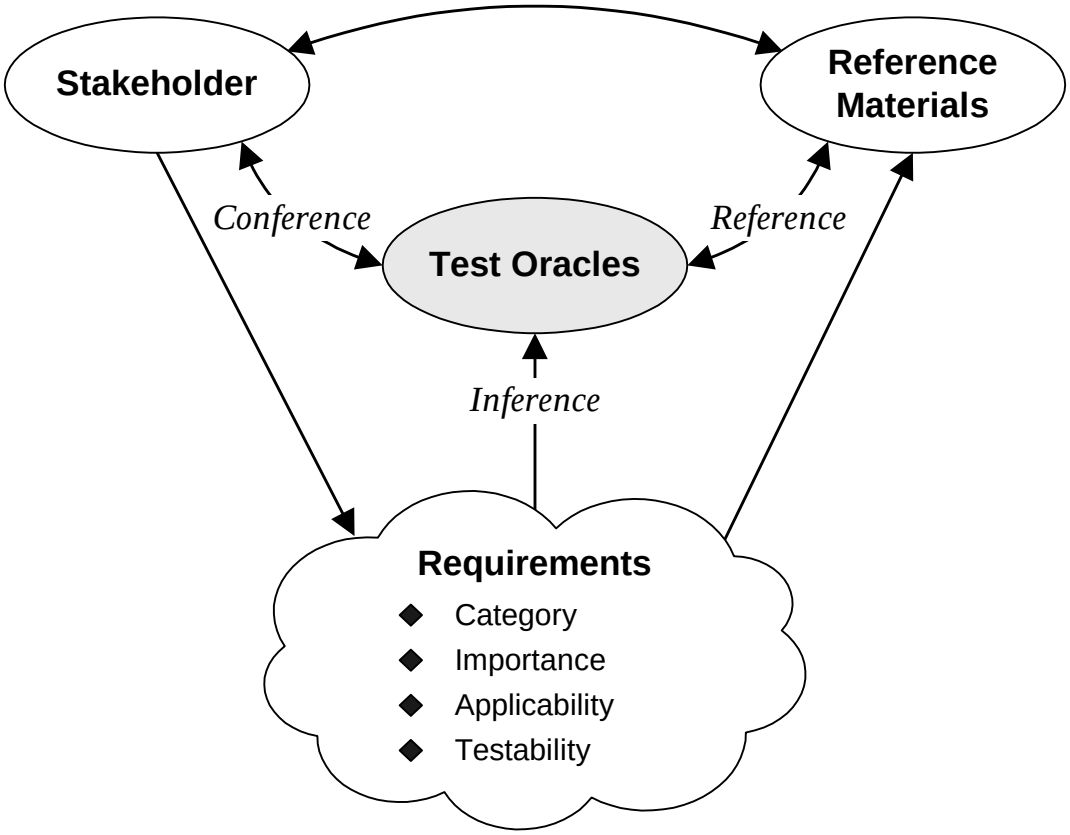
- **External Hardware:** hardware components and configurations that are not part of the shipping product, but are required (or optional) in order for the product to work. Includes CPU's, memory, keyboards, peripheral boards, etc.
- **External Software:** software components and configurations that are not a part of the shipping product, but are required (or optional) in order for the product to work. Includes operating systems, concurrently executing applications, drivers, fonts, etc.

Operations

How the product will be used.

- **Usage Profile:** the pattern of usage, over time, including patterns of data that the product will typically process in the field. This varies by user and type of user.
- **Environment:** the physical environment in which the product will be operated, including such elements as noise, light, and distractions.

Quality Criteria Model



Oracles are strategies for detecting product failure. An oracle may be an idea in the mind of a tester that allows him to recognize a bug when he sees one. A major task of testing is determining oracles that are in line with the true requirements of the product. We do that mainly through **conferencing** with the stakeholders, **referencing** authoritative materials, or by **inferring** requirements based on other things we know.

A **stakeholder** is anyone whose opinion matters with regard to the quality of the product. All requirements come, directly or indirectly, from one or more stakeholders. A software tester is an agent for the stakeholder throughout the test process.

Reference materials are any documents or artifacts that can be used as a source of information about requirements. If a stakeholder acknowledges a reference as an authoritative source, we call it an *explicit* reference. An *implicit* reference is any useful material that is not acknowledged. Often, the process of testing will cause the stakeholders to improve the reference materials. Testers may also prepare reference materials based on exploratory testing and conferring with stakeholders.

Requirements are the desires of the stakeholders. It takes a lot of time, effort, and skill to elicit anything like a complete set of requirements from the stakeholders. Even when that is done, requirements will continue to change as the stakeholders gain experience with the product, or the world around them itself changes. When identifying requirements, it can help to start with **categories** of possible requirements. For each requirement, also consider how **important** is it for that requirement to be satisfied? Where, when and to what is that requirement **applicable**? Is the requirement **testable** enough to know if the product satisfies it?

Quality Criteria Categories

When we say that a product is "high quality" we mean that it satisfies the quality criteria of its stakeholders to a "high" degree. We often have to test without knowing exactly what those criteria are, but we can use this list of criteria categories to brainstorm or otherwise ask pointed questions aimed at revealing what we need to know. For each category, determine if it is important to your project, then think how you would recognize if the product worked well or poorly in that regard. This list particularly benefits from customization. We recommend reviewing the ISO 9126 quality characteristics standard or the Encyclopedia of Software Engineering for other category ideas.

Operational Criteria

Capability ***Can it perform the required functions?***

Reliability ***Will it work well and resist failure in all required situations?***

- **Error handling:** the product resists failure in the case of errors, is graceful when it fails, and recovers readily.
- **Data Integrity:** the data in the system is protected from loss or corruption.
- **Security:** the product is protected from unauthorized use.
- **Safety:** the product will not fail in such a way as to harm life or property.

Usability ***How easy is it for a real user to use the product?***

- **Learnability:** the operation of the product can be rapidly mastered by the intended user.
- **Operability:** the product can be operated with minimum effort and fuss.

Performance ***How speedy and responsive is it?***

Installability ***How easily can it be installed onto its target platform?***

Compatibility ***How well does it work with external components & configurations?***

- **Application Compatibility:** the product works in conjunction with other software products.
- **Operating System Compatibility:** the product works with a particular operating system.
- **Hardware Compatibility:** the product works with particular hardware components and configurations.
- **Backward Compatibility:** the products works with earlier versions of itself.
- **Resource Usage:** the product doesn't unnecessarily hog memory, storage, or other system resources.

Development Criteria

Supportability ***How economical will it be to provide support to users of the product?***

Testability ***How effectively can the product be tested?***

Maintainability ***How economical is it to build, fix or enhance the product?***

Portability ***How economical will it be to port or reuse the technology elsewhere?***

Localizability ***How economical will it be to publish the product in another language?***

QW2000 Vendor Technical Paper 2V2

Mark L. Van Name
(ZD Labs)

Benchmarking the Future: Future Directions

BACK TO QW2000 PROGRAM

Key Points

- Learn how to use Ziff-Davis Publishing's i-Bench to test the new generations of Web appliances as they arrive
- Get the inside scoop on how WebBench's custom workloads can reveal the performance of Web-related, server-side technologies
- See how Winstone's "hot spot" testing orientation can help you distinguish among the flood of ever-faster processors and PCs

Presentation Abstract

The headlong rate of change of technology today presents those of us in the testing business with serious challenges: How can we test these new technologies? Measure their performance? Though we sometimes have to create new tools to meet these challenges, existing tools can often do the job. In this presentation you'll learn how to use the current generation of Ziff Davis Publishing's industry-standard benchmarks to cope with the technologies that are hitting you now and will be hitting you in the months ahead.

Many of us, for example, are trying to sort through the rush of new Internet-access devices. From Web-savvy refrigerators to Internet-linked cell phones, devices of all sorts claim to offer great online capabilities-but how can you determine what each device can really do? i-Bench may well be the answer. We'll discuss how to use it to measure the capabilities and performance of these Internet-access devices as they appear.

On the server side of the Web, technologies are also evolving rapidly. We all want to understand how our sites and servers will perform, but finding tools to apply the right workloads can be tricky. A large number of vendors offer powerful and expensive products to address this problem, and those products are important. You can also, however, get significant data about new technologies from Ziff Davis Publishing's freely available WebBench product by using it to create custom workloads. We'll give you the lowdown on building the workload you need for many types of Web-server tests.

The desktop PC isn't standing still, either. Though we're confident most people would agree that current PCs are fast enough for most business applications, people still waste a lot of time waiting on PCs. Ziff Davis Publishing's Business Winstone and Content Creation Winstone have evolved to address this problem by focusing on what we call "hot spots," computing tasks that leave you twiddling

your thumbs. Learn how you can use these tools to determine which of the ever-faster processor, RAM, and system technologies deliver the best performance and the best value.

About the Author

Mark L. Van Name is Vice President and General Manager of ZD Labs, the independent, for-hire testing service of the Ziff Davis Publishing Company. With more than a decade of work as a software developer and an M.S. in Computer Science, he brings extensive experience and a strong academic base to product testing.

The former Editorial Director of Ziff Davis' Windows Sources Magazine, he is also, with Bill Catchings, the author of over a thousand articles and has been a columnist for Windows Sources, PC Week, and Computer Shopper.

BACK TO QW2000 PROGRAM

QW2000 Paper 2T2

Mr. James Elder & Mr. Ricard Roma i Dalfo
(Microsoft)

Object Based Machine Automation ("OSIRIS Project")

BACK TO QW2000 PROGRAM

Key Points

- Discuss the problems / difficulties of automating configuration and scenario based testing
- Present the OSIRIS project as a possible strategy / technique for solving the above-mentioned problems - explain functionality, advantages, implementation, etc.
- Provide examples of how this automation strategy is being applied, scenarios being covered, how it could be expanded or adapted to fit other environments

Abstract...

The "OSIRIS Project" is the architecture of combining many existing tools and processes and exposing their functionality as automation objects so that they can be used in writing code to automate user scenarios or control machine configurations. These automation objects are implemented in a server-client model so that code running on a server can either control itself or one or more client machines all from a single code module. These objects include applications, machine images, snapshots, file/reg key functions as well as other system level control (reboots, auto logon, execute command, etc.). Many of these tools or processes already exist in some format, but by combining them in a common, automation object architecture, OSIRIS provides functionality and characteristics (extensibility, robustness, flexibility, etc.) unavailable in previous tools.

Author Bio...

James Elder Bio:
5-7/93 - Test Engineer Intern for United Technologies Carrier Corporation
5/95 - Graduated from the University of Arkansas with a BS degree in Computer Systems Engineering
6/95 - Hired as a Software Test Engineer in the Microsoft Excel test group
10/97 - Moved to "Release and Deployment" shared feature test team Currently working on upgrades testing for Office installations

Ricard Roma i Dal Bio:
1994-95 - Completed my 4th year in Computer Science at Trinity College of Dublin (Ireland -

EUROPE)
1996 - Final year project at iMAGIS, Institut National Polytechnique of Grenoble (France - EUROPE)
1991-96 - Graduated on Computer Science Engineering by Polytechnic University of Catalunya (Catalunya - EUROPE)
1996-98 - Worked as IT Manager at Bonmacor (services company in Barcelona, Catalunya, EUROPE)
1998- Joined Release and Deployment Office group at Microsoft (Redmond, USA).
Currently working on automation solutions for the group and other groups inside Office and Microsoft.

BACK TO QW2000 PROGRAM

Osiris radTools

Microsoft

OSIRIS Project

Object Based Machine Automation

James Elder
Ricard Roma i Dalfó

13th International Software / Internet Quality Week (QW2000)

Osiris radTools

Microsoft

Introduction

OSIRIS Project - automation strategy for scenario-based testing

Agenda

- Overview of scenario-based testing
- Project specification
- Implementation
- Practical applications

Osiris radTools

Microsoft

Scenario-based Testing

- Application vs. Scenario-based testing
  - App-based: code and internal functionality
  - Scenario-based: install, uninstall, upgrade, …
- Difficulties of scenario-based testing
  - Resource intensive
  - Large matrix of environment configurations
  - Complex user scenarios

Osiris radTools

Microsoft

Difficult to Automate

- Complex
  - Numerous configurations and variables
- Loss of control
  - Reboot, logoff, etc.
- Multiple machine scenarios
- Custom tools
  - Inadequate, impractical to use

Osiris radTools

Microsoft

Project Specification

- Object based
  - Familiar methods, properties
  - Modular
  - Consistent
- Usable in professional development environments
- Compatible

Osiris radTools

Microsoft

Project Specification, cont.

- Robust
- Client/Server structure
  - Multiple machine scenarios
  - Loss of control
  - Single code source
- Extensible

Osiris radTools

Microsoft

Object-based

- Objects
  - Methods
  - Properties
- Use current development environments
  - Extra support for VB and scripting

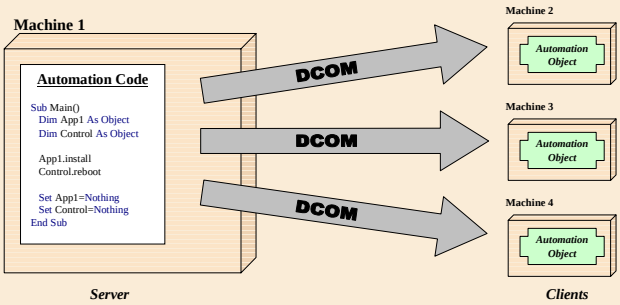
COM *Automation Objects*

Osiris radTools

Microsoft

Remote Control

- Multi-machine
- Loss of control
- Single code source



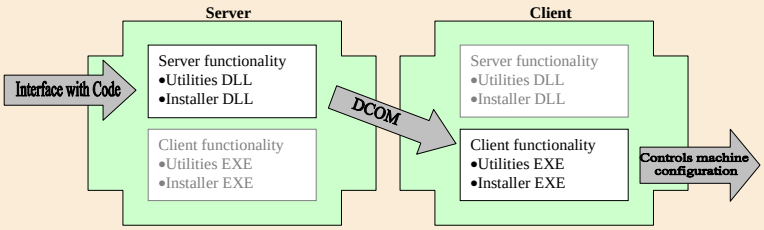
DCOM - *Distributed COM*

Osiris radTools

Microsoft

Server/Client Components

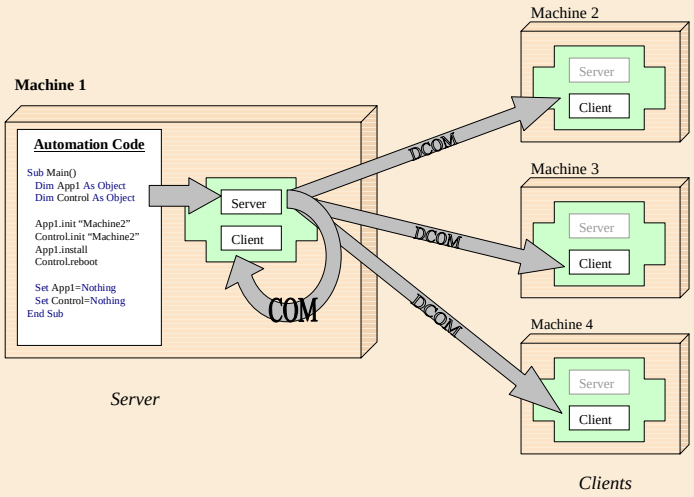
- DCOM too complex for automation writers to use effectively
- Proxy to client component
- Encapsulates object creation, remote communications, and object state recovery



Osiris radTools

Microsoft

Implementation Overview



Osiris radTools

Microsoft

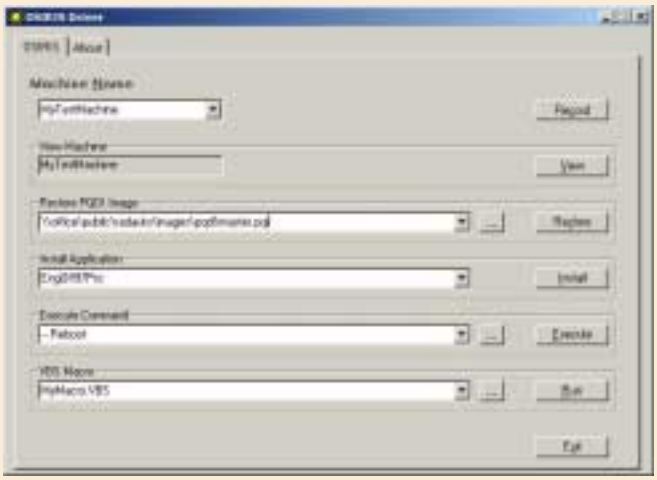
Practical Applications

- Scenario-based automation
  - Application install, uninstall, upgrade
  - Configuration variables: OS, language
  - Complex user scenarios
- Manual Tools
  - Visual Basic applet
  - Web page
- Additional applications

Osiris radTools

Microsoft

OSIRIS Driver



Osiris radTools

Microsoft

Web-based OSIRIS Driver

Osiris radTools

Microsoft

Additional Applications

- Lab Management
  - Multiple machine control
  - OS image restoration / creation
  - Utilize existing automation
- Any custom solution

Osiris radTools

Microsoft

Conclusions

- Scenario-based testing is difficult
- OSIRIS Project is a strategy for automation
- Implementation notes
- Practical applications to improve product quality

OSIRIS Project

Object Based Machine Automation

James Elder
Microsoft Software Test Engineer
jelder@microsoft.com

Ricard Roma i Dalfó
Microsoft Software Developer in Test
ricardrd@microsoft.com

International Quality Week, 2000

Abstract

Automation is an important part of the testing process and can greatly improve testing efficiency and product quality if used correctly. This paper does not discuss testing theory or methodologies, but rather a specific automation strategy for automating both simple and complex user scenarios by defining the different components as code objects. This strategy, which we call the "OSIRIS Project," is based on current industry technology, such as COM[1], and allows automation writers to control multiple machines, application installs, reboots, complex machine interactions, and so on, from a single code source.

This paper first defines and gives a brief overview of "scenario-based" testing, then explains the conceptual and technical implementation of an automation strategy using "objects" and finally discusses some of its practical applications for improving software quality. This paper reflects the authors' perspective gained from experience testing Office applications at Microsoft.

1.0 Introduction overview of Scenario-based testing

Testing the code and user interfaces of a software application is difficult enough by itself, but it is even more difficult and complex when you add the testing of that application's installation and its interactions with the many different operating systems, other related software and even previous versions of the same application. To help us clarify and prioritize our testing focus, we divided this testing into two parts: "application-based" testing which focuses on the application code and internal functionality, and "scenario-based" testing which focuses on how the applications are installed and/or how they interact with other software. Typically application-based testing receives more resources and attention during application development, but as the complexity matrix of our applications and the number of environment configurations increases, the need for better scenario-based tools, better testing processes and better overall understanding and management becomes apparent.

1.1 Difficulties / history of scenario-based automation

In trying to automate scenario-based testing for Office applications, it was difficult to find a practical tool or method for managing the different variables of the operating system configuration, the install and uninstall of applications, the many different types of user interactions and the scenarios with loss of control such as reboots, different user logon, etc. First, we tried different industry tools and different programming methods such as batch files, PERL scripts, Visual Basic (VB), "quiet" application installations, "send keys," and so on. These strategies were sufficient for particular pieces of the problem, but they did not provide a complete solution. Next, we tried creating our own scripting language and development environment to accommodate scenario-based automation. This later strategy seemed like a practical solution and did provide us with a modular, extensible approach to the problem, but we soon realized that we were spending as many resources on the testing and maintenance of the scripting environment as we were on using it for application automation. This strategy also had a steep learning curve and was only understood and used by a small number of testers.

Through our experience with scenario-based testing and our previous attempts to automate these scenarios, we were able to clearly define the characteristics and functionality that would be needed to solve this problem. Our solution needed to be:

- Robust - to handle the diverse environment configs, reboots, timeouts, etc.
- Modular - to have reusable "objects" and reusable functionality
- Extensible - to easily add additional functionality
- Compatible - to utilize existing investments in tools and testing methods
- Programmable - to provide functionality of professional development environments (looping, variables, conditionals, etc.)

We also realized that the different components and variables used in scenario-based testing could be considered "objects" with properties and methods just like familiar objects in any object-based development environment such as VB or Visual C++ (VC++). For example, if you consider the application Excel 97 as an object, then it would have respective methods such as *install*, *uninstall* and properties such as *source*/*destination* directory, *username*, *product identification key*, etc. A user object for example could be defined with methods *logon*, *logoff* and properties *administrator*, *poweruser*, etc. Consider the implications if your scenario-based automation code could be as simple as the following sample pseudo-code:

```
Module InstallTest
   Declare variable MyApp as Application = Excel97
   Declare variable MyUser as User = administrator
   MyUser.logon
   MyApp.install
   Call VerifyInstallation subroutine
End Module InstallTest
```

Note: Although our actual implementation does not parallel this object structure exactly, the concepts and characteristics form the basis for the OSIRIS Project. Also note that our objects and functionality are unique for our testing needs, but the overall strategy can be applied to almost any software application.

1.2 OSIRIS Project Specification

Our first task was to write a specification of the different objects and functionality needed to automate our scenario-based testing and to elaborate on required characteristics of the automation strategy.

Required characteristics

Utilize professional development environments

A major problem with current tools for scenario-based automation is that they often require the automation writer to either learn a new scripting language or to learn a new complex user interface. The tools and processes we tried were often limited in flexibility, were not extensible and were not compatible with our existing tools. In some cases, simple programming functionality like looping, variables, or conditionals was not available. We needed a way to write automation using professional development tools, such as VB, VC++, so that we could utilize the development environment's features and would have almost no learning curve if already familiar with the language.

Client-Server based to handle loss of control

Many scenario-based tests include reboots or multi-user logon, both of which usually cause a loss of control for automated tests. The automation strategy must be client-server based so that a single code source executing on a server can lose and regain control of a client machine without disrupting the programmatic flow of the automation code.

Client-Server based for multiple machines

The automation strategy also must be client-server based so that it can handle multiple machine scenario-based tests such as roaming user, administrative software deployment, web / database / mail server interactions, and so on.

Robust

The strategy must be reliable, maintain its own error handling and work across many different environment configurations.

Modular and Consistent

The many different components must be clearly defined as stand-alone objects so that they can be reused throughout different code modules, in any order, etc. with expected and loggable results. Objects also must be implemented in a consistent manner so that familiarity of one object can transfer to another as well as provide a template for future objects.

Compatible and Extensible

The strategy must be compatible with our custom tools and processes while still allowing for future functionality additions.

Required Objects

Applications

A set of objects to install different versions, languages, SKUs of Microsoft Office applications such as Office 95, German Office 97, Internet Explorer, Office 2000, etc.

Logging

Object used to manage log files and pass / fail test results

Execute

Object for execution of executables, batch files or any other command that is executable through the OS command-line

FileInfo

Object used to query file information (times, language, version, size, etc.), to copy / move / delete / rename files and directories and to set NTFS file / directory security permissions

Images

Object to create and restore images of a particular hard drive (this allows automation to control restoring different operating systems such as Win95, NT4, German Win98, or any previously created configuration)

ProcessInfo

Object used to query for or to terminate running processes

Registry

Object used for creating / setting / deleting keys and values within the system registry or within INI files

Shutdown

Object to shut down / reboot / set autologon for a client machine

Snapshot

Object used to take snapshots of a machines files / registry; also allows taking the difference between two snapshots

SystemInfo

Object used to query information about a machine's configuration: path locations (system path, user specific paths, programs paths, temp path, etc.), system language, machine name, logged on username, OS type, environment variables, etc.

TimeDate

Object used to query for and to modify system date and time

ToolsManagement

Object used to verify that the automation tools are installed and to update the tools to the latest version

UserGroups

Object to create / delete NT users and groups

Note: There are an infinite number of "objects" that could be implemented for use in this strategy. The previous abbreviated list provides examples of the types of objects we knew we needed to implement.

With this set of objects a tester can automate almost any scenario by just writing simple code to setup a machine and even launch other automation.

2.0 Implementation details

Once the conceptual architecture of the OSIRIS Project was clearly defined, we needed to find a technical solution to solve these problems. The following text reviews the problem set, solutions we found and step-by-step implementation notes.

Our strategy needed to implement the requirements from the specification:

- Robust - to handle diverse environment configurations, timeouts, and so on.
- Client/Server - to handle reboots and multiple machine scenarios
- Modular - to have reusable objects and reusable functionality
- Extensible - to easily add additional functionality
- Compatible - to utilize existing investments in tools and testing methods
- Programmable - to provide functionality of professional development environments

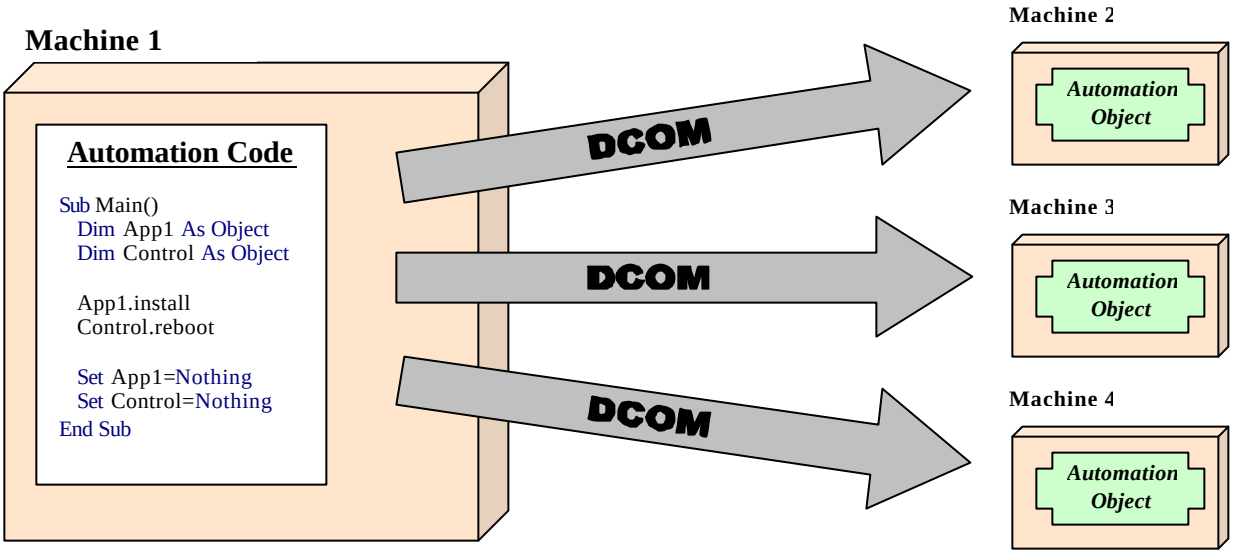
2.1 Automation Objects

The first goal was to create a set of objects that we could use across different programming environments and that would define the functionality we needed. To solve this, we moved toward a Component-programming paradigm. In short, we chose COM as the framework on which we defined our objects. COM is based on components that can be instantiated from any programming language (VB, VC++, Delphi, scripting languages like JavaScript, VBScript, etc.). This allowed us to encapsulate all the functionality that we needed in distinctive components that the user could use however they needed. To be able to support scripting languages and also make coding easier for VB automation writers (our main target audience), we developed these COM objects as *Automation objects*[2].

The following is an example of an application *Automation object* EngOff97Pro (**Eng**lish **Off**ice **97 Pro**fessional). This object provides properties and methods to install or uninstall, specify source or destination locations, userName, etc.

2.2 Automation Objects + DCOM

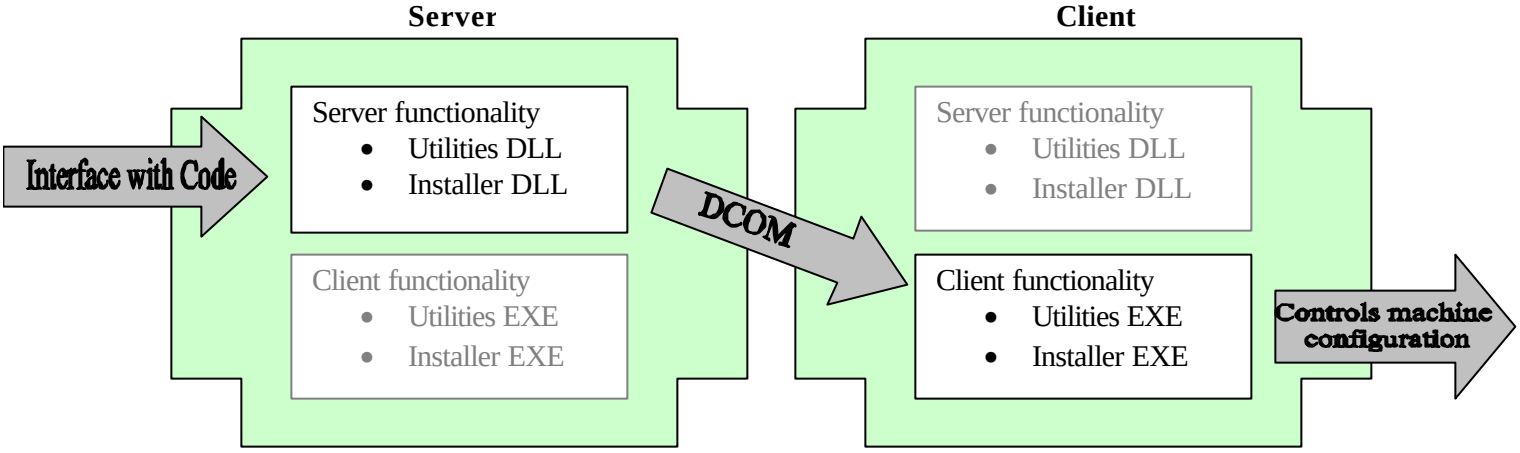
The next goal was to implement remote control of a client machine so that we could manage reboots, logon as a different user, multiple machine scenarios, etc. all from a single code source. The solution was to use a server machine that controlled the flow of the entire scenario as well as the communication with each of the client machines. COM technology provided us with a remote control mechanism called DCOM[3], or "Distributed COM." DCOM allowed us to remotely instantiate the *Automation objects* and work with them as if they where local to the server machine. DCOM allowed us to set permissions on the objects so that we could control which objects to expose as well as who could use those objects.



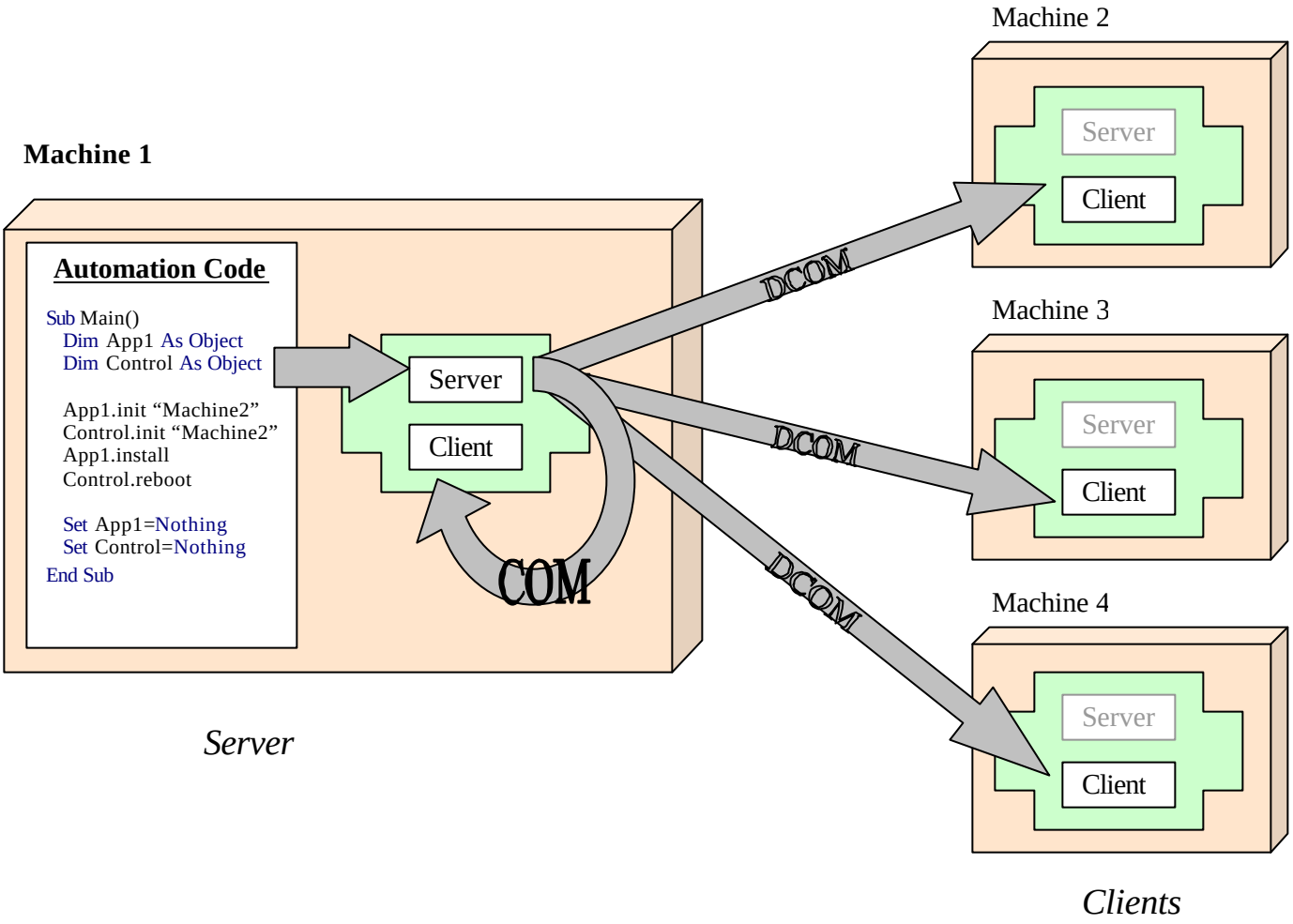
2.3 Automation Objects + DCOM + Server/Client considerations

A new problem we quickly recognized was the complexity of the DCOM remote control mechanism, and although DCOM was not much more complicated than COM, there were many considerations that were not present in the client-object-only approach. The first complication we had to address was that the instances of the objects and the objects' properties would be lost when the control code rebooted the client machine. Therefore, the next attempt by the control code to access a particular object would generate an error because that object no longer existed. The solution was to add special handling to recreate the objects after the reboot and to restore each objects properties to their pre-boot state. We also found that the automation writer needed to know whether they were accessing an object remotely or locally because they would need to use different instantiation methods for both, thus making usability more complicated.

To solve these new problems, we decided to create the objects as two components: a client component and a server component. We had already implemented the client component, which provided the functionality we needed to control a client machine, but we added a similar server component, which acts as a proxy to the remote client objects. This new server component exists on the server machine and is a set of *Automation objects* that manage the remote control specific actions. These server-side *Automation objects* have a one-to-one mapping of the properties and methods to the original objects. The automation writer simply creates these server side objects and the *Automation objects* themselves internally determine whether to create the objects remotely or locally. To implement this solution, we added a method, *Init*, to all of the server objects thus allowing the automation writer to specify which machine to control by simply using this method. The server object maintains the object information to recreate and recover the state of the client object after a loss of control, as in a reboot. We also decided to install both the server and the client piece in all of our installations so that any machine could act as a server or a client, or even a client of itself (only in those cases without reboots or other loss of control).



Finally we had our implementation model with a pair of *Automation objects* (the server and the client component) on all machines. As you can see in the diagram the user always instantiates the local *Automation object* (the server component) and this object determines whether to create the client component either locally or remotely depending on the Init function parameter that specified which machine to control.



Because all of the client objects relate to machine configuration (mainly Win32 API calls), we implemented them in C++ using Microsoft ATL libraries. We also created a process that uses an *Automation object* to programmatically generate its server component. This process allowed us to quickly expand our object library. The server component code was written in Visual Basic.

Note: The OSIRIS object libraries were implemented in two parts: utilities library and applications library. The utilities library includes functionality-related Automation objects (for example, reboots, imaging, file/registry key manipulation, etc.) and the application library includes Automation objects specifically for installing applications (Office 97, German Office 2000, Internet Explorer, etc.). These two areas could easily be written as one library or be divided into many different libraries. It was divided into these two parts simply to balance development resources. Also, although most of our current application Automation objects are related to the Office products, these same types of objects can be created for any application.

3.0 Practical applications of the OSIRIS Project

After we implemented a few of the core objects and functionality of the OSIRIS project, we quickly realized that this automation strategy was not only going to work well for its original goal (scenario-based automation), but would also provide an easy way to build helpful, manual tools and to help create lab management systems. We also noticed that our approach and perspective to automating scenario-based testing changed from trying to work around tool and process limitations to focusing on automating new scenarios and creating new solutions. The following examples describe how we are currently utilizing this strategy to automate scenario-based testing for Microsoft Office applications and how to create new tools. Keep in mind, however, that you can apply this strategy and its techniques to any software application and in many diverse ways.

3.1 Automation of scenario-based testing

The original reason these tools were created was to automate setup, upgrades and other complex user scenarios such as "roaming user," software deployment, policy administration, etc. We have been using OSIRIS for a while now and this automation strategy continues to provide us with a powerful, easy to use solution for automating scenario-based testing. This strategy has dramatically increased our coverage of the broad matrix of possible, real world configurations; it has increased our manual testing efficiency (by using half automated / half manual type testing); and it has increased our confidence in quality as our applications change during the development process.

The following is a simplified example of real OSIRIS type automation code. The code is for an Office 97 Professional upgrade to Office 2000 Premium CD1.

```
Private Sub Main()
    'DECLARE VARIABLES
    Dim TestMachine As String
    Dim objApp1 As radInstaller.EngOff97Pro
    Dim objApp2 As radInstaller.EngOff2000PrmCD1
    Dim objReboot As RadUtilitiesSrv.ShutDown

    TestMachine = "TestMach1"

    'CREATE APP AND REBOOT OBJECTS
    Set objApp1 = CreateObject("radInstaller.EngOff97Pro")
    Set objApp2 = CreateObject("radInstaller.EngOff2000PrmCD1")
    Set objReboot = CreateObject("RadUtilitiesSrv.ShutDown")

    'INITIALIZE OBJECTS ON CLIENT MACHINE
    objApp1.Init TestMachine
    objApp2.Init TestMachine
    objReboot.Init TestMachine

    'SET APP1 USERNAME PROPERTY AND INSTALL APP1
    objApp1.userName = "Office UserName"
    objApp1.Install

    'REBOOT
    objReboot.Reboot

    'SET APP2 DESTINATION DIRECTORY AND INSTALL APP2
```

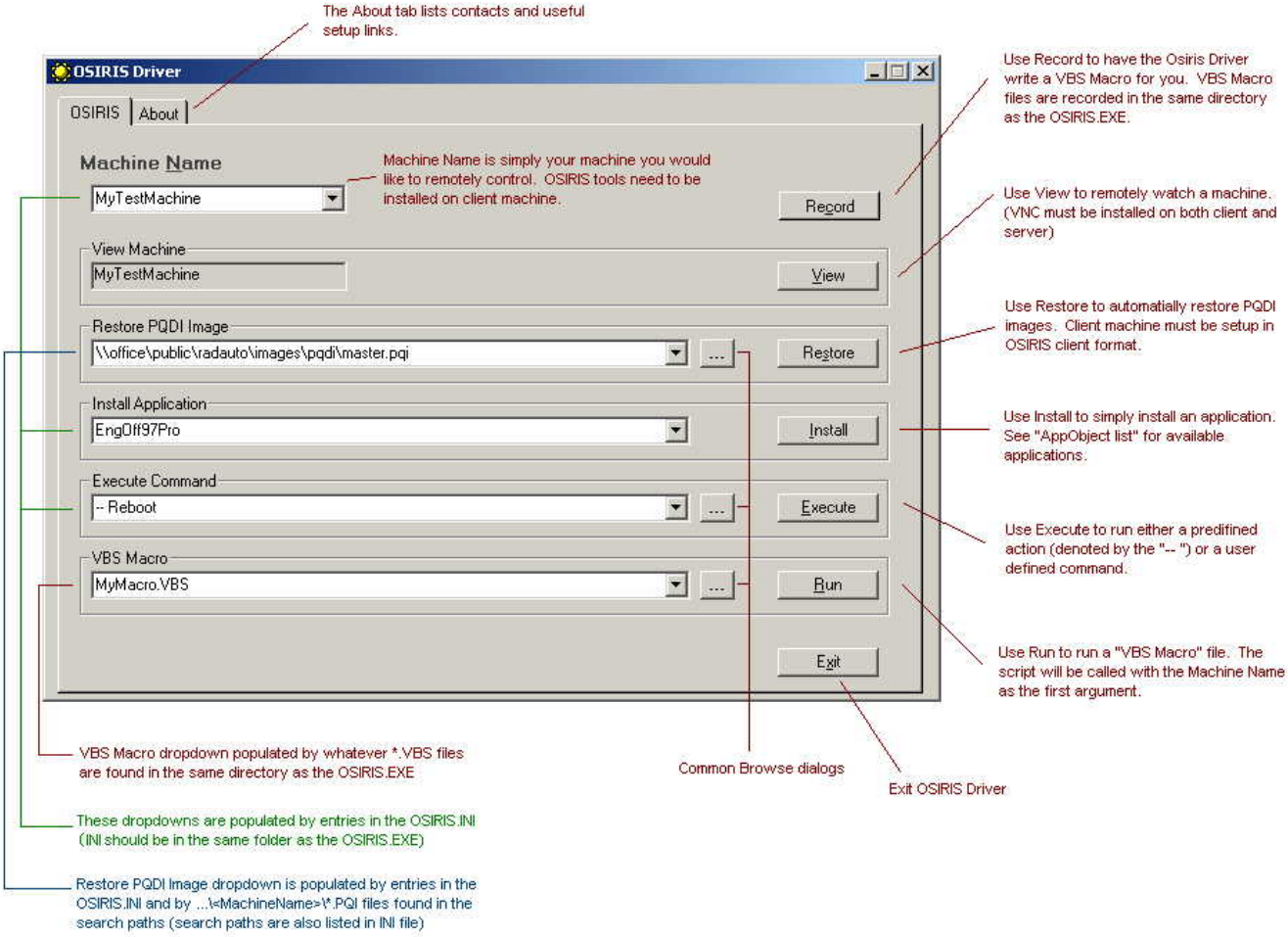
```
    objApp2.destination = "C:\Program Files\Microsoft Office 2000"
    objApp2.Install

    'CLEAR OBJECTS
    Set objApp1 = Nothing
    Set objApp2 = Nothing
    Set objReboot = Nothing
End Sub
```

3.2 Improve Manual Testing

Manually executing scenario-based tests is very resource intensive because even simple scenarios take a long time to run. Scenario-based testing also requires many redundant processes such as restoring a test machine to a clean state, pre-configuring a specific environment for testing new builds of the application, taking file and registry snapshots, etc. – all of which are important steps, but are not always that interesting for finding bugs.

VB Applet OSIRIS driver

Below is a screenshot of a tool we created called the OSIRIS Driver.

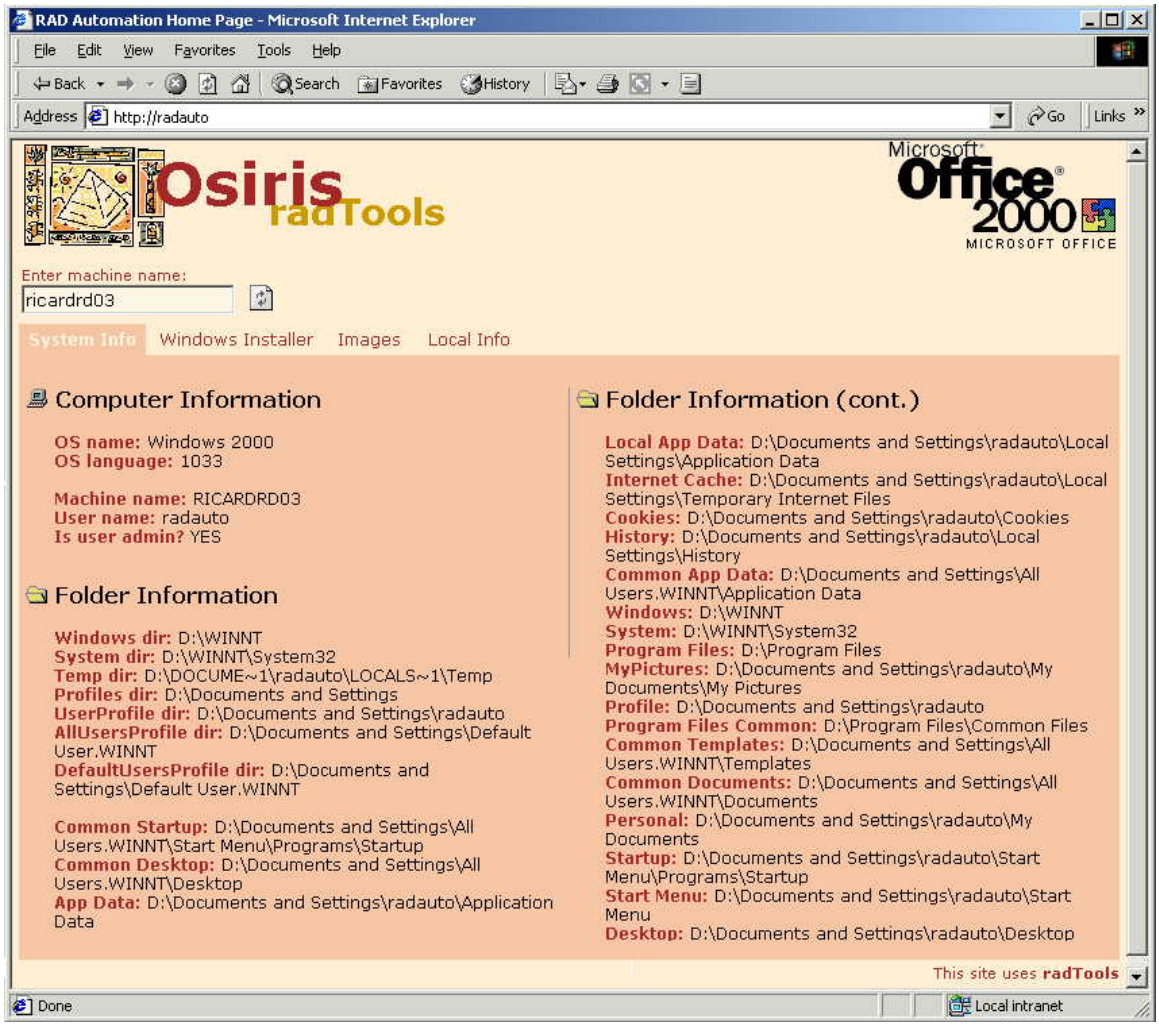


This tool is a simple VB applet, built on the OSIRIS objects, which provides the test engineer with a nice, simple UI for controlling a test machine. One of the key features of the OSIRIS Driver is that by utilizing our "image" *Automation object*, a user can restore or create an image of an operating system configuration by simply clicking a couple buttons (the actual creating and restoring of the disk image is performed by an imaging tool such as Ghost, Power Quest Disk Imaging, etc.[4]). Another key feature of the OSIRIS Driver is that it can execute VBS code (which can contain OSIRIS objects) on a remote test machine. Using VBS text files allowed us to apply the concept of macros, which can be recorded by simply echoing text to a file.

Web Based OSIRIS driver

Since these objects can be used in any code language that supports *Automation objects*, such as VBScript or JavaScript, we have been using them on internal web sites to control test machines. Hosting these objects on a web site can reduce the number of required machines by consolidating the server side of the *Automation objects* onto the web server.

Below is an example of a web page that displays directory and system information about a specified machine as well as what applications are installed, what images are available for that machine, etc.



3.3 Lab Configuration Control

Although we already have a powerful automation lab control system for our applications, we have found that by using the OSIRIS objects combined with a VB Applet or web page front end that it would be fairly easy to create a utility application to control a multi-machine lab environment. Because the OSIRIS system is client-server based and uses an "image" object, you can programmatically restore different operating system images, run existing (or new) test automation, collect log and/or snapshot files to a file server, restore the same machines to different operating system images and continue running automation, etc. all from a single code source with no user interaction whatsoever. This automation strategy simply provides intuitive building blocks for creating custom solutions large or small.

3.4 Additional Advantages

Another big advantage of using this automation strategy is that it is extremely flexible. We can create as many *Automation objects* as we need, and we can expose whatever methods and properties we need to customize relating to those objects. Writing automation for any of the different scenario-based tests is now limited by what we want to automate not by the tools that we use to automate. This strategy has helped us in developing certain manual front-end tools, but it does not prevent us from creating new solutions, better tools, or better automation.

Conclusion

Scenario-based testing is a difficult part of the software development process because it involves many different variables with complex interactions and it is typically more resource-intensive than traditional application-based testing. By taking the different components of scenario-based testing and implementing them as objects with methods and properties, a tester, or other automation writer, can automate scenario-based testing without learning custom tools or processes, by utilizing professional development environments, and by managing complex user scenarios all from a single, robust code source. The OSIRIS Project automation strategy was developed to solve the problems that limited our automation of scenario-based testing for Office applications. There are many different strategies for solving these kinds of problems and every application development process has unique needs and obstacles and thus requires a unique strategy. We have learned that the OSIRIS Project not only provides a generic solution for scenario-based automation, but it also can be utilized for creating tools and for addressing other machine management needs.

Appendix

[1]COM - Component Object Model

Specification that defines Microsoft Component architecture. COM components consist of executable code distributed either as Win32 DLLs or EXEs. A component behaves as an object but it is language independent.
http://www.microsoft.com/com/

[2]Automation objects

Class of COM object that uses a limited number of types and can be instantiated from scripting languages.

[3]DCOM - Distributed COM

Provides a remote control mechanism for creating and using COM components on remote computers.

[4]Power Quest Drive Image, Ghost

Imaging tools used to create and restore hard drive partition images.
http://www.ghost.com
http://www.powerquest.com

Additional References

MSDN – http://msdn.microsoft.com/

VNC - Remote display system used for viewing remote machines.
http://www.uk.research.att.com/vnc/

COM, Automation objects:
Inside COM - Dale Rogerson - Microsoft Press

DCOM:
Inside Distributed COM - Guy Eddon / Henry Eddon - Microsoft Press

QW2000 Paper 2A2

Mr. Juichi Takahashi
(Florida Institute of Technology)

Is Special Software Testing Necessary Before Releasing Products to an International Market?

BACK TO QW2000 PROGRAM

Key Points

- Present methods of developing international software
- Present methods of testing international software.
- Present methods of organizing international developing and test team

Presentation Abstract

Today, software developed by North American companies is sold all over the world. These companies also already have the basic knowledge of how to develop international software, by accommodating the fact that there is a difference in European and American date formats,

Today, software developed by North American companies is sold all over the world. These companies also already have the basic knowledge of how to develop international software, by accommodating the fact that there is a difference in European and American date formats,

currency symbols, and so on. Almost all basic knowledge of global software development can be understood by reading books from Lunde and Talor. Yet, we still run into trouble (or bugs) in Asian or European environments that isn't found in the North American market. Fortunately or unfortunately (depending on how you look at it), companies, which make the effort to develop for the international market, have acquired a great volume of sales through that international market. For instance, Microsoft gets around 50% of their sales from products sold in the international market.

Through my 6 years working for international software testing, I have had plenty of experience in software development for the international market, mostly obtained by trial and error, and feedback from unsuccessful projects. I am writing this paper to help others avoid making some of the same mistakes I made.

Outline

A. Is Special Software Testing Necessary before Releasing Products to an International Market?

B. Differences between English software and Globalized Software
   1. Quality Requirements
   2. Cultural, Geographical
C. Typical Issues
   1. Double byte and Single Byte
   2. Unicode (ISO 10646)
   3. Keyboard
D. International-Specific Software Testing
   1. Library, OS, Component
   2. Test Pass
   3. Double Byte and Extended Byte Checking
   4. Domain Test
   5. Keyboard
   6. Font checking
   7. Compatibility with local hardware
E. Testing Process for International Software
   1. Organize an International Test Team
   2. Project Control
   3. Cost Estimation
   4. Scheduling

About the Author

Juichi Takahashi is master degree candidate of Software Engineering at the Florida Institute of Technology. He worked at Microsoft in both the U.S.A. and in Japan for 8 years as Software Test Lead. He contributed to testing Far East products during his Microsoft days. It follows that his interests include software testing and international software development.

Is Special Software Testing Necessary Before Releasing Products to an International Market?

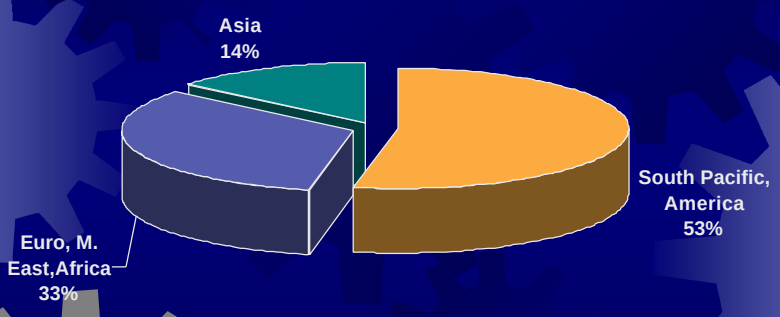
Juichi Takahashi
Florida Institute of Technology

- This slide is on:
  http://www.fit.edu/~jtakahas/qw
- My e-mail address is:
  juichi@IEEE.org

Who am I?

- I worked at Microsoft in both the U.S.A. and Japan for 8 years as Software Test Lead.

Importance of international product



Microsoft Revenue Chart in 1998

Issue, solution, and Testing

Unicode Benefits

- Can write a program in multiple languages program
- Easy testing
- Easy Maintenance

Test Pass

- We must finish the international part test before releasing international products.
- We must run full test cases for international products, which are designed for North American version too.

International Automated Testing

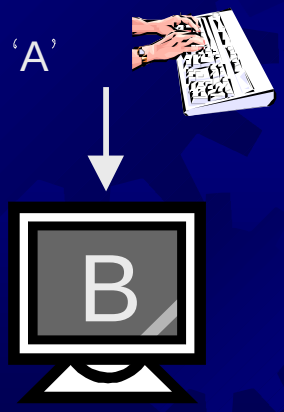
```
'check if there is the button
hwndResult = WButtonFind(…cOptButtonName)
if hwneResult=NULL then
  Fail "**Option Open::Aborting test***"
endif
```

```
cOptButtonName
 ="&Options  >>"
```

Keyboard Testing

Do you believe?

Type 'A'



Cultural and Geographical Testing

- Is Taiwan a country?

Example of Geographical Issue

Localized Side Effect

- Definition

Localized side effect is side effect from changing the codes to fix bugs in certain language.

Localized Side Effect Cont.

German Version

Japanese Version

How to avoid localized side effect bug?

- Do regression Test within International Products
- Keep communicating among international testing teams

Scheduling

- Mayers, who wrote *The Art of Software Testing*, said:

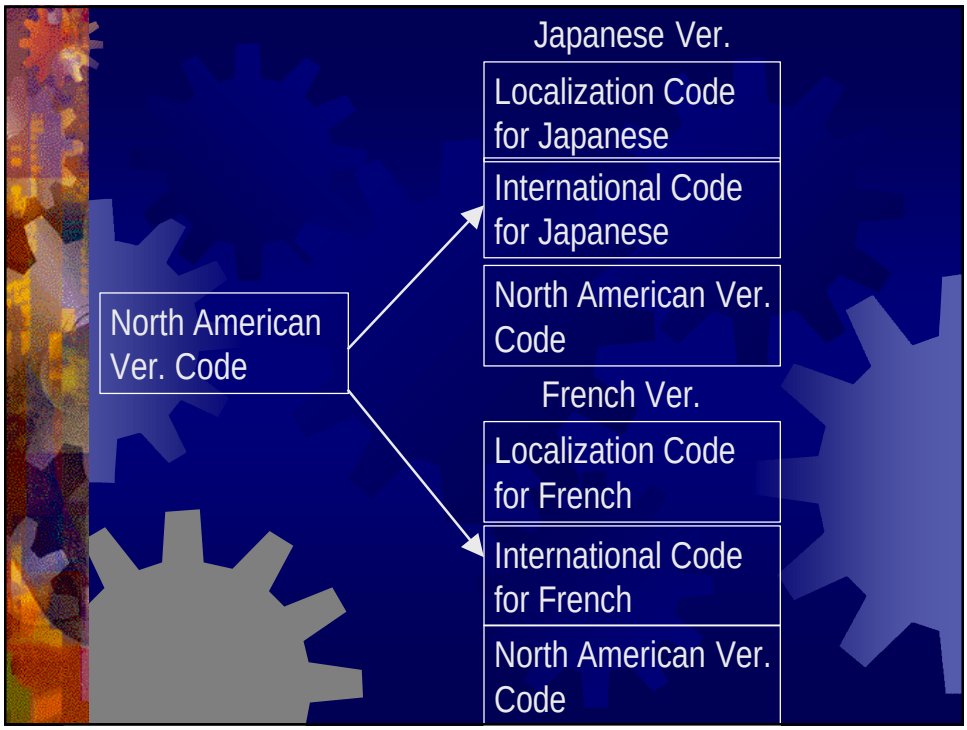
It is generally recognized that the earlier that errors are found the lower are the costs of correcting the error.

Scheduling Cont.

- Too early testing cost much for international products

Rafi Definition for Intl. Developing Process

1. Base product engineering
2. Internationalization
3. Localization



Testing Cost depends on if

- Target product handles double byte system.
- Target product handles high ANSI characters
- Target product handles 0x5c character set
- Team has help from local engineers.
- Target product is written in Unicode
- Target product program has any hard code.

- Target product is shipped simultaneously with North American version.

Conclusion

- Special software testing is necessary before releasing products to an international market.

Is Special Software Testing Necessary Before Releasing Products to an International Market?

Juichi Takahashi
Florida Institute of Technology
juichi@IEEE.org

Abstract

In this couple of decades, the international software market grows at a fast rate. North American based software is released to the world more often and get more than ever before. It is time that we need to know more about quality, the testing method, and the software development process to release international products.

I. Introduction

Today, software developed by North American companies is sold all over the world. These companies also already have the basic knowledge of how to develop international software, by accommodating the fact that there is a difference in European and American date formats, currency symbols, and so on. Almost all basic knowledge of global software development can be understood by reading books from Lunde [1] and Talor [2]. Yet, we still run into trouble (or bugs) in Asian or European environments that isn't found in the North American market. Fortunately or unfortunately (depending on how to look at it), companies, which make the effort to develop for the international market, have acquired a great volume of sales through that international market. For instance, Microsoft, which I worked at, gets around 50% of their sales from products sold in the international market. Thus, I have had plenty of experience in software development for the international market, mostly obtained by trial and error, and feedback from unsuccessful projects. I am writing this paper to help others avoid making some of the same mistakes we made at Microsoft.

Through the end of the twentieth century, most of the important software that exists (namely operating systems, compilers, or office suites) was developed by North American software companies. They often developed their software English version first, and then developed the other language versions in sequence. Coming into the twenty-first century, I assume that there will be reason to develop international software in a similar fashion. Therefore in this paper, I will make a point to elaborate upon the method of developing an English version first, and then developing an international version. This sequential development style is sort of an irregular style compared to the simultaneous development style of developing both the English and international versions at that same time. In the case of developing software with such an irregular development style, testers or quality assurance engineers must put much more effort and time into testing software developed with the sequential style than that developed using the simultaneous method. Because almost all project team leaders wish to have no major problems when they release their products to the international market, they naturally hope they won't need to make any code modifications to adjust only for the international market. Also, project managers try to cut the cost of international product development as much as they can. Consequently even on large projects, a tester is often

only working for international versions of products for a few months to make sure there are no major product quality issues. The project managers, of course, will assign developers and international project managers after testers start finding issues. Yet, they will not assign anyone beyond a tester to that part of the project until he or she finds issues that may be a bug or cultural issue. In such a case, a tester appears to be the only person responsible for the international project. Therefore, in this paper I attempt to include all related items, such as cultural issues, coding issues, and quality issues, that are to be checked in order to release high quality international software products. Moreover, this paper includes discussion of cost and schedule, since they can contribute to the difficulty of international software development.

Finally, this paper, which is aimed at a specific language, was written in general in as best a way I can. However, many parts of the examples are Japanese specific, for the reason that the Japanese market is the best next to the United States in sales according to many North American large software companies. Japanese examples are an excellent to use in international software discussion because the Japanese language system is one of the most complicated ones to be handled on a computer.

II. Is Special Software Testing Necessary before Releasing Products to an International Market?

Yes. For instance, while I worked in United States, one of the developers asked me if special testing was necessary before releasing products for an international market. He continually said that our products had no user interface, including cultural things. In addition, our products were based on Unicode. But I said "yes" again. We must test them before releasing. Otherwise we loose our customers because nobody knows if our product runs on localized operating systems. The worst case is that we must stop selling the product when there are critical bugs in the product. This is not a special case.

In this paper, I will explain one by one why North American companies need to test products before releasing for an international market and how to do it.

A. Importance of International product other than Northwest products.

Seeing the Fig. 1, one of the largest software companies, Microsoft, has around 50% profits from international software. Now, most of all large North American software company have released international product. The international market is more competitive than in the last decade. Therefore, if they release products which have low quality, they may damage their half of the profit.

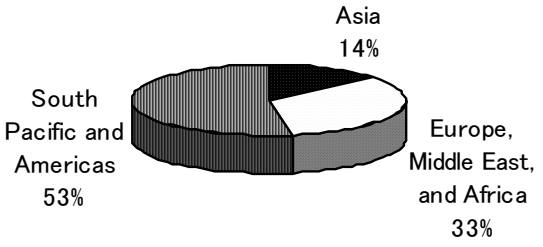


Fig. 1
Microsoft Revenue Chart in 1998

III. Differences between English and International Software Quality

A. Quality Requirements

One of Microsoft executives announced the delay of Windows 2000 shipment. He said that

users could easily forget the delay of shipment date, but never forget products which are of low quality. For instance, most of the Japanese still do not buy American car because they think the quality of American cars are much lower than that of others. As far as I know, the quality of American cars has been improved. Yet Japanese still remember that American cars were of poor quality in the past. In addition, American cars were not localized. In the Japanese traffic system, drivers drive left lane, so cars have right side steering. However, American automobile companies tried to sell the same American-style cars in Japan. Therefore, Japanese still believe that American cars are inconvenient for driving and of poor quality even though they have changed these things. The same thing might happen if North American computer companies keep shipping low quality products and do not consider the situation in another countries. Without changing code and testing, North American software companies may sell North American software in an international market, but it is risky, such as was automobile development.

We need to know the different user preferences between North American people and people in other countries to sell software worldwide.

IV. Issue, Solution, and Testing

A. Double byte and Single Byte

Before starting to develop international products, we must carefully choose the way to develop international software. Especially character set is the most important part. In case the products are Internet related, we may choose the Unicode as one of the development styles. On the other hand, in case that the products are for the government, we cannot use Unicode system but use the local code system. For international (not using Unicode), code should be in like

Fig. 3

```
for(char *p=szStr; *p; p++)
  if(*p='\\'){
     }
```

Fig. 2
Sample Code for ANSI

```
for(char *p=szStr; *p; p=AnsiNext(p))
  if(*p == '\\'{
     }
```

Fig. 3
Sample Code for International

Input Domain Test Strategy

Since there are complicated character sets, such as 7 bit, 8 bit, 2 byte, and 3 byte, input domain test is time-consuming but important. First, it is useful that we think how many types of characters there are in the world because Americans use only a type (1 byte) of character set in general.

- 1 byte base : ANSI, a part of Japanese.
- 2 byte : Japanese, Korean, Chinese which are for Chinese character set, and Unicode.
- 3 byte : EUC(Extended UNIX code)

International products are expected to correctly handle these three kinds of character types at maximum.

If Unicode character set is used for the products, domain testing for them is much easier than the ones without Unicode. In general, there are no different test cases between ANSI(ISO 8859-1) character set and Unicode. Yet, most of the products are still not based on Unicode or Unicode is used as just part of their products.

In the ANSI characters test cases, it is easy to know the boundary of the input domain, and some of the random input testing may be effec-

tive. However, there are incredible numbers of Chinese characters and many of the Latin characters in the world. Random input test strategy is not useful in most of cases.

Thus, before starting testing, testers should carefully investigate character set, and must know as many of the boundaries that are available.

Automated Testing Strategy

There are infinite numbers of input domain test cases because each country version has its own character set and there are many more character in the Chinese languages. Therefore, the only way to test those infinite numbers of character inputting is with an automated testing tool. Fortunately, most of American automated testing tools, such as WinRunner and Visual test, can handle any international character.

B. Unicode (ISO 10646)

ISO 10646 is also called Unicode, which has great benefit to reducing developing and testing costs. But, Unicode is not a perfect solution in the Asian character set environment though some English speakers think it is.

As it is known, Windows NT, HTML 4.0, and Java implement Unicode, but Japanese people still think several problems[3] of Unicode should be solved.

The main Unicode problem is that Unicode does not support complete Chinese character set that is used in China, Japan, and Korea. There are slight differences in Chinese character face among China, Japan, and Korea. Though Asian can read all Unicode characters even when there are slight differences, Asian may feel odd in reading Unicode character. Of course, when the differences can be ignored for some reason, Unicode should be used because of its benefit, such that:

- Can write a program in multiple languages. We do not have to have a different program for releasing multiple. Microsoft Internet Explore is one of the sample products, which uses Unicode technology.

- Easy maintenance

  Writing a program with Unicode, we will have great benefit of maintenance ability. When developers code without Unicode, the program will be adjusted to all languages. We might need to build for every language. It means we have to make each build file. Also, we might need to insert such code without Unicode program:(Fig. 4)

```
#ifdef DBCS
#ifdef JPN  //For Japanese
//Do things for Japanese
#endif
#endif
```

Fig. 4
Low maintainable Code

These kinds of code clearly make code complicate and reduce maintenance ability.

- Easy to Test

  In terms of testing, we can delete large portion of domain test, and font test case. The input domain testing can reduce the testing cost 40% to 60%.

Unicode Testing Strategy

Because of the reasons above, before releasing product, Chinese character font shape must be checked whether Asians accept or not.

C. Software Resources

When Lotus wanted to introduce its original 1-2-3 spread sheet to the European market in 1983, Lotus Development first had to rewrite

most of the North American version code[6]. And it is still necessary for bad international program rewrite a lot of their code because they do not pull resources code out nor have another resources files which distinguishes main control program and resources, such as Fig. 5

```
Printf("Error: Invalid File Name\n");
```

Fig. 5
Hard-coded Resource

To avoid such a problem, all resources, which are string, font face, font size, should be taken out from program control source code. Otherwise, we will have a couple of problems.

- Translators change C or other sources code directly.
- We should make all language object and executable files.

It is not wise to make each executable code for each language. In addition, this makes the code reduce maintenance ability and quality.
In case that a product is capable of handling international code set, and all the resources are pulled out, we will have a single product for all language without any recompile work.
Practically speaking, there is another benefit to have one set of executable files for an international product. If we have series of products for a Japanese version, north American version, French version and so on, its operational and manufacturing cost will be much higher than one the product.

Dummy Localization Test

Dummy Localization testing is one of the successful testing approach for internationalization. As I mentioned above, we must not hard code any translated string code.
But, unfortunately the translation work would happen in the last stage of development. Because, in most cases, user interface keeps changing from beginning through ending, such as adding dialogs, and adding error messages. Therefore, even if developers take out hardcode and move it to resources, testers can work with it only after translated resources are available.
Consequently, it is the best to add dummy localized string into the real program instead of translated string. We forcedly insert mean-less localized string into international programs. When we do so, we can know whether international programs accept DBCS and extended characters. Dummy localization testing has several benefits:

- Can know whether all resources are not hard code'ed
- Can know whether the code handle Euro extended character and DBCS character
- Can know whether the code choose correct font face

Additionally, especially finding following character set bug/problems.

- Can know whether a product works with string that include 0x5c characters in trailing byte.
- Can know whether a program works with high ANSI character
- Can know whether a program works with mixed characters that are mixed single byte and double byte character.

D. Cultural and Geographical

Cultural and Geographical issues seems difficult if we live in another cultural setting.
In terms of cultural and geographical issues, I believe testers are not supposed to be expected find them out. A international program manager or other related people should take care of them. Yet, as I explained in introduction, a tester is often the only person who is checking products regarding the quality and other aspects. Thus people expect, it is natural that a tester should be expected to find bugs related to culture and geography. Likewise, North American

testers may find own cultural bugs.

1) Sample of geographical issues

First of all, is Taiwan a country? No. Taiwan is the name of a region. Is Hong Kong a part of China? The answer is Yes. It means that we cannot be too careful if products sell both in Taiwan and Mainland China. Products may have dialogs, which show our branch company. The dialog box title includes "Country or Region". Otherwise, we cannot sell the products in China.

Test Strategy

It is difficult to find such cultural or geographic bugs. The best way is to test these kinds of bugs by testers in each country where we sell products.

E. Keyboard

Some developers still use low-level API to get key input information, for example, int 21h in PC environment. Now, almost all the platforms, such as Sun Solaris, Linux, Windows 9x have international keyboard API. We must use these API calls, which support international program. If not, users cannot type some of the characters. In the worst case is when users type 'A', the program displays 'B' instead of 'A' on Japanese NEC PC platform.

Test Strategy

In general, people except North Americans use two types of keyboards. Some people use an English keyboard, other people use a localized keyboard which is capable of typing localized character directory. On the other hand, by using English keyboard, people type localized character indirectly, for instance, typing twice and typing combination. Thus, we must test both English keyboards and localized keyboards to confirm that the applications can handle user input behavior in all input.

F. Font checking

First, I will show you typical font error in Fig. 6. Fig 6 uses International font instead of ANSI font(Fig. 7).

Releasing products without font checking or string checking, we may see this kind of bug. If the companies really care about legal issues, they must stop selling the product until fixing this bug. We must know that there is a different character set over 0xA0 between ANSI and international fonts.

Florida Tech_ Miami Windows _

Fig. 6
Using International Font

Florida Tech® Miami Windows ©

Fig. 7
Using Ansi Font

Test Strategy

Testers must test over 0x80 character set for displaying English characters by using correct international fonts.

G. IME (Input Method Editor)

There are over several thousands of Chinese characters in Korean, Chinese and Japanese. Thus, we have to use IME (Fig. 8), which converts phonetic symbols to Chinese characters. Though double byte handling and keyboard handling have similar interface in any international

computing environment, such as UNIX, Windows. There is no standard program interface to input double byte characters, and many problems come from this area.

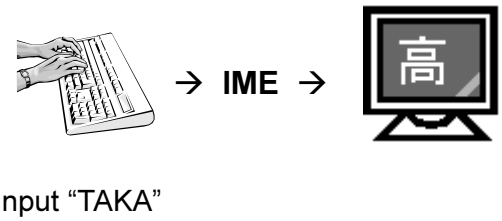


Fig. 8

System to input Chinese character

Test Strategy

IME testing is the most difficult part of testing because IME keeps sustaining to convert character inputted into double byte character during computer running. Under most of the operating systems, the modules of these IME programs are separated from the program and the operating system.
As the same token, mouse drivers have similar functions. A bug as this example is that some applications might not handle DBCS correctly when users use advanced mouse control and IME. This mean inputted 'a" is shown as 'B'. Testers are supposed to carefully check these memory conflicts between IME and international program.

H. Compatibility with local operating system and local hardware

Most operation systems have been changed to adjust to each language system. This means that the program has a possibility to fail to run on a local operating system. Nobody can assure that a program, which can run on English operating system, can run on a localized operating system. For example, my international project team released USB devices with PC USB control program. At that time, we considered there was the only one USB standard in the world. Therefore, American test manager did not do any USB interface testing between USB driver and international USB control program. He planned that the only user interface test, which is checking strings and fonts, was required. Unfortunately, even North American version had 10% of USB driver compatibility problem. And we tested compatibly between the USB driver and international control program, we could know 70% of PC have problems with Japanese OS because there was critical issue between USB driver and Japanese operating system. Because of this problem, we finally gave up selling USB system in Japan.

1) Video

Drawing and animation functions produce much more bugs in the recent computing environment since most of the current applications have used graphical user interface.
In this computing environment, redraw is one of the major bug areas. Because Chinese characters are wider than alpha numeric Fig. 9 , we especially need to care about it.

Fig. 9

Difference width between alphabet character and Chinese character

Test Strategy

If products are used in the window operating

system, which is capable of graphic user interface, testers are expected to test all drawing areas with Chinese character.

I. Other Stuff

I listed most of the undocumented information, which was not concerned in any books and paper so far. In terms of common internationalization issues, which are data format, currency format, different user interface, I recommend that you read [1] [2] as well to do complete international testing.

V. Test Pass

Through the section above, I have already explained every aspect of what we are supposed to test before releasing international software. I believe that we must finish the test before releasing international product. In addition, we must run full test cases, which are designed for North American version. Most people think testing only international related area is enough. However, it is not enough. Software will run on local operating system. Naturally non-English speakers do not use English operating system but localized operating system. In terms of Windows Operating system, Microsoft has changed their Win32 API code to adjust local environment and language. For example, even if we localized products, which do not have any input area and cultural issues, any localized string, we would pass the same test cases as North American version. Fortunately, in some cases, North American version has not been hang up on localized OS. But unfortunately, the program often does hang up on localized operating system. We assume it depends on if operating system gives good memory. We can hardly know this kind of bug without running full test cases.

Thus, we must run the whole test cases on the local OS platform to find such bugs. Thielen[4] mentions that once we change a code, we are supposed to execute all test cycle.

A. Automated Testing

Kaner [5] recommends automated testing for international products because the international products have essentially the same functionality and user interface as the North American version.

Additionally, I recommend that we make reusable international automated test cases. When we use automated test case ,they include a great number of hard code strings, such as for picking up menu control, error message, short cut key to control automation test scripts. These items are localized into other control name. During developing North American automated test cases, we should not write in the hard coding style. We must pull out the string resources from control code, and recreate them individually to outside file or header file to reduce international testing cost.

VII. Localization Side Effect

In the past decade, most of the international products did not share any source code between North American versions and international versions. In addition, North American versions are always developed before localized ones. At that time, we did not have to think about localization side effect. But today when we develop international software simultaneous, we have to carefully develop localization part of code because there are the same source code and the same schedule in the international project. I define the localized side effect is that:

- Localized side effect is a side effect from changing the code to fix bugs in certain language. The changed code causes a bug in another language product.

For an example of localized side effect, if we change keyboard function code for Japanese

version, there will be a possibility to have another bug for French products. Another example is when we developed Japanese products, we fixed IME (Input Method Editor) bug for Japanese version. That bug fix was that we prohibit to input Japanese characters in one of the edit boxes. Unfortunately, the same product of Chinese version, users cannot input any Chinese characters in all the edit boxes on Chinese OS, since we fixed Japanese IME bug.

Test Strategy

Regression test is much more important to check localization side effect among international products than any other tests. When we develop an international product that is more than one language version a multi-language version, a test manager must establish the information sharing about localization side effect among international testers to do a regression test.

VIII. Testing Process for International Software

Rafii[6] defined internationalizing software development process as three phase:

1. Base product engineering
2. Internationalization
3. Localization

Such as Fig. 10 :

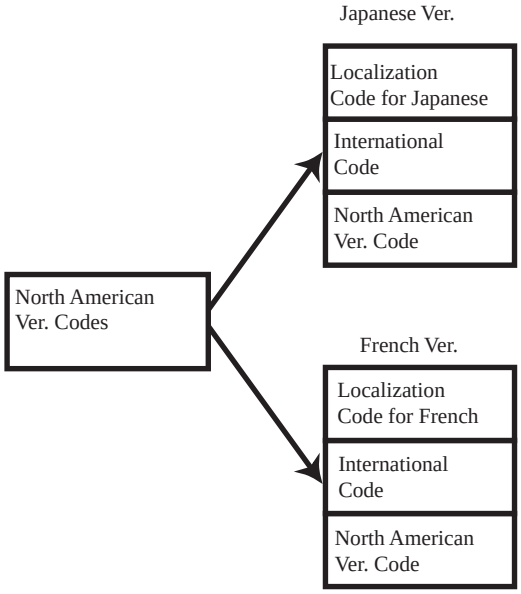


Fig. 10
Three-phase development style

In case that North American developers do not know anything about internationalization, they try to write code without consideration for international. Consequently, developing costs would be expensive because there is a duplication task for foreign language versions. Fig. 10 shows the case that is three-phase development style when every country project team is supposed to develop international part as duplicated task. If we develop a North American version and only one foreign language version, this way is acceptable. However, if there are more than two language versions, in most cases, the developing and testing cost would be expensive to conduct three phases development.
Therefore, we should have only two phases instead of three phases, such as

1. Base-product engineering and internationalization
2. Localization

See Fig. 11

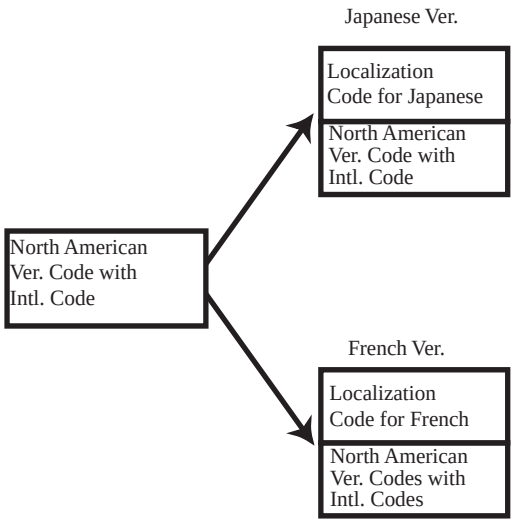


Fig. 11

Two-phase development style

A. Maintainability

Suehiro[7] also mentioned about maintenance ability. If there are more than two versions of the products, it is very difficult to keep international part code because we must merge back additional international code to a North American version as well as a localized one. Considering maintenance ability, we should take two-phase development style.

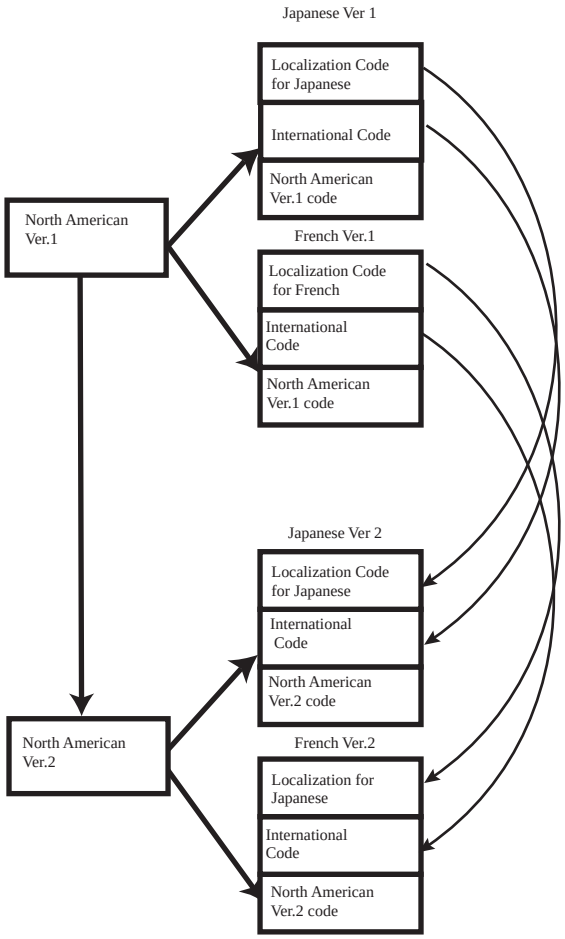


Fig. 12

Low maintenance-able upgrade

Test Strategy

Before releasing a localized version, a North American version should be completely tested for international parts of the product to reduce duplicated task in the localization test team. Otherwise, for example, Japanese and French testers need to test the same parts as duplicated task.

VII. Organize an International Test Team

Rafii [6] researched the development team project on centralized and distributed. I will break down more of his research since his research did not mention the development team function.

In almost all the cases, according to my experience, developing/coding task should be done by the North American team. American developers know much more details of code. Even if developers can not read/write foreign language, they could easily understand what international products require and read international source code which are written in C , Java or others. However, testing team has some options in the working location.

On the other hand, organization of international test team is very nasty issues to test international product because, in some cases, American testers need to test unknown language and world' software.

A. Type of Organization

There are 3 type of organization to have development team

1) All work done in Local Country

When testers work in local, they could know the local configuration and the user trend. And it is easy to distinguish important bugs and trivial bugs. Sometimes American engineers make a mistake how much the bug is important and risky in the local country. Especially, they can hardly understand what Asian local configuration and user trend are. Thus, for example of Microsoft, they have 4 local R&D center in Beijing, Korea, Taipei, and Tokyo though there is the only one Euro R&D center in Dublin because of reason above.

On the other hand, it is difficult to know how progressive the main part of products is from remote testing team. The Lack of communication between North American based developers and local country testers often cause troubles. When we test international version in local, we need to establish communication protocol between North American developers and local testers.

2) All work done in North America

Testers can communicate easily with core product development staff if developing staff is in North America heading R&D. Distance is a big issue; face-to-face communication is often more efficient to find solution than the exchange of e-mail [8] . Especially, it is easy to reproduce bugs by using the same machine or debugger between developers and testers when they are in the same place. In most cases, this style is cost-effective and we should try to take this way first.

3) All work done in both side

Since we erase the risk of both local country testing and north America testing, sharing testing work is the most logical testing style.

B. Other issue

Generally, wages in developing countries is lower than those in developed countries. We can consider testing cost in each country. For example, the wage in Japan is higher than that in the United States when a product is localized in Japanese, it may cost more to test the product in Japan than any other countries.

On the other hand, in India, the wage is less expensive than North American countries, and it is easy to hire skillful engineers who speak English.

IX. Scheduling

As Myers[9] pointed out, "It is generally recognized that the earlier that errors are found, the lower are the costs of correcting the error".

However, this rule is not appropriate for international testing. Too Early testing cost much for international products. For example, if North American version is still unstable to run and has high MTTF(Mean Time To Failure) value, it is better not to start international testing. If we do so, testing costs may increase because it may have into more MTTF than North American version does. In such case, an international product may be reboot'ed more than hundred times a day. The appropriate schedule is divided into two phases.

A. The First Phase

1) Test if there is hard code string or not

Almost all the developers do not want to change their code that work correctly even when the international program needs to adjust internationalization. Thus, checking hard coding will start at the same time as North American version coding starts.
Once developers pull hard code out , testers should use dummy localized testing.

2) Test if a product handle extended and DBCS characters.

Writing code for DBCS and 8 bit character set should be developed in the first stage of development as the same reason above. There are different string handling code between ANSI and international character set(Fig. 3). These testing should be happened the first phase. But some developers may say that they could change it after finish the North American version, Yet this idea is wrong and increase international developing and testing cost(Fig. 11)

B. The second phase

In finishing the work for both erasing hard code sting and DBCS enabling, developers will do specific work in each language, such as IME or handling vertical writing, handling right-to-left writing for Arabic, and so on. Consequently, testers are supposed to test these functions.

X. Cost

Estimating an internationalization testing cost is difficult because there can be different approaches from both general cost estimation and international cost estimation.
There are several aspects to estimate cost of testing, such as:

- Target product handles double byte system.
- Target product handles high ANSI characters
- Target product handles 0x5c character set
- Team does hot have help from local engineers.
- Target product is written in Unicode
- Target product program does not have any hard code.
- Target product is shipped simultaneously with North American version.

When products include most of all items above, it is better to re-design the product. Otherwise, the products need cost too much for internationalization.

A. Cost versus developing style

There is definitely close relationship between cost and developing style to release international products.
The goals of our international product shipment are:

- To Ship with short delay from North American products because of not losing business chances at the local market.
- To ship quality product in local country.
- To ship international version with low developing cost.

Based on the goals, we could take two kinds of

international developing style, which are:

- Simultaneous shipment and simultaneous development
- Non-simultaneous shipment and non-simultaneous development

1) Non- Simultaneous Shipment and non- Simultaneous Development

Originally most of the companies have developed this style for international products. The companies develop North American products and ship them. Afterward, all source code are exported to the localization offices or teams and develop localization products individually. At that time, there was no relationship between North American and international products.
This type of development has:

Advantage:

- Developing cost is low.
- Risk is low whether a product ships on time.

Weak:

- Business chance may be lost in a local country.

2) Simultaneous shipment and Simultaneous development

To be expected good selling in international market, companies choose this method.

Advantage:

1. Sell products worldwide at the same time
2. Easy maintenance

It is difficult to indicate Simultaneous development, localization cost because of delaying schedule. In fact, most of the software development schedules delay. On the average large software projects are one year late and 100 percent over budget [10]. For example, the shipment date of Windows 2000 is being delayed more than two years. Therefore, if the project is delayed for 1 year, localization team must follow the north American schedule, it is no way to ship a localized product. Consequently, international product shipment is delayed more than a year.

We should follow up the localization schedule as well. During the slipping schedule of North West version, localization section keeps staff to develop, test localization version. This situation is really painful for localization development cost. The staff has nothing to do the product development until reaching the milestone.

On the other hand, if we take a Non- Simultaneous shipment and non- Simultaneous development way, we do not have to worry about schedule delay. In the case, usually, localization products start after finishing the North American developing. We can use our staff as best as we can. Therefore, when we decide to take the Simultaneous shipment and Simultaneous development way, we carefully plan arrange testers. We should know the schedule North American version as exactly as possible: when North West version team starts system test, the date of release candidate, and the date of code complete.

In short, when the North American product is going to be shipped without delay, simultaneous shipment and simultaneous development cost may be low. But non-simultaneous shipment and non- simultaneous development way is always low-cost and low-risk to develop international products. However, marketing point of view always focused how well the products sell in local country. And marketing people think that simultaneous ship and simultaneous development have great benefit to achieve large volume of sales. Today, global companies tend to have the same computing system because Internet world makes us easily communicate in the world. Thus, the companies want to have an international product at the same time worldwide.

Because of the reason when we think about scheduling the testing, we had be better talk with

marketing people about how much effective the both marketing profit and developing cost are. Marketing profit and developing cost are inverse.

B. Testing cost versus development cost

In general, localization testing cost is much higher than localized developing cost. In case that ratio between developers and testers is about to be 1:1, the ratio is around 3:1 in localization product development. And testing cost occupies 20 through 50% compared the whole localization developing cost. Thus, testing cost is more worth than code developing.

XI . Conclusion

Fortunately, Windows 2000 was just released, and I can show some good example from this product too. Windows 2000 is the biggest size of software products not only in the U.S, but also in the international market. Microsoft sold international version of Windows on the same day as North West version was released. Yet this product has several critical international related bugs which I explained in this paper.

1. Different date format bug in between North America and other country. This is the most famous and typical bug.
2. When Active directory is used, some Japanese characters are not recognized correctly.

In terms of Active directory, this bug came from localization side effect. Windows 2000 team changed international code to fix German Umlaut(German special character), so this fixed work affected Japanese version at the same time. Consequently, because of this bug, Microsoft has announced that they are sending bug fixed CD to all users. In fact, Windows 9x was sold 7.15 million copies and NT were sold 1 million only for Japan in 1998. Through this fact, we could easily imagine that the Windows 2000 bug has huge impact for international selling and special software testing is necessary before releasing products to an international market.

XIII. Acknowledgment

The author wishes to thank Microsoft co-workers for useful technical discussions.

XIV. References

[1] K. Lunde, *Japanese Information Processing*, O'eilly & Associates, Inc., Sebastopol, CA, 1993

[2] D. Taylor, G*lobal Software, Springer-Verlag*, New York, 1992

[3] *Unicode wa Sukidesuka?*, IPSJ Magazine Vol.39 No.4 Apr. 1998(in Japanese)

[4] D. Thielen, *No Bugs!*, Addison-Wesley Pub Co, September 1992

[5]C. Kaner, J. Falk, H. Nguyen, *Testing Computer Software*, International Thomson Computer Press, Boston, 1993

[6] F. Rafll and S. Perkins, “Internationalizing Software with Concurrent Engineering”. *IEEE Software*, Vol. 12, No.2, Sep. 1995, pp.102-106

[7] Y. Kiyogane and Y. Suehiro, *International Programming*, Kyoritsu Publishing, Tokyo, 1998(in Japanese)

[9] J. Myers, The Art of Software Testing, John Wiley & Sons, New York, 1979

[8] L. Belady, “Global R&D: How to Break Barriers”, *IEEE software*, Vol. 16, No. 2, July/August 1999, pp. 15-17

[10] J. Caper, *Programming Productivity,* New York, McGraw-Hill, 1986

QW2000 Paper 2W2

Mr. Steven Porter
(API / Independent)

From Web Site to Web App: Ensuring Quality in a Complex Environment

BACK TO QW2000 PROGRAM

Key Points

- Differences between websites and web applications
- Things to consider for a Quality Strategy
- Some metrics as a guideline for estimating time needed for testing

Presentation Abstract

The continued success of web development teams will depend heavily on the ability to make the transition from deliverying web sites to delivering reliable, stable web applications.

We must develop a cost-effective strategy for quality assurance and testing through this transition and beyond. The foundation of this strategy must rest on a thorough understanding of the nature and dynamics of applications within a web environment. The key to success is separating the volatile aspects of the application and the web from the non-volatile ones, and focusing on the quality assurance efforts accordingly.

About the Author

Steven Porter hs been a certified instructor for Rational Team Test since 1996. He has written a one-day Quality Assurance class that he teaches as part of the Object Oriented Project Management Certification Series developed by the Advanced Programming Institute of El Dorado Hills, CA.

He has also provided content for the Process Management course of the same series, and has developed the CD that is used with the course. In 1998 he was the QA Manager for a year-long Y2K object-oriented project for CalPERs that involved over 40 reengineered applications.

BACK TO QW2000 PROGRAM

From Web Site to Web App: Ensuring Quality in an Increasingly Complex Environment

Steven D Porter – sporter@2xtreme.net

1

The Medium is The Message

-Marshall McCluhan

- The Web Creates Great Potential
  - Increased Profits from e-Commerce
  - Lower Barriers of Entry
  - Easier Information Distribution and Access
- The Web is Developing at Warp Speed
- The Web is Still Immature, But is Being Asked to Perform in a Mature Manner

2

Agenda

- "Web Site" vs. "Web Application"
- User Quality & Business Quality
- The Web Paradox & Its Effect On Quality
- Quality Strategy & Categories
- Strategic Steps
- Examples for the Categories
- Wrap Up

3

Web Site

- Static Pages Built with HTML
- Organized Documents and Files Accessed Through Hyperlinks
- HTTP Protocol
- Site Accessed Through Browsers
- Capable of Providing Secured Access
- Examples
  - www.iseer.com
  - www.a-p-i.com

Volatility
Complexity
Risk

4

Web Application

- Contains the Elements of a Web Site and Extends the Capabilities to:
  - Allow User Input
  - Perform Standard Database Transactions
  - Perform Business Logic
  - Provide Client and Server Side Validation of Data
  - Perform Queries and Display Dynamically Generated Reports
- Examples
  - On-line Reservation Systems
    - www. AlaskaAir.com
  - On-line Sales
    - www.Amazon.com

Volatility
Complexity
Risk

5

Quality to User

- Responsiveness
- Accuracy
- Completeness
- Ease of Use
- Relevance
- Aesthetics
- Security
- Privacy

6

Quality to Business

- Customer Impact
- Sales
- Business Credibility
- Cost
- Time to Delivery

7

The Web Paradox

- Web Browsers allow for Platform Independence
  - Thin Client / Fat Server
- Competing Web Browsers do not:
  - Uniformly Display Content
  - Uniformly Perform Logic
  - Uniformly Accept User Input
- Examples
  - Buttons Do Not React the Same in Netscape and Internet Explorer.
  - Some Browser Versions Do Not Support JavaScript
  - File Downloading is Handled Differently from Browser to Browser.
- User Plug-Ins Effect Capabilities

8

Effect on the Quality Strategy

- The Strategy for Quality is becoming more Complex due to the Rising VCR Forces.
  - The Web is a Cauldron of Rapid Change
  - The Transition from Building Web Sites to Web Applications Increases Complexity and Risk
  - Escalating User Expectations Increases Complexity
  - Security and Privacy Issues Increase Risk
  - Market Demands Compress Development Schedules

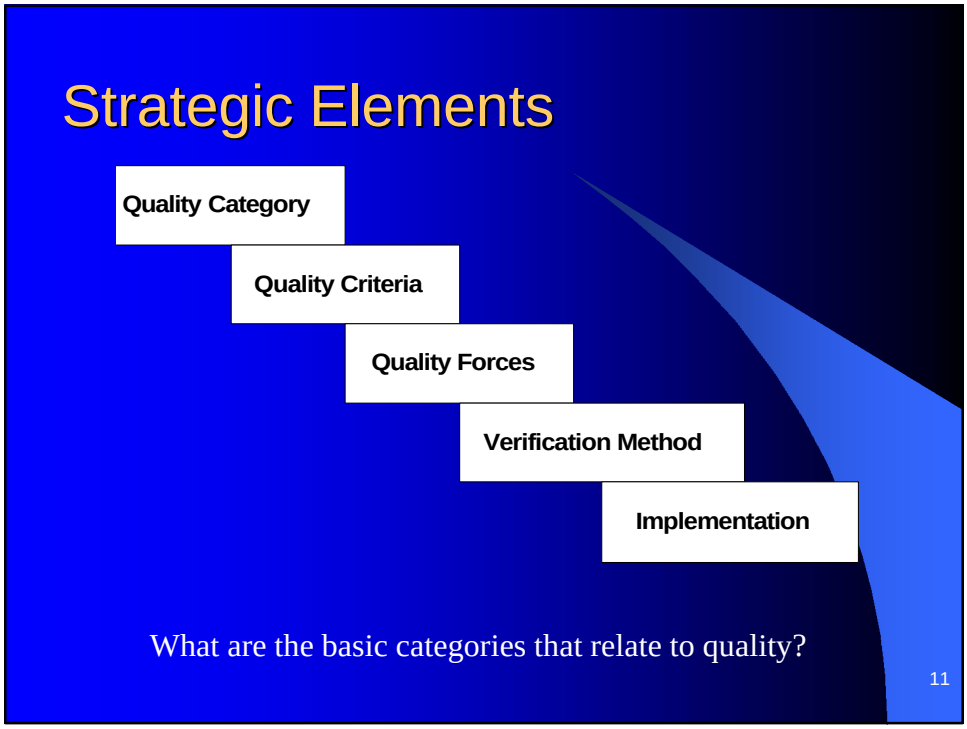
9

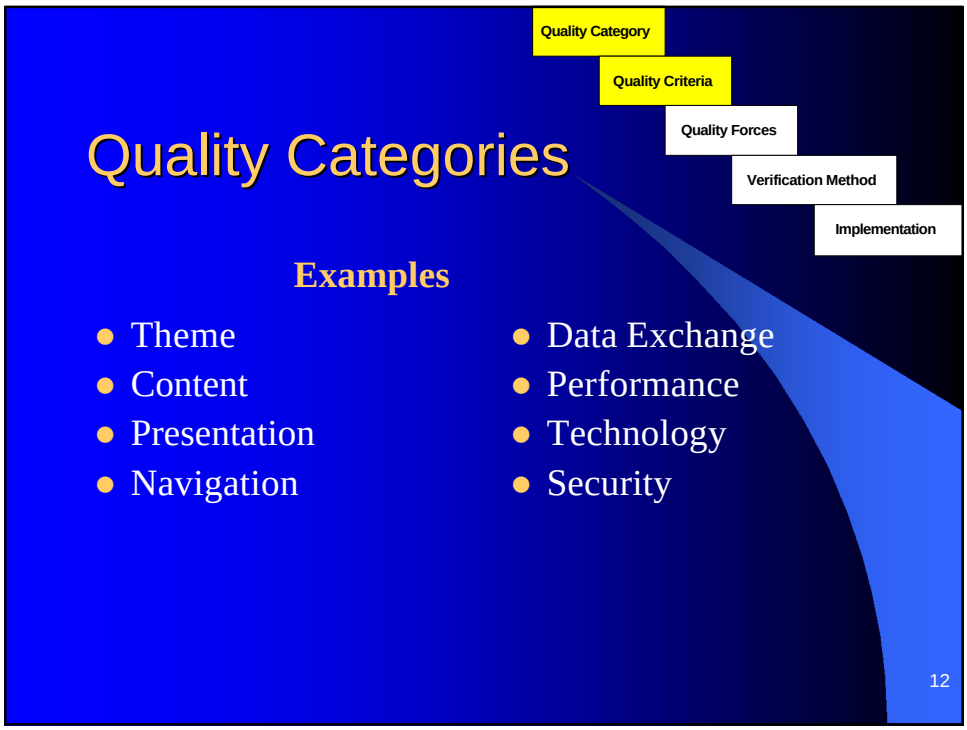
Quality Strategy

- Manage the VCR Forces that Influence Quality
  - Volatility
  - Complexity
  - Risk
- Negotiate Cost and Time

- Organize the Quality Effort into Categories
- Establish the Standards of Quality for each Category
- Manage and Leverage Resources

10


Strategic Elements
Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation
What are the basic categories that relate to quality?
11


Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation
Quality Categories
Examples
Theme
Content
Presentation
Navigation
Data Exchange
Performance
Technology
Security
12

Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation

Create Standards

- Product
  - Based on Quality to the User
  - Based on Quality to the Business
  - Based on Technical Constraints

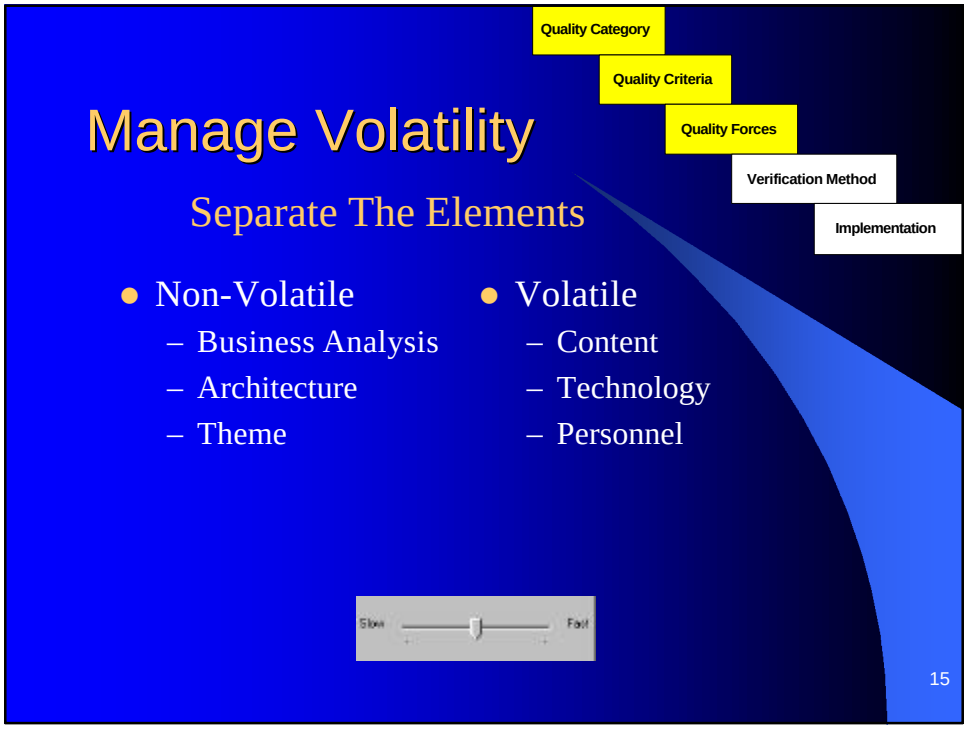
13

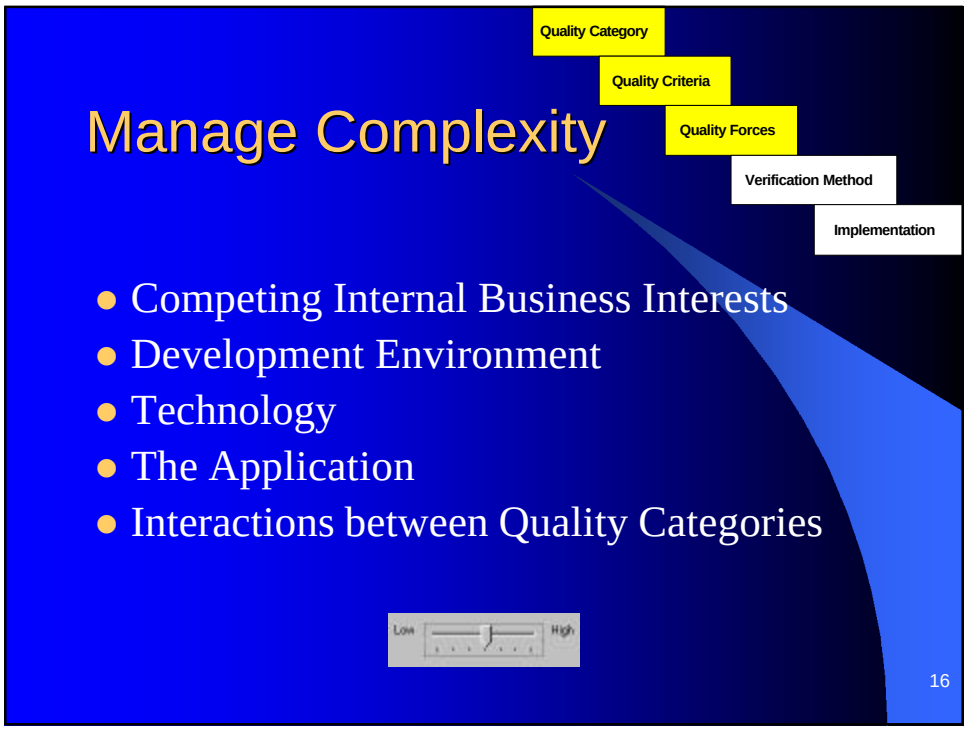
Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation

Create Standards

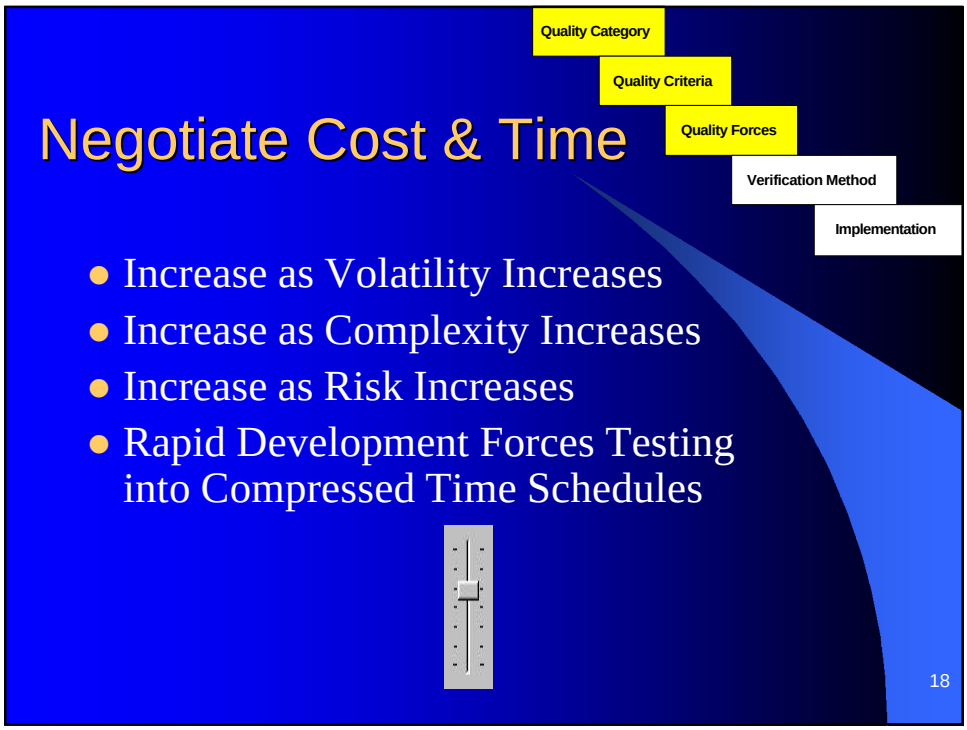
- Development Process
  - Input Artifacts
  - Activities and Tasks
  - Output Artifacts
  - Change Management
  - Roles and their Relationship to other Roles
- Manage to the Standards

14


Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation
Manage Volatility
Separate The Elements
Non-Volatile
Business Analysis
Architecture
Theme
Volatile
Content
Technology
Personnel
Slow
Fast
15


Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation
Manage Complexity
Competing Internal Business Interests
Development Environment
Technology
The Application
Interactions between Quality Categories
Low
High
16

Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation
Reduce Risk
Affects
All Quality Categories
All Development Related Processes
Estimates and Schedules
Identify Analyze Mitigate
Less
More
17


Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation
Negotiate Cost & Time
Increase as Volatility Increases
Increase as Complexity Increases
Increase as Risk Increases
Rapid Development Forces Testing into Compressed Time Schedules
18

Strong Business Analysis

Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation

- Promotes Understanding
- Reduces Volatility
  - Allows for Many Solutions
  - Transcends Technological Changes
  - Maintains Consistency Over Time
- Decreases Complexity
  - Communicates the Same Information to Different Development Teams or Different Members of the Same Team

19

Determine Verification Method

Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation

- Manual or Visual Review
- Spell Checkers
- Link Checkers
- Site Checkers

- Manual Testing
- Automated Regression Testing
- Performance Testing
- Beta Testing

20

Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation

Use the Web

- Web-Enabled Defect Trackers
- Web-Accessible Project Notebook
- Link Checkers
- Web-Oriented Automated Regression Tools
- Web-Oriented Performance Tools

21

Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation

Maximize the Use of Resources

- Create and Maintain an Affordable Testing Environment
- Develop the Application in Phases that Allow Testing to be Performed in Tandem with Development
- Divide the Testing Effort between In-House Resources and Independent Test Labs
- Use Trusted Beta Sites to Duplicate Test Coverage and provide Unusual or Expensive Testing Configurations

22

The Strategy in Practice

- Theme
- Content
- Presentation
- Navigation
- Data Exchange
- Performance
- Technology
- Security

23

Theme

Standards	Corporate Image Site Objective
Volatility	Low Volatility
Complexity	Depend on objective (Inform vs Exchange)
Risk	Consistency with Overall Corporate & business objectives.
Verifications	Corporate Image Guidelines Standards etc. Traceability to site objectives
Implementation	In-House Manual Review

24

Content

Standards	Writing Style Language Usage Legal Constraints Accuracy, Completeness, Relevance
Volatility	Potentially volatile
Complexity	Variable - Depends on audience and message
Risk	Moderate – Depends on rate of change
Verifications	Traceability to Objectives Spelling Syntax Grammar Logical organization etc.
Implementation	Manual Reviews Spelling & Grammar Checkers Professional Editing

25

Presentation

Standards	Graphic & Visual Standards, Corporate Image Guidelines, Fonts, Logos, Trademarks etc. Target Display Standards
Volatility	Moderate – Depends on supported browsers, platforms & versions.
Complexity	Moderate to High – Depend on Html usage (frames etc) Target browsers Dynamic page presentation The use of plug-ins
Risk	Moderate – Affects Performance, Consistency, Reliability etc.
Verifications	Verify that all features of the site /Application work in the supported browsers. (Compatibility Testing)
Implementation	Consider a strategy that supports in-house compatibility testing on current versions of major browsers, and out-source compatibility testing of all other browsers

26

Navigation

Standards	Site architecture Navigation standards User Interface guidelines
Volatility	Depends on site objectives and the outside links
Complexity	Low to High Sequential Hierarchical Grid Web-Linked
Risk	Low to Moderate – Outside links, unstable site architecture
Verifications	Manual review of architecture Verify functioning of all links
Implementation	Use link checkers to: Verify links Identify orphans Slow downloads

27

Data Exchange

Standards	Accuracy & Completeness, N-Tier Architecture, Coding & Validation
Volatility	Low to Moderate for the exchange mechanism.
Complexity	Moderate to High - Depending on N-Tier & Server Environment
Risk	High – due to complexity and web
Verifications	Verify that communication objects pass data correctly between client and the servers, as well as between servers Verify that Web Application handles Add, Modify, Delete and Query capabilities correctly
Implementation	Functional Regression testing Manual techniques Automated techniques

28

Performance

Standards	Established project requirements Industry standards User expectations
Volatility	Low – If changes comply with the standards for performance. Performance stability is affected by the rate of change of the hardware, software and site content and structure
Complexity	Moderate to high. Depends on the Architecture
Risk	High – Concurrent access to the site and database is higher than predicted. Hardware is insufficient to handle load. Application is not designed for optimum performance
Verifications	Volume, Load and Stress Testing across the Web.
Implementation	Use In-House owned performance tools that verify performance for 200 virtual users submitting distinct requests Outsource performance tests for higher numbers of users

29

Technology

Standards	Minimum requirements for hardware and software configurations. Supported environments
Volatility	Moderate to High –We can manage the rate of change of the configurations we control, we have no control over the rate of change of technology, nor the use of it by our customers.
Complexity	High – Client/ Server Hardware coupled with firewall implementations create a complex application environment.
Risk	High – The broad spectrum and rapid change of technical solutions causes the system environment to be unpredictable and uncontrolled
Verifications	Testing configurations. Review changes to server-side and evolving client configurations
Implementation	In-House compatibility testing for primary supported environments. Out-source compatibility testing of marginal environments. Perform beta testing at beta sites with different firewalls.

30

Security

Standards	Corporate Security Policies Fiduciary Responsibilities Purpose of Site
Volatility	High – Changes caused by new methods to bypass current technology
Complexity	High – Various encryption schemes Various levels of security access to users and groups of users
Risk	High –Breaches of security may be catastrophic to the business and the users
Verifications	Automated and manual monitoring
Implementation	24/7 monitoring Immediate notification for security breaches Mitigation Plan

31

Summary

Quality Category
Quality Criteria
Quality Forces
Verification Method
Implementation

- Web Sites are increasingly becoming complex Web Applications
- Establish a Clear Business Direction and Purpose
- Manage the Quality Forces
- Divide and Conquer

32

Contact Information
Steven Porter
3034 Estepa Dr.
Cameron Park, CA 95682
Cell: (916) 718-5734
Sporter@2xtreme.net
Thank you to David Wilkerson for the hours spent helping me organize, clarify and edit this presentation.
Dwilkers@calweb.com
33

QW2K Paper 2M2

Ms. Johanna Rothman
(Rothman Consulting Group)

The Influential Test Manager

BACK TO QW2000 PROGRAM

Key Points

- An introduction to ways to influence others in the organization
- Reframes of typical test manager problems and possible solutions
- What to do when it doesn't look like it's you

Presentation Abstract

Many of us have worked in test groups where we felt as if we didn't have enough time, hardware, or staff to do the work. Consequently, we feel that while somebody might be in control, we are certainly not. As test managers, we do not have to work this way. There are other, more effective ways to develop and use your influence within your organization to help your test groupùand projectùsucceed.

About the Author

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people. Johanna is the founder and principal of Rothman Consulting Group, Inc., and is ASQ certified as a Quality Auditor and Software Quality Engineer.

BACK TO QW2000 PROGRAM

The Influential Test Manager

Johanna Rothman
Rothman Consulting Group, Inc.
www.jrothman.com
781-641-4046
jr@jrothman.com

What's a Test Manager to Do?

- We have lots of problems, many of them tactical
  - Too many defects in the product
  - Too little equipment
  - Cascading schedule slips
  - Not enough people
- We have strategic problems too
  - How we are perceived within the organization
  - How the organization does business
- Influence others to change

© 2000 Johanna Rothman

What Is Influence?

- From the dictionary:

"The act or power of producing an effect without apparent exertion of force or direct exercise of command"

"The power or capacity of causing an effect in indirect or intangible ways: sway."

© 2000 Johanna Rothman www.jrothman.com 3

Influence

- Mutually valuable decision about a problem
- Swaying other people to your point of view
- Producing an effect without an apparent exertion of force
- Influence is not
  - Control
  - Taking away choices

"A man convinced against his will is of the same opinion still" -- Anonymous

© 2000 Johanna Rothman www.jrothman.com 4

Unique Perspective

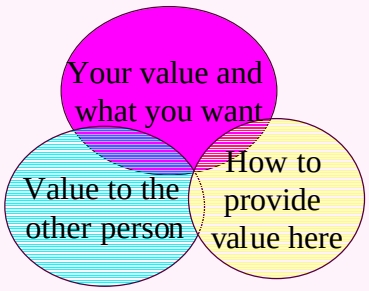
- "Testers hold the flashlight" -- James Bach
- Test managers can be change agents
  - Risk illumination
  - Risk management

© 2000 Johanna Rothman www.jrothman.com 5

Exchange: the Essence of Influence

- Define the problem context
- For you and the other person: What's In It For Me? (WIIFM)

- What do you have of value?
- What does the other person want?
- Choose how to provide value
- What to ask for in return



© 2000 Johanna Rothman www.jrothman.com 6

Test Manager's Mission

- Many possible missions
  - Find and report defects, especially Big Bad Bugs.
  - Test this software beast until they pry it from my unwilling fingers, finding and reporting on all defects.
- The mission I'll use here:

  Assess the state of the product under development at any time, and report on that state
- How do you use your mission to define value?

© 2000 Johanna Rothman www.jrothman.com 7

Tales of Three Test Managers

- Each test manager wanted to hire more people
- Each test manager was initially turned down by his/her boss

© 2000 Johanna Rothman www.jrothman.com 8

Andrea Was Not Influential

- Wanted to hire 2 more testers immediately, 2 more before the end of the year
- VP only wanted to hire support people because of perceived salary cost
- Can see technical debt coming, but can't do anything about it
  - Cascading effects of technical debt:
    - Higher support costs
    - Higher test costs
    - Higher development costs

© 2000 Johanna Rothman www.jrothman.com 9

Bill Used Influence Across the Company

- Wanted to hire 2 more testers
- Approached the Sales VP: "If you support my request for more testers, I can give you pre-sales support after the testing effort is complete."
- Approached the Engineering VP: be able to assess ship risk better

© 2000 Johanna Rothman www.jrothman.com 10

Carol Influenced Within the Product Development Organization

- Too few testers for development staff
- Enormous technical debt
- Worked with management to make the product release dates
  - Multi-pronged plan, including
    - Project manager
    - Peer review of bug fixes
    - 4 more testers

© 2000 Johanna Rothman www.jrothman.com 11

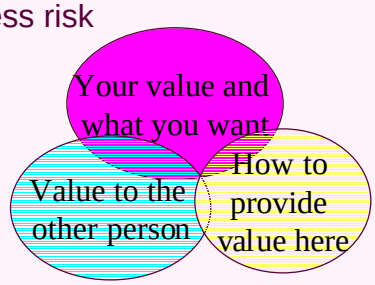
What You Have of Value

- Personal value
  - Credibility
  - Fairness
  - Track record
  - Management capabilities
- Test Manager value
  - Information about the product under development or test
  - People who test
  - Machines that execute the tests
  - Tests

© 2000 Johanna Rothman www.jrothman.com 12

People Use Their Value Differently

- Sometimes, trade project information to obtain a specific end result
  - Is that result valued?
- Sometimes, trade capability to assess risk
  - Is the assessment valued?
- Others must value what you offer, or you become less valuable
  - ➡ Result: lessens your influence



© 2000 Johanna Rothman www.jrothman.com 13

How You Provide Test Manager Value

- Assess risk
  - Risks inside testing, and across product development
  - Who's affected by technical debt or schedule slips?
- Identify and solve testing problems
  - What are your alternatives when the test time is inadequate?
- Manage test and measurement work to assess product state
- Helping make ship decisions

© 2000 Johanna Rothman www.jrothman.com 14

Assessing Product State

- Did we build the product we intended to build?
- How much have we tested?
- How stable is the product?
- How complete is our information?

- You may have other questions to add to your list that reflect your goals and priorities

© 2000 Johanna Rothman www.jrothman.com 15

Helping Make Ship Decisions

- Are you in a good state to influence people?
- If you don't fall into classic testing mistakes
  - Being the ***only*** one responsible for product quality (per Marick)
  - Being the ***only*** one responsible for assessing product shipment risk
  - Being the ***only*** one who can see the implications of shipping the product in its current state

- When you're the only one with the data,
  and you make all the decisions,
  you will make wrong decisions

© 2000 Johanna Rothman www.jrothman.com 16

What Do You Want in Return?

- Assume your managers aren't stupid or corrupt
  - They may be ignorant, but they're probably not stupid
  - You choose whether to placate corrupt managers

© 2000 Johanna Rothman www.jrothman.com 17

Use Your Value to Decide What You Want From Your Organization

- Be clear that you're in a two-way relationship
- Explain what you want (People, machines, time, other resources)
  - The implications of shipping defect-laden products
  - Funding the test group's equipment requests will help the other managers know sooner when the product is ready to release
  - Shortening your test schedule will prevent you from collecting enough data to know whether to ship
  - You should hire enough people to do the right amount of testing so everyone will know when to ship

© 2000 Johanna Rothman www.jrothman.com 18

When It's Not You

- If you're responsible or product quality (classic mistake)
  - Reframe that to "Here's how I can ***help you*** make that quality happen"
- If you're the only one assessing product ship risk
  - It's unwise to have a decision of such broad implication with only one person (a single point of failure)
  - You may not know the business decisions involved with this product release
- If any of the managers are having a bad-manager day
  - Wait until you're all centered, sure of what you have to offer each other

© 2000 Johanna Rothman www.jrothman.com 19

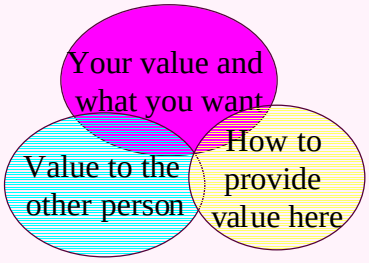
Influence May Not Work Immediately

- Maybe people don't know what they really want
- Maybe people don't know their value to the organization
- You can
  - Help define their value
  - Help them understand your responsibility
    - If your group doesn't put the defects in, you are not responsible for getting rid of the defects

© 2000 Johanna Rothman www.jrothman.com 20

Influence Works

- Product ship decisions or other risk-based decisions
  - Knowing what you have of value
  - What's in it for the other person
  - What you want in return



© 2000 Johanna Rothman www.jrothman.com 21

Influential Test Managers

- Provide value
  - Ask questions and assess answers about the product and project
  - Facilitate release tradeoff discussions
  - Generate and test release criteria
- Understand what the other person wants
- Choose how to address that want
- Decide what to ask for in return
- Control is not the issue
- Share your risk perceptions to influence your project and make it successful

© 2000 Johanna Rothman www.jrothman.com 22

The Influential Test Manager

© 2000 Johanna Rothman

Many of us have worked in test groups in which we felt as if we didn't have enough time, hardware, or staff to do the work. In those situations it's hard to escape the feeling that while *somebody* might be in control, *we* are certainly not. As a test manager you don't have to work this way. There are other, more effective ways to develop and use your influence within your organization to help your test group—and project—succeed.

Influence is not about control, nor is it about taking away other peoples' choices. Influence is about coming to a mutually valuable decision about a problem—redirecting your organization's power and focus. As a test manager, you may want to influence others in the organization when:

- Your management wants to release a product you know is full of defects, and you're sure the customers will be disappointed with the release.
- Other managers ask you to give up equipment your test group needs to do their jobs, or you can't get the equipment you need.
- Schedule slips shorten your test schedule, and your only alternative seems to be to test less.
- Your budget isn't large enough to hire the people you need.

These are examples of tactical problems in obtaining specific objectives. You can also exercise influence more broadly—strategically—to change how the organization perceives itself, or to change how it does business.

As test managers we can be valuable agents of change. We have a different perspective on these problems than other managers in our organizations, because of our role in planning and measuring testing to obtain and share information about the product under test. If testers, as James Bach says, "hold the flashlight," then test managers figure out at least some of the places to shine that flashlight, and how long to spend flashing—a specific form of risk management. Because we illuminate risk from a different perspective than anyone else on the project, we often have a unique approach to problem solving.

We can influence other people to see our perspective and agree to it, or to come up with an even better solution to our problem.

What is influence?

A dictionary definition of influence is:

"The act or power of producing an effect without apparent exertion of force or direct exercise of command"

or

"The power or capacity of causing an effect in indirect or intangible ways: sway."

To sway someone, you get that person to change what they're doing. Sometimes you give them something they want—their WIIFM (What's In It For Me). Sometimes, you get them to reconsider their WIIFM, to enlarge their wants to include more of the organization. To be truly

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

influential, first you must become fully aware of your problem context—the setting or environment where the problem exists, including all those who might be affected by it. Once you've surmised the problem context, you can decide how to influence the other person. (You might even change your perspective, based on your understanding of the problem context.)

Consider how you can provide value, and finally, what your "influencee" wants. Then, you can choose how to provide that value and what to ask for in return. This value exchange is the essence of influence.

Note that I'm not talking about a strictly monetary exchange. Sometimes you loan your credibility to some project your influencee wants to succeed. Sometimes you give your time to some other group, or perform some other service to the organization. Sometimes you get a budget increase to give more value to the organization. View your problem within the context of your organization, so you can see what you want and what you're willing to give.

Share the "problem context"

Many test managers don't have the staff they need to do an effective job, nor do they have the budget to hire the staff. One way to define that problem is:

I don't have enough testing staff.

That problem statement is a good start, but is limited in scope and shows no way for you to provide value to more of the organization. An alternative problem statement is:

I don't have enough people to execute my mission.

That's vague as it stands, because it depends on what your mission is. Your mission could be: "Find and report defects, especially Big Bad Bugs." Or your mission could be: "Test this software beast until they pry it from my unwilling fingers, finding and reporting on all defects." There are many possible test manager missions—maybe as many as there are test managers! I choose to define the test manager's mission as:

To assess the state of the product under development at any time and report on that state.

With this mission statement, I can describe the problem context as:

> *Without more people to do the testing, I am unable to sufficiently assess and report on the state of the product as well as I would like. I can't give you, Project Manager or Senior Management, the picture I think you need to see. That means you, the management team, will make decisions with sketchy information. You can still make the decisions, but you are assuming risk. I don't know how to give you better information without more people. If you want to take the risk of operating with very incomplete information, that's okay. But I recommend we get more staff so you don't have to make blind decisions.*

This problem context statement brings the problem's significance to the rest of the organization. It shows that the problem is bigger than the test manager whining about not enough people. It makes inadequate test staffing an organizational issue, not a testing issue. Now that you've shared the problem context, consider what *you* have of value to the rest of the organization.

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

What do you have of value?

You provide value both as a person and as a test manager. Your perceived credibility, fairness, track record, and capabilities as a manager have value not just to the test group, but to your peers, your managers, and the rest of the organization.

Your own level of test-manager-value depends on how you define your mission as a test manager. Your mission defines the work you choose to accomplish: with the test organization as a whole, with the project manager, through your mentoring of staff, and in the way you structure your team's work. When you perform that mission, you have at least these valuable items:

- Information about the product under development or test
- People who test
- Machines that execute the tests
- Some number and type of test procedures (e.g. automated and manual, complete tests or smoke tests)

Different people use their value differently, even when going after the same goal (getting funding for more testers, to provide more information about defects to the organization). Here are stories of three test managers and their approaches to getting more positions funded in their organizations.

Andrea

was a test manager who also ran the customer support group *and* the technical writing group, in addition to the test group. She had two testers, four technical support reps and two writers. (Her organization had six developers on staff.) Andrea needed at least two more testers immediately, and wanted an additional two automation and tools developers over the next year. Andrea's approach was to tell her boss, the Vice President of Engineering, exactly what was necessary: "I need two more testers right now, and two more over the next year."

The Vice President's response was bottom-line-based: "How expensive are those testers? Every time you ask me for more people, it costs me money—as much money as a developer. We're a young company, and we have to make our balance sheet look good. Otherwise our investors will be upset, and we'll be in trouble. Live with what you've got."

Andrea and her boss have had this same conversation at least half a dozen times. Every few months the development team adds more people to its staff, but Andrea can only add tech support people—because they cost less. As the number of developers has increased, so has the software's complexity and the size. Andrea *doesn't* have enough staff to test the product sufficiently, so the company is building up its **product technical debt**.

[Technical debt, as defined by my colleague Dave Smith, is the debt a company "owes" to a product they persisted in shipping in an incomplete or unstable condition over several releases. This is a situation referred to by Hunt and Thomas's book *The Pragmatic Programmer* as "software rot"; but the software isn't necessarily rotten—it's just incomplete.]

At some point, as the technical debt increases, the load on the customer support staff will be so overwhelming that the company will have to use developers to staff the phones and solve the support problems. Andrea can see this coming, and knows there is a window of opportunity

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

before the “technical debt overload” overwhelms the company’s ability to do development—and actually prevents new product development. She *knows* she needs more testers to prevent this from happening. But she doesn’t know how to explain the significance of what she’s seeing to her management. Even though Andrea can see what’s happening, she hasn’t defined her organizational value in a way that would allow her to address the complete problem.

Bill

has approached the technical debt overload problem differently. Bill also has responsibility for both a test group and the technical support group. In his organization, the writers are part of the development team. Bill already has five testers, including one full-time test tool and automation developer. (There are ten product developers.) Bill wants to hire two more testers—because two releases ago, the test group fell behind the developers and significant parts of the product were not fully tested. (Technical debt is starting to affect Bill’s organization. The testers don’t feel as if they are fully testing the product, and the tech support folks are hearing about the same defects repeatedly.)

Bill used the assessment and reporting mission statement to explain the problem context, the problem significance, *and* his value to the situation:

> *If I can get two more people to do the testing, I will be able to assess and report on the state of Modules A and B. Since the product is evolving to be based more on Modules A and B, I will be able to provide management the information to understand the development progress, and make ship decisions. If I get two testers now, I can do enough testing of Modules A and B to avoid incurring more technical debt, and having to keep people testing the inevitable fixes after we ship the product. I can use this release’s testing as an early warning signal to anticipate the load on the technical support group.*

Bill had a two-pronged approach. First, Bill went to the Sales Vice President and asked if she needed technical pre-sales support. (He knew she did, because the sales reps constantly called the support reps for help before they went on sales calls.) When she said yes, he made an offer: “If you support my request for more testers, I can give you phone-based pre-sales support after the product ships.” (More testers helped find defects before the product shipped. The testers could focus on the next release, and the support staff wasn’t kept busy finding and reporting already-known defects.) The Sales Vice President was thrilled, and agreed to support his cause in the Senior Management meeting.

Then Bill went to his boss, the Engineering Vice President, and made another offer: that if he got two more people in testing he could provide benefits across the organization.

- Engineering would know about the defects earlier. That would help them choose what to fix and when.
- Knowing about the defects would help the testers react quickly to areas of the product, choosing where to do more of which kinds of testing.
- Bill would be able to anticipate the support load, and possibly help the company avoid having to hire more people for support.
- The support staff could spend more time duplicating new customer-reported defects—which would help the developers—instead of writing up more already-known defects.

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

He said, “If you sign two more testing requisitions, and help me hire more testers before the freeze milestone, I expect to be able to provide pre-sales phone support to our sales staff. That should increase our sales from this product.”

Bill’s boss said he’d bring it up in the budget meeting, and was pleasantly surprised that the Sales Vice President supported Bill’s request. Bill got his requisitions, and was able to hire the people he needed.

Andrea’s value to the organization is more limited than Bill’s. Andrea is focused on the testing part of the problem, which is not sufficient for showing the value she provides the organization. Bill realized he could use his staff in a number of ways, all of which would provide overall value to the organization. Another test manager, Carol, took a different approach, focusing on a narrower scope of influence than Bill: the product development organization.

Carol

was recently hired to run a software test group of four people. Those four people attempt to test the product that sixteen developers are working on. Carol realized during her interview process that this organization had enormous technical product debt. Carol decided to investigate the implications of the technical debt. She talked to the other managers and listed two major consequences of the inadequate testing: developers were spending almost half their time on product support, and the last two product releases missed their ship dates by 100%. (Since they were working on 6-month releases, 6-month slips were disastrous.)

Carol went to her boss (who was also the development manager) with a proposal: “I have some ideas about how to meet our desired six-month release dates, and how to reduce the time the developers spend supporting the product.” That got her boss’s attention. She laid out a plan that included hiring a project manager, peer review of bug fixes, and hiring four more people with a variety of skills for the test group.

Carol chose to exercise her value *inside* the product development organization—more than just the test group, but not across organizational lines in the company. Carol is showing she has more than just “manage-the-testing” value to the organization.

Provide Value and Recognize What Others Want

To use influence, you provide value and help others see the significance of the problem. Then it’s time to look outward to see what other people want. What is valuable for you to provide?

You provide value to your organization in many ways. Sometimes, it’s your general managerial assets that others value: your credibility, fairness, track record, and management capabilities. In addition to your management capabilities, you can provide additional value by:

- Assessing risk, starting at the beginning of the project and continuing through to the end
- Identifying and solving problems that prevent testers and developers, all of Product Development from making progress
- Managing the test and measurement work (for both the product and the project) to assess the product state, report on it, and see if it needs improvement
- Helping make ship decisions

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

You are much more than just someone who manages or leads the test effort on a product. If you can see the significance of an issue, and can share that with the rest of the organization, you are incredibly valuable to your organization. One way to share significance of issues is to assess and report on risk.

Assessing Risk

Product testing and measurement is a primary vehicle for the test manager to help assess risk during the project. The risk isn't just limited to the *testing* organization, where a risk might be "We can't get the testing we'd planned to do done on time." The risk exists *across* the product development organization, and sometimes across the company. The three test managers above recognized that the risk to early shipment of inadequately tested product was higher-than-desired support costs, and less capability in development for the next release.

To provide value in risk assessment, it's a good idea to look beyond the boundaries of the test group to see who else is affected by creation of technical debt, or by slipping schedules. Technical debt, for example, affects developers' ability to adequately design the next round of changes (and creates an unstable base on which to implement those changes), and also has an impact on Support and Sales.

Identifying and solving problems

Here's a problem many test managers run into: the developers miss their deadlines, and your time in test is reduced. Before you even start testing you know there is a substantial risk that the product that will ship is not the product your organization had planned to ship. If you can identify that problem early to the project manager, you may be able to influence the solution.

One solution could be to ship the product as an early release, such as a Beta version, to give you more test time. Another solution could be to change how the developers fix the defects, or to target certain areas of the system for defect fixing and defer fixes in other areas. Another possibility would be to reduce the number or scope of the features to be shipped in this release. The solutions you develop will depend on why your company is shipping the product.

Managing the test and measurement work to assess product state

The testing results help all the decision makers evaluate risk. As the test manager, you look for information about the product in order to answer a number of questions:

- Did we build the product we intended to build? When you test the product against the requirements and the design, including functional, performance, and reliability testing, what do you find?
- How much of the product have we tested? Does it make sense to ask this question?
- How stable is the product at this point in the testing? I expect the product to be very unstable during the beginning and middle of the project. I expect it to become more stable as we approach the final milestones, especially the ship date. What happens if the product does not become more stable?
- How complete is our information? How good is the information we have now? How wise is it to make risk assessments based on this information?

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

You may have other product- or project-specific questions to add to your list. Choose questions that help you answer whether you've accomplished your goals and priorities. I tend to choose measures that help me know when we can ship the product.

Helping make ship decisions

"Can we ship this product yet?" If you can help your organization make that ship decision, you've satisfied many people's WIIFMs.

It is difficult for many of us to make product ship decisions. As test managers, some of us want to keep the product in test until they physically pull it away from us. As project managers or development managers, we may have bonuses riding on release dates (a foolish, but very common idea). But if we are responsible for the tech support work, we have even more motivation to find and fix more of the defects before the product ships.

No matter what our perspective is, we may think we know what the right ship decision is. It's not possible, however, for the test manager *alone* to make a ship decision without having negotiated and objectively defined release criteria with the rest of the project team. At the very least that criteria has to define what the organization is looking for from this product for this release, and may be an excellent way to identify leverage points for what other people want.. (For a brief discussion of how to create release criteria see Rothman and Lawrence.)

What do you want in return?

For the purposes of this discussion I'm going to assume you don't work in a "corrupt" organization, in which managers exploit their positions just because they can. Using influence on those people requires that you look at their personal positions in the organization and satisfy some of their selfishness. I find that placating those people gives me heartburn, so I choose not to work with them.

Instead, I'm assuming you work with reasonable, well-intentioned project and senior managers who Deming was talking about in Out of the Crisis-: "No matter how it looks, everyone is doing their best." They might not know how to use test managers, or they might be making some classic testing mistakes themselves (such as assuming you, the test manager, are somehow responsible for quality).

Your managers and colleagues aren't stupid. They know they need test managers, and they know they need the product tested. But they may not understand how to translate that abstract "knowing" into how they can use you more effectively. When you influence people, you help them realize what you can provide them. Now you can use your value to decide what *you* want in return from the organization.

When you help define the consequences of releasing a product with technical debt…or when you define and track release criteria…or when you have conversations about what the organization requires from the release…you've provided substantial value. You may get some well-deserved resources or support by *using* that value to illustrate to your managers:

- Why shipping a defect-laden product increases other groups' pain
- Why funding the test group's equipment requests will help the other managers know sooner when the product is ready to release

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

- Why shortening your test schedule will prevent you from collecting enough data to know whether to ship
- Why you should hire enough people to do the right amount of testing so everyone will know when to ship

You may want something *else* in return. That's fine. Whatever your goals are, just be clear on what you want to get from this two-way relationship.

When It's Not You

At times, when we attempt to provide this value to the organization, we feel like it's us against the rest of the world. That's not a good place to start when you want to influence people. Let's examine why you might feel that way:

- If you've somehow gotten yourself into the position as the person *responsible* for quality in the product. This, as Marick points out in Classic Testing Mistakes, is a classic testing mistake. If the people in your organization want to hold you personally responsible for the quality of the product, say, "Here's how I can **help you** make that quality happen in the product." It's not your job to uphold some arbitrary quality standard just because they say it's so. Work with the other people to define the value the test group provides to the rest of the company. If other people in the company have a quality standard they want upheld, the developers should uphold it. This might lead other people to say "That test manager just isn't a team player." But you can be the best sort of team player by asking the important questions: why the quality standard is important to the organization, what results they want to see, and what a successful release looks like.
- If you're the only one assessing product shipment risk and other situations where you, and you alone, can see the implications of shipping this product on the business. If you're the only one with the data, and you're the only one to make all the decisions, Murphy's Law says you're bound to make some bad decisions. Murphy's Law holds true at other times too, but here you have a single point of failure, which is much higher risk. You also may not be aware of the business decisions involved with this product release.
- Maybe you're just having a bad-manager day. That's okay. Wait to work on this problem until you feel more centered, more sure of what you have to offer.

Even when you've assessed your value and you think you've discovered the other people's WIIFMs, influence may not seem to work. If that's the case, perhaps it's because the other people—the project manager or senior managers—may not have considered *their* value or what they want. When faced with this unevenness of perceived value, you may want to offer to help them define their value, and in the process help clarify their WIIFMs. There's no simple solution when your colleagues don't know what they want. Although we'd like to think we can figure out a way to work with everyone, sometimes you just can't. Ultimately, you choose with whom you work.

Summary

As a test manager, one of your roles is to assess product state and report on it. You provide value by planning the product testing so that you can assess—along with the rest of the management

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

team—the risk of shipping the product. The more widely you disseminate the data about the product state, the more perceived value you will provide, and the more influence you can have.

As an influential test manager, you can

- Ask vital questions about the product and project, and assess the answers to those questions
- Facilitate a discussion about tradeoffs for the release
- Generate and test the release criteria, so you have an objective way to answer the question "When is it ready to ship?"

You don't *have* to feel out of control. The more you share your perception of the risks, the more you can influence what happens with your project, and the more successful you can make the organization you've invested in.

Acknowledgements

I thank the following reviewers for their helpful comments: Mark Druy, Elisabeth Hendrickson, Cem Kaner, Brian Lawrence, Brian Marick, Noel Nyman, Jeffery Payne, Dave Smith., Jerry Weinberg.

References

Cohen, Allen R. and David L. Bradford, *Influence without Authority*, John Wiley and Sons, New York, 1991.

Deming, Edward, *Out of the Crisis*, MIT Press, Cambridge, MA, 1986.

Gause, Don and Gerald M. Weinberg, *Exploring Requirements, Quality Before Design*, Dorset House, New York, 1989.

Kaner, Cem. *Black Box Software Testing*, a 3-day course taught by Cem Kaner, March 1999.

Marick, Brian. *Classic Testing Mistakes*. STAR 1997.

Rothman, Johanna and Brian Lawrence, *A Pragmatic Strategy for NOT Testing in the Dark*, Software Testing and Quality Engineering, Mar/Apr 1999.

Weinberg, Gerald M. *Quality Software Management, Vol. 1 Systems Thinking*, Dorset House, New York, 1992.

© 2000 Johanna Rothman, originally published in Software Testing and Quality Engineering, Vol. 2 #2

QW2000 Vendor Technical Paper 3V1

Michael R. Perdue
(Software Development Technologies)

New Automation Solution for Testing and Technical Reviews

BACK TO QW2000 PROGRAM

Key Points

- New architecture for test design and automation
- Automating technical reviews
- Create more cost effective and maintainable test suites

About the Author

Michael R. Perdue, executive vice president of Software Development Technologies (SDT), has over twenty-five years experience in software engineering and software quality with extensive experience in process design and improvement, organizational design for quality, quality costing and ROI, change management, assessment practices and pre & post release measurement systems.

Prior to joining SDT, Mike was the Chief Operations Officer of Integral Systems, Inc. a provider of mainframe and client/server HR/Payroll software systems. He was responsible for reversing several quarters of losses resulting in over 13 months of operating profits. He accomplished this by redesigning the organization to be customer focused, implementing effective and efficient processes for product development and customer support and building a management team that could lead the company to be more collaborative and focused. He started with Integral as Vice President of Product Engineering for the InPower subsidiary.

Prior to that he held positions as Vice President of Quality and Productivity, Director of Corporate Quality, Director of Quality Technology and Sr. Manager of Operations for such firms as Cadence Design Systems, Sun Microsystems and Motorola.

Mike has developed and led organizations consisting of over 200 technical and administrative persons and has held positions ranging from software developer and tester to mid level manager to executive officer of medium to large software product development firms.

Mike has developed and conducted seminars in quality management, metrics, process design and improvement, change management, team dynamics and project management. He has been a lead process architect for five new corporate software development processes in four different high tech companies.

Mike devotes most of his efforts at SDT to helping clients assess and improve their development and test process capabilities.

BACK TO QW2000 PROGRAM

SOFTWARE DEVELOPMENT SDT TECHNOLOGIES

Presents

Automating Test Design and Inspections

Michael R. Perdue

www.sdtcorp.com mperdue@sdtcorp.com

Agenda

- **Introduction to Software Development Technologies**
- **Technical Review and Inspection Overview**
- **What is SDT ReviewPro®?**
- **Results**
- **Test Design and Automation:**
- **TestFrame™ Architecture**

ReviewPro is a registered trademark of Software Development Technologies. TestFrame is a registered trademark of CMG.

© 2000 Software Development Technologies

Software Development Technologies

- Founded 1992, San Jose, CA
- CEO - Edward Kit
- Mission - Provide practical, proven, effective software quality products and services worldwide
- Training and Consulting for:
  - Software Testing, Test Planning and Design
  - Technical Reviews and Inspections
  - Test Automation
  - Test Assessments
- ReviewPro - Technical Review Management Application
- TestFrame - Integrated Test Design and Automation

© 2000 Software Development Technologies

SDT Training and Consulting Services

TRAINING SERVICES

- Public and Onsite Classes in:
  - Software Testing
  - Testing Tools
  - Technical Reviews and Inspections
- Course licensing available

CONSULTING SERVICES

- Expertise in all areas of software quality improvement - test, automation, inspections, assessments
- Senior consultants grounded in practical, hands-on approach

REVIEWS/INSPECTIONS

- Simple, Effective Process for technical reviews
- Web-based tool for managing Reviews
- Customized to your process

TESTFRAME METHODOLOGY

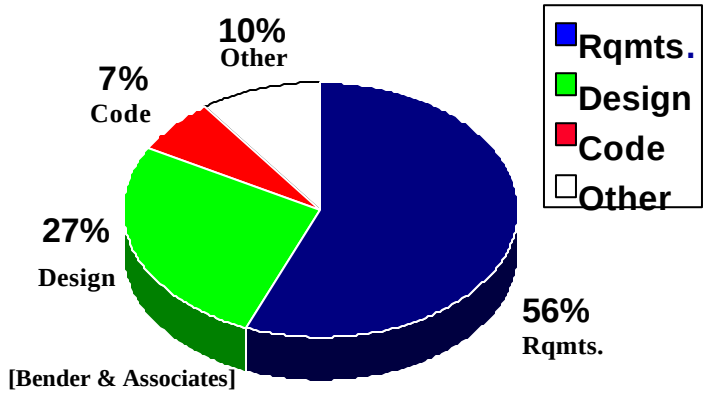
- Third generation test design and automation integration system
- More efficient use of resources
- Maintainable and cost-effective test suites that leverage the real strength of capture/playback tools

© 2000 Software Development Technologies

Agenda

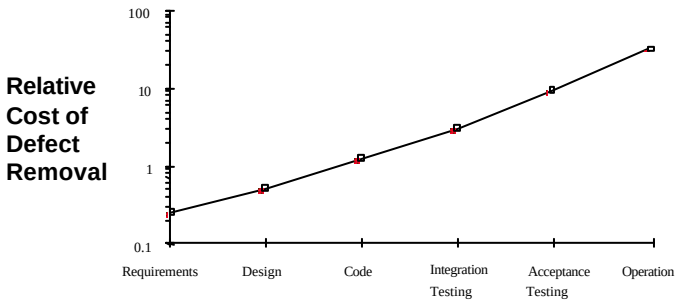
- Introduction to Software Development Technologies
- Technical Review and Inspection Overview
- What is SDT ReviewPro?
- Results
- Test Design and Automation:
- TestFrame Architecture

Defect Distribution



[Bender & Associates]

Defect Removal Costs Increase Radically During Software Life Cycle



The Lesson: Find Defects Early!

© 2000 Software Development Technologies

Benefits

Inspections provide a powerful way to:

- **Detect defects early in the development cycle**
- **Prevent the migration of defects to later phases**
- **Improve communication to the entire project team**
- **Provide additional perspectives of the work to the author**
- **Improve the quality and productivity of the development and test process**
- **Reduce cost and cycle time**
- **Reduce maintenance effort**

Review early and often

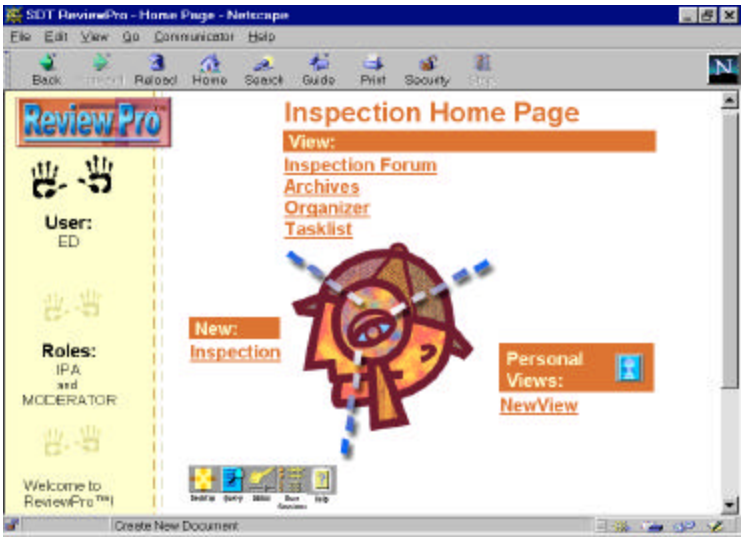
© 2000 Software Development Technologies

Agenda

- Introduction to Software Development Technologies
- Technical Review and Inspection Overview
- What is SDT ReviewPro?
- Results
- Test Design and Automation:
- TestFrame Architecture

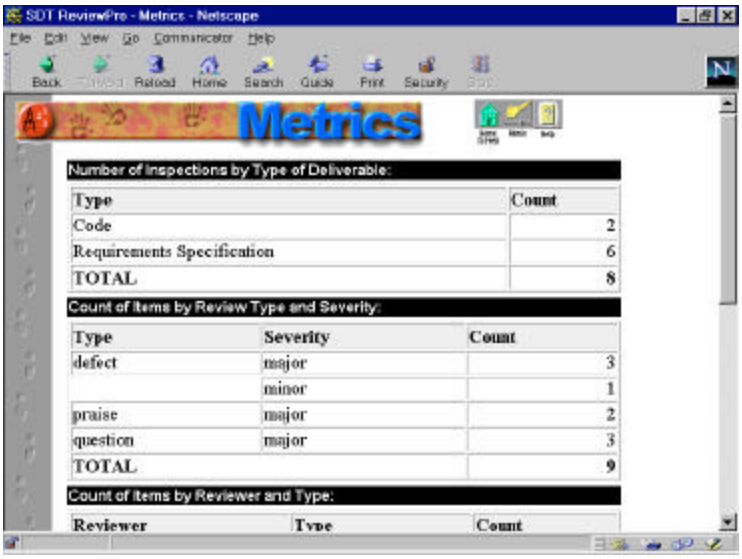
SDT ReviewPro Highlights

- Brings automation to the most powerful defect detection and removal process - Technical Reviews and Inspections
- Web-based, collaboration application
- Open architecture:
  - Integrates with existing IT infrastructure
  - Accommodates any Inspection process
- Project management for geographically-distributed teams
- Data collection and metrics reporting capabilities
- 1.0 FCS (first customer ship) in June 1998; shipping and installed at Sun, US West, Lucent, US Navy, Cisco, etc.
- Only known commercially-available product today



From the ReviewPro Home Page, the user can create a new Inspection or go directly into an Active or Archived Forum. This screen is customizable to reflect your process name, page title, etc.

Indented log entries show conversations by various, possibly distributed, team members. Most issues resolved this way.



A variety of metrics is tallied from information in the ODBC-compliant data store.

Agenda

- **Introduction to Software Development Technologies**
- **Technical Review and Inspection Overview**
- **What is SDT TRIP?**
- **What is SDT ReviewPro?**
- **Results**
- **Test Design and Automation:**
- **TestFrame Architecture**

Inspection Savings Summary

Reasonable expectations:

- **Net timescale reductions: 25%**
- **% of all defects eliminated before test execution starts: 60 - 80%**
- **Reduction in test execution costs: 5X**
- **Number of hours of downstream correction effort saved for each major defect found: 9**
- **Total systems maintenance costs reduction: 5-10 to 1**

Upstream effort creates downstream benefits.

Invest early or Suffer later!

© 2000 Software Development Technologies

Quotes from Industry Leaders

" . . . projects with a full system of reviews report a 10 times reduction in the number of errors reaching each stage of testing. The concomitant cost reduction for testing efforts runs between 50 and 80 percent, even when the cost of reviewing is added to testing costs."

Gerald Weinberg, Daniel Freedman

© 2000 Software Development Technologies

ReviewPro Customer Quotes

"ReviewPro is the most interesting piece of software for giving support to the Inspection process I have seen. I am recommending that my clients evaluate it. Too many of my clients have made multiple versions of such software within the same corporation. This should no longer be necessary. ReviewPro will give us the tools we need to focus on getting inspection results, not toolmaking." **Tom Gilb, Author of *Software Inspection***

"The biggest complaint we' d had with our current Inspection method/process was getting data into the system because of the amount of paper required; this problem will be solved with ReviewPro. Its collaborative feature allows defects to be documented and visible before our meeting so productivity is greatly improved."

Edward Weller, Fellow, Bull Worldwide Information Systems

Agenda

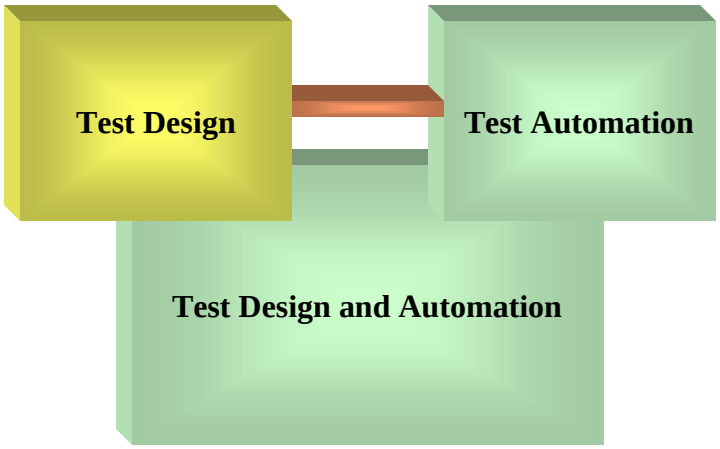
- **Introduction to Software Development Technologies**
- **Technical Review and Inspection Overview**
- **What is SDT TRIP?**
- **What is SDT ReviewPro?**
- **Results**
- **Test Design and Automation:**
- **TestFrame Architecture**

Test Design and Automation: Serious Problems

- Lack of an effective test automation architecture
- Lack of required competencies:
  - Test Design
  - Technical Automation
  - Application
- Using capture/playback at the wrong time
- Lack of sufficient resources:
  - Not enough time for automation implementation
  - Ratio of testers to developers
  - Dedicated capital equipment

© 2000 Software Development Technologies

Recommendation: Separate and Bridge



© 2000 Software Development Technologies

Recommendation: Separate Functional and Technical Tasks

Test Cases
(Action Words in a spreadsheet or db)

A	B	C	D
. . .			
log in	iris rose	XXXXX	
enter rev log	Proj1	Proj1 Reqts	Error
check rev log	Proj1	Proj1 Reqts	Error
log out	iris rose		
. . .			

Functional

Technical

automation system
(test tool)

```
case action
   "log in": ...
   "enter rev log": ...
   "check rev log": ...
end
```


Agenda

- **Introduction to Software Development Technologies**
- **Technical Review and Inspection Overview**
- **What is SDT TRIP?**
- **What is SDT ReviewPro?**
- **Results**
- **Test Design and Automation:**
- **TestFrame Architecture**

Proven Test Architecture

- TestFrame is an example of a proven test framework
- Created in 1994 by CMG, this approach has been used successfully by hundreds of projects
- SDT successfully used TestFrame to test ReviewPro, a Web-based Collaborative Application that brings automation to Technical Reviews and Inspections
- SDT and CMG are working together to evolve TestFrame
- TestFrame has been used for On-line, Web, Batch, API, Embedded for function, system, acceptance test
- Reference [Buwalda, 1998]

© 2000 Software Development Technologies

TestFrame Key Roles and Responsibilities

- Test Architect -- Creates the Testing Framework, i.e., overall approach to verification and validation, including an integrated approach to test process and automation
- Test Planner/Manager -- Provides test planning, schedule, scope, resources, etc.
- Test Designer / Analyst -- Creates and documents test design and test cases
- Test Automation Engineer / Navigator -- Creates test case automation processing capability
- Test Executor -- Runs and evaluates tests

© 2000 Software Development Technologies

Action Words: Key to the Bridge

Action Words:

- **Establish a high-level application usage abstraction**
- **Standardize application actions**
- **Enable communication between Test Design and Test Case Processor**

Next Step:

- **Translate test cases into Action Word spreadsheets**

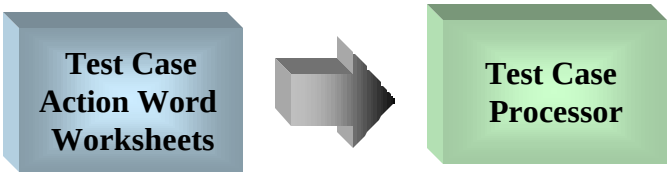
© 2000 Software Development Technologies

Action Word Cluster / Spreadsheet

	A	B	C	D	E
1	cluster	RFPL			
2	sheet	Test the Reviewer's Log form			
3	version	2.2			
4	date	2/12/00			
5	author	SP			
6	testcase	RFPL1	Enter log		
7		Expected Result:	Log saved		
9		*button*	*username*	*password*	
10	log in	SDT ReviewPro	iris rose	XXXXX	
11					
12		*button*	*location*	*type of item*	*description*
13	enter rev log	topSaveLog	page 5	defect	Users cannot enter date
14					
15		*button*	*location*	*type of item*	*description*
16	check rev log	topSaveLog	page 5	defect	Users cannot enter date
17					
18		*button*	*username*		
19	log out		iris rose		
20					

Re-usable Framework Components

Functional Test Development **Technical Test Execution**



A	B	C	D
. . .			
log in	iris rose	XXXXX	
enter rev log	Proj1	Proj1 Reqts	pg 5
check rev log	Proj1	Proj1 Reqts	pg 5
log out	iris rose		
. . .			

automation system

```
case action
   "log in": ...
   "enter rev log": ...
   "check rev log": …
   "log out" ...
end
```

© 2000 Software Development Technologies

The End

© 2000 Software Development Technologies

QW2000 Paper 3T1

Mr. Robert Bauer & Mr. Russell F. Ingram (Levetate Design Systems)

Building a Parallel Test Environment

BACK TO QW2000 PROGRAM

Presentation Abstract

For the past two years, we have been developing and enhancing a test strategy called early regression testing. In this strategy, we provide development teams with automated tests that enable them to run complete regression testing prior to feature integration - at our organization multiple development teams simultaneously enhance the core software product, so it is not uncommon to have several teams making concurrent changes to the same source base.

Prior to the introduction of early regression testing, development teams would run their own unit tests, integrate the various features and turn the software over to test teams. One problem with this approach is that when a failure is observed during testing, development has to figure out which team is responsible for fixing the problem. Often the problem is assigned to one team only to be transferred to another team when the first team figures out that the problem is not in their code. Sometimes a problem is passed back and forth among several development teams.

Early regression testing, on the other hand, enables the developers to identify and fix problems associated with their code prior to integration. Results of this approach were significant: More than 70% of all failures detected by the regression suites were discovered prior to feature integration. By shaving nearly 2 months off of the schedule, we helped to significantly reduce the delay of an already delayed product.

Early regression testing makes a lot of sense, but to make it successful requires a powerful infrastructure:

(1) Tests must be automated - Development teams are under a lot of pressure. If it takes a lot of effort to locate, run, and analyze tests, developers will devise their own tests. These unit tests will help them with whatever specific enhancement they are making; however, these unit tests will never be as encompassing as the regression suite through which the software is certified.

(2) Results analysis must be automated - Automating the running of the test and leaving the test engineer with hundred's of pages of diff'd output to analyze borders on the inhumane. We developed a "rule-based" algorithm that analyzed the differences (between the control and output) to determine whether the differences constituted failures or were acceptable.

(3) Present results in a useful form and make it easy for the test engineer to do their job - Our initial release presented the test engineer with lists of tests: Those that passed, those that failed, and those

that didn't run to completion. We then added an X-windows based user interface whose main feature was that with a single button press, the test engineer could bring up a side-by-side display of the control and output files (aka windiff). We provided a "button" so the engineer could sequence from one "error" to the next (the engineer sees the certified output in the control file, the actual output in the results file). For differences, the tester also sees the "rule" used to determine that the difference was not a failure.

(4) Tests and control files must be maintained under revision control. Concurrent feature development teams require the capability to independently create and enhance feature tests. These feature tests need to be merged into the overall regression suites. We often refer to the regression test accounting equation:

New Regression Suite = Old Regression Suite + New Features - Obsolete Features

Thus, as each feature team demonstrates that they can pass the the Old Regression Suite with the Obsolete Features removed and their New Feature tests, we "merge" the various regression suites enhanced and modified by the feature test teams into a New Regression Suite that is initially used for integration testing. Later, after product release, the New Regression suite is used for maintenance and emergency release testing. Also, as work begins on the "next" release, the new regression suite becomes the old regression suite as the cycle is repeated.

As the infrastructure for early regression testing proceeded, it was clear that a significant problem remained: As we automated away much of the tedious and error-prone manual efforts, execution time of the tests began to dominate the overall cycle-time. Also, roughly at the same time, the next generation hardware was being released that offered a significant improvement in the database throughput capacity. Although we employed such optimizations as overlapping test execution with results comparison - achieving a 33% reduction in cycle-time, we knew that the next generation of database machines would be under utilized in the present test environment.

About the Author

Robert T. Bauer was the technical lead and architect for the PTE environment. He has a MS in Computer Science. Robert is presently a principal engineer with Levetate Design Systems where as a member of the formal reasoning group he is responsible for developing a theorem prover for use in formal verification of software and hardware.

Russell F. Ingram is the manager of the development group that created the PTE. He also designed the architecture for the test lab and led the technical system administration efforts. Russell has a BS in Electrical Engineering and is presently involved in coordinating off-shore development as well as building a new corporate computing environment for NCR.

BACK TO QW2000 PROGRAM

Building a Parallel Test Environment

Problems with Manual Testing

1. Long Test Cycle Time

2. Required Experts to Locate, Run, & Evaluate test results.

3. Millions of lines of test code and no commercial test tools.

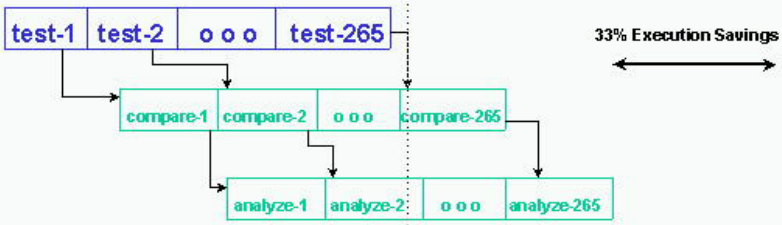
4. Low machine utilization

5. Manual review of compare output error prone.

History

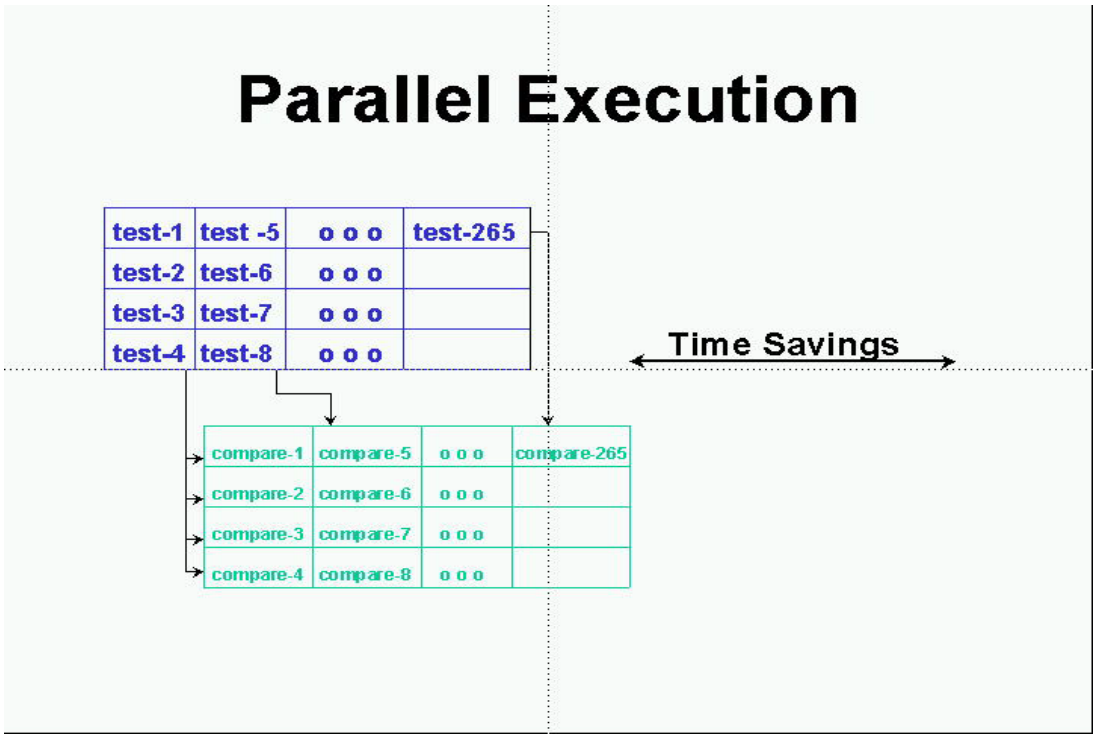
Original test suites

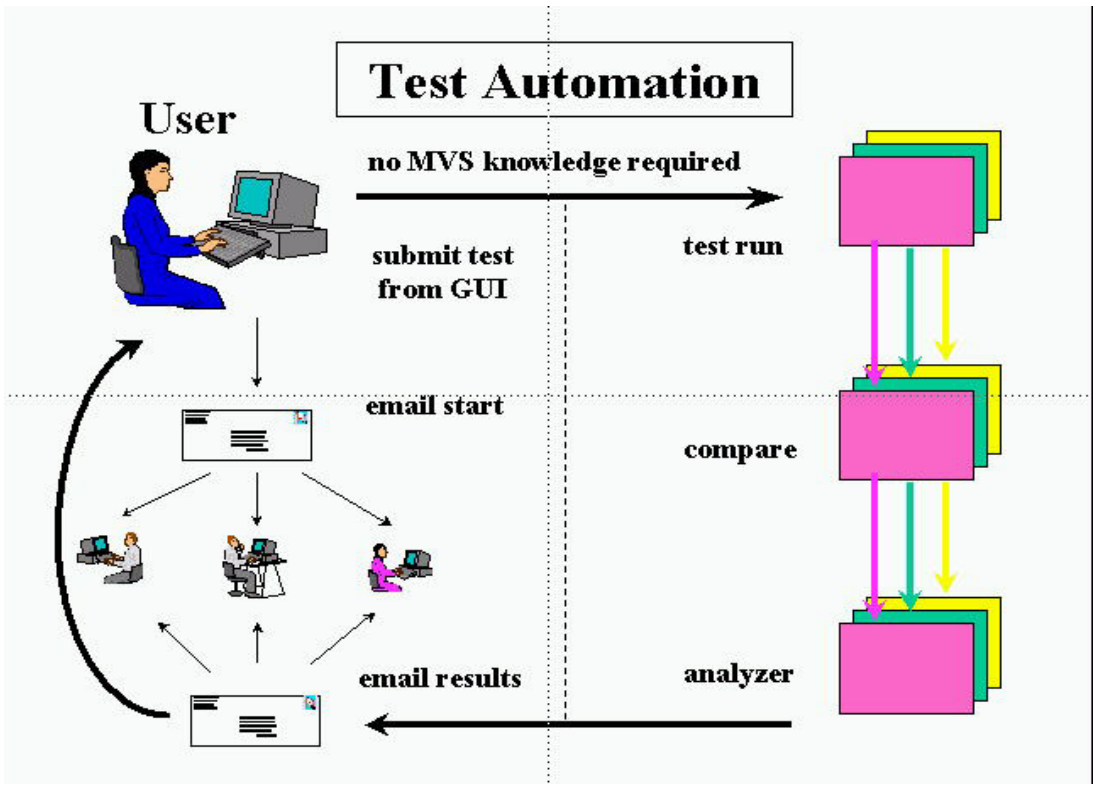
Automated test suites

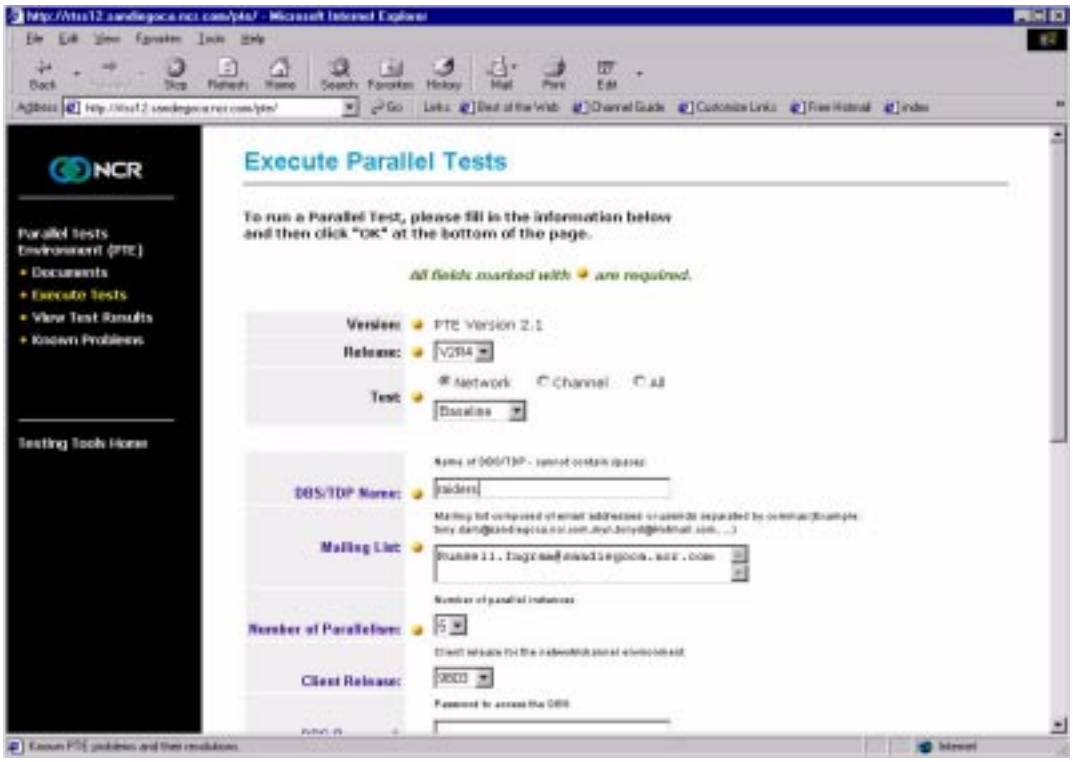


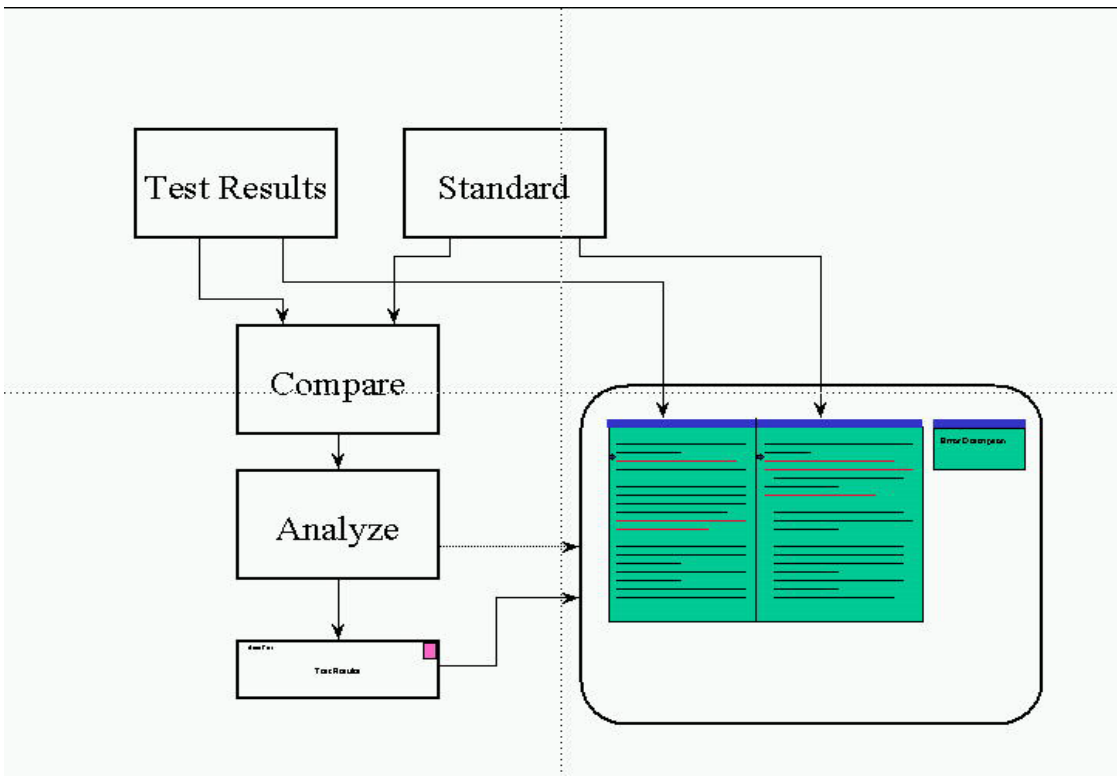
Improvements Request

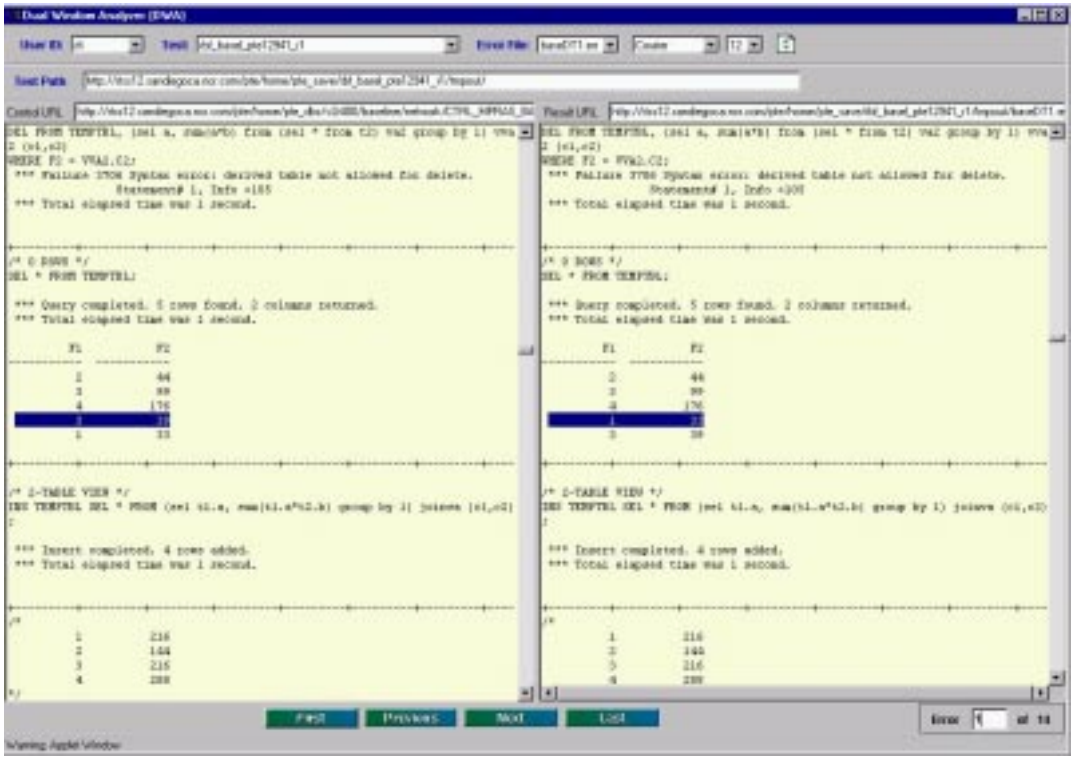
- Restartability
- Unique Job Names
- Unique User Names
- Easier to look at Results
- Report Generation

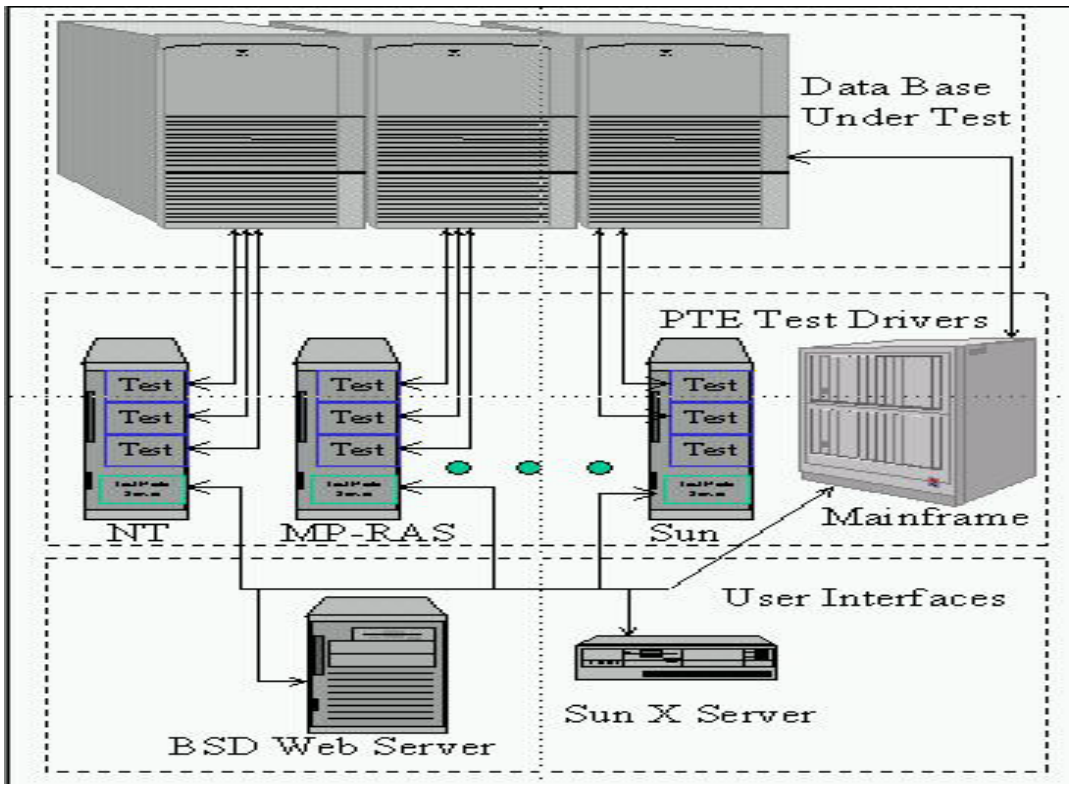

Parallel Execution
test-1
test -5
o o o
test-265
test-2
test-6
o o o
test-3
test-7
o o o
test-4
test-8
o o o
Time Savings
compare-1
compare-5
o o o
compare-265
compare-2
compare-6
o o o
compare-3
compare-7
o o o
compare-4
compare-8
o o o

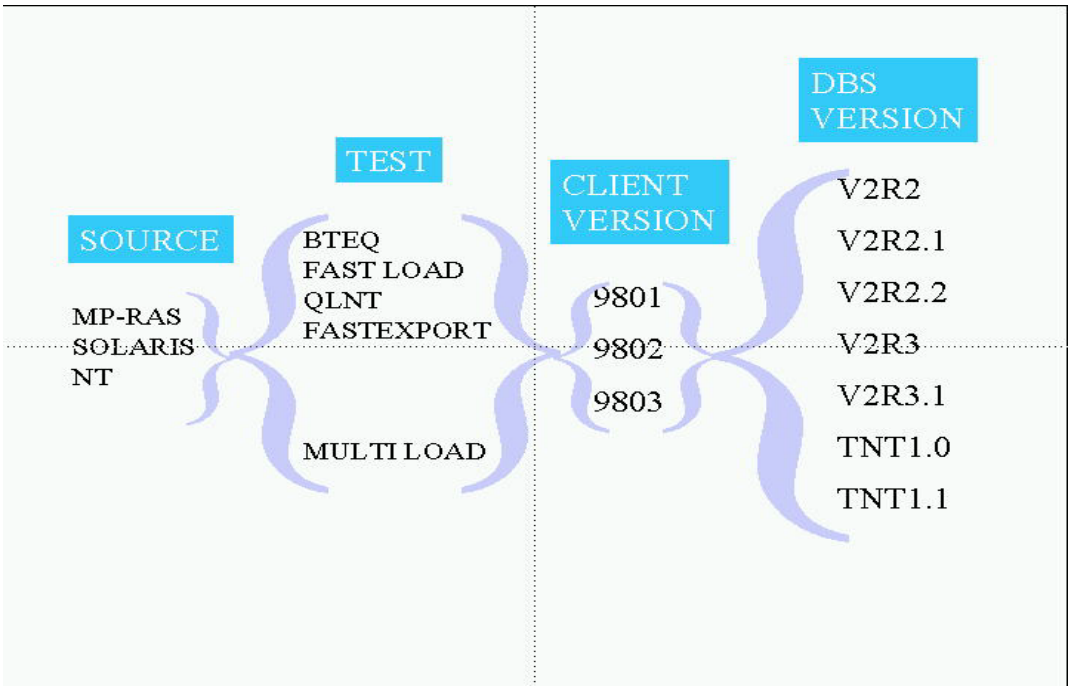

Test Automation
User
no MVS knowledge required
submit test
from GUI
test run
email start
compare
email results
analyzer


NCR
Execute Parallel Tests
To run a Parallel Test, please fill in the information below and then click "OK" at the bottom of the page.
Parallel Tests Environment (PTE)
Documents
Execute Tests
View Test Results
Known Problems
Testing Tools Home
Version:
PTE Version 2.1
Release:
Test:
Network
Channel
All
Baseline
DBS/TDP Name:
Mailing List:
Number of Parallelism:
Client Release:


Test Results
Standard
Compare
Analyze
TestRoute




Data Base
Under Test
PTE Test Drivers
Test
Test
Test
Test
Test
Test
Test
Test
Test
NT
MP-RAS
Sun
Mainframe
User Interfaces
Sun X Server
BSD Web Server


SOURCE
MP-RAS
SOLARIS
NT
TEST
BTEQ
FAST LOAD
QLNT
FASTEXPORT
MULTI LOAD
CLIENT VERSION
9801
9802
9803
DBS VERSION
V2R2
V2R2.1
V2R2.2
V2R3
V2R3.1
TNT1.0
TNT1.1

QW2K Paper 3A1

Mr. David Carman
(Telcordia Technologies)

Measuring Test Effectiveness: The Use and Misuse of Test Coverage

BACK TO QW2000 PROGRAM

Key Points

- Runctional coverage
- Code coverage
- Software measurement

Presentation Abstract

Quality assurance is a never ending journey to insure that the each and every customer will be delighted with a usable and reliable product. In-process metrics are an important element of quality assurance and they provide a cornerstone of a defined and managed software development process.

There are many compelling reasons for measuring not only the quality of software products, but also the quality improvement impact of each phase of the development process. This talk will address practical test coverage methods for measuring the effectiveness of your quality assurance process. Experiences from five different case studies will be shared.

About the Author

David W. Carman is the Principal Automation Architect for Software Quality Assurance within Telcordia Technologies. Mr. Carman has ten years of quality assurance experience in telecommunications network and operations support systems testing. He has presented and authored papers at international conferences including Software Quality Week, STAR, EuroSTAR and ISSRE. He has applied theories to practice in the areas of automatic test case generation, measuring test effectiveness and software reliability engineering.

BACK TO QW2000 PROGRAM

Telcordia Technologies

Performance from Experience

Measuring Test Effectiveness
The Use and Misuse of Test Coverage

David W. Carman
Principal Automation Architect
Software Quality Assurance
dcarman@telcordia.com

An SAIC Company

**Copyright 2000 Telcordia Technologies.
Used by permission only.**

Measurement of Test Effectiveness

- Q: Why should you measure effectiveness?
- A: Maximize resources and minimize cost:
  - Which test cases are needed?
  - Which order should they be executed?
  - When should you release the product?
- Q: How do you measure effectiveness?
- A: Some popular techniques:
  - Bugs reported by the customer
  - Defect removal efficiency and defect density
  - Categorizing defects found by phase (ODC)
  - *Test Coverage*

[Are defect counts an effective measure of effectiveness?]

Misconceptions* of Test Coverage

- Test coverage is only code coverage.
- Functional coverage is only requirements traceability.
- Test coverage is only a number.
- 100% test coverage is good and low test coverage is bad.

* i.e. Impressions that I try to dispel.

Telcordia Technologies
Performance from Experience

Definitions: Test Coverage

- Comparison of what *can* (or should) be tested with what *has been* (or will be) tested.
- Other useful definitions:
  - "Every test technique implies a coverage metric that measures the ***extent*** to which that technique has been employed", Boris Beizer from Black-Box Testing.
  - "Any metric of ***completeness*** with respect to a test selection criteria", Dr. Shmuel Ur (IBM Research) at STAR 98.
  - "Coverage criteria specify a class of paths she should test. In contrast to absolutely complete path testing, these criteria define ***achievable*** (if possibly expensive) amounts of testing" from Testing Computer Software, 2nd ed.

Telcordia Technologies
Performance from Experience

Types of Test Coverage

- Structural (Program-based)

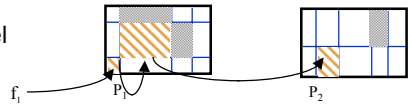
  Determined by the system architect/designer. Tangible units which work together to implement functionality. Well-defined and bounded. Automated collection.

  – Code blocks / functions
  – Component interfaces
  – Control-flow / data-flow

- Functional (Feature-based)

  Determined by the customer/system engineer/tester. Sometimes intangible units. Not well-defined and almost infinite. Manual collection.

  – Requirements
  – Use-cases
  – Abstract behavioral model



Telcordia Technologies
Performance from Experience

Why is test coverage important?

- Fault finding:
  – Find untested areas of the system.
- Measuring quality:
  – Calibrate test quality with quality of system under test.
- Test case minimization / selection:
  – Reduce duplicate test cases.
- Change management:
  – Define regression test suite according to code changes or changes in feature content.
- Education (training & mentoring):
  – Improve mechanism for communicating testing approach.

Telcordia Technologies
Performance from Experience

Novel Uses of Test Coverage

- Customer defect feedback loop
  - Find weaknesses in test strategy and improve coverage model.
- Cost allocation:
  - Use test coverage data to negotiate and allocate testing budget.
- Risk management:
  - Define what is not tested against cost and customer usage.
  - Explain test coverage to a customer when describing the risks of releasing a product before final certification.
- Documentation testing:
  - Improve completeness of the specification and user guide.
  - Facilitates the review process when using a "checklist" approach.
- Test case design:
  - Formalize test design process to assist automatic test generation.

Telcordia Technologies
Performance from Experience

Test Coverage Case Studies

- Measurement by Instrumentation of Code:
  - ***Trial 1:*** A large mainframe product, 1.3 million lines of code, with a substantial regression package, 50,000 test cases, and low rate of customer failures.
  - *Result:* There were many operational issues which made code coverage instrumentation difficult, but somewhat beneficial.
  - ***Trial 2:*** A distributed client/server product with a rapidly changing set of features with many customer enhancement requests. The code changes dramatically from release to release.
  - *Result:* Coverage was measured to be high, but there were several additional functional areas, not tested. There were issues which made analyzing this data difficult, but test quality was improved.
  - ***Trial 3:*** A traditional 3-tier client/server product with a stable set of features with good customer satisfaction.
  - *Result:* Coverage was measured to be high with few duplicate test cases. The process was very easy to introduce and it was cost effective.

Telcordia Technologies
Performance from Experience

Test Coverage Case Studies (continued)

- Measurement by Inspection of Structure:
  - ***Trial 4:*** A legacy mainframe product with limited documentation and a large regression package.
  - *Result:* Coverage was used to document why several areas were not tested. Coverage was used successfully for economic justification.
- Measurement by Inspection of Functionality:
  - ***Trial 5:*** An emerging client/server product with up-to-date specifications and documented designs.
  - *Result:* Coverage was evaluated based on documented work-flows. A few additional test cases were created based on this analysis.

Telcordia Technologies
Performance from Experience

Comparing Test Coverage

Which project made the most improvement?

Project DB

release 7.0 = 90%
release 7.1 = 100%
} +10%

Project PW

release 6.0 = 67%
release 8.5 = 90%
} +23%

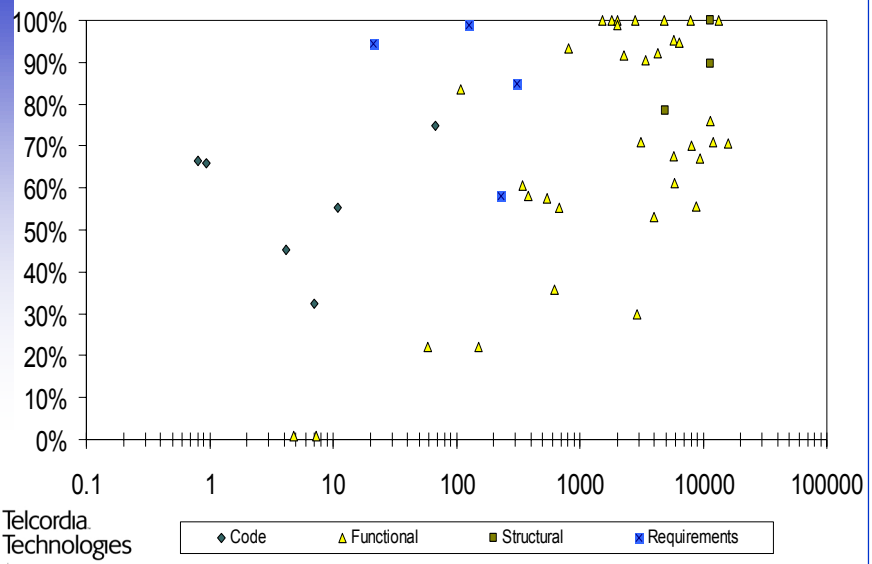
Telcordia Technologies
Performance from Experience

Defining Normalized Test Coverage

- Problem: How do you make test coverage results comparable across releases and projects?
- Simple approach:
  - Calculate the size of each coverage item.
- Granularity of coverage items based on:
  - System size
  - Scope of the coverage analysis
  - Total number of items counted

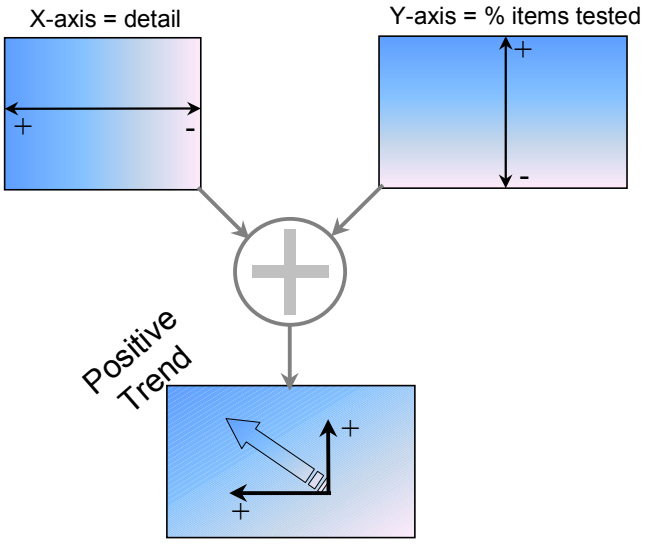
Telcordia Technologies
Performance from Experience

Normalized Test Coverage Results



Telcordia Technologies
Performance from Experience

Interpreting Normalized Test Coverage

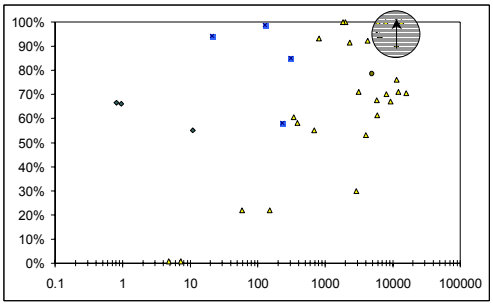


Telcordia Technologies
Performance from Experience

Changes in Test Coverage

Project DB	R7.0	R7.1
Coverage	90%	100%
System Size	4,927K	4,979K
Scope	20%	20%
Total	86	88
Tested	77	88
Units	function	function

Explanation:
Extra test cases created to fill uncovered holes.

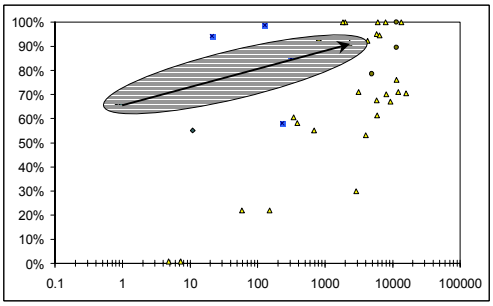


Telcordia Technologies
Performance from Experience

Changes in Test Coverage

Project PW	R6.0	R8.5
Coverage	67%	90%
System Size	303K	352K
Scope	5%	100%
Total	18,527	104
Tested	12,334	94
Units	blocks	function

Explanation:
Changed scope and method from detailed, localized approach to generalized, global approach.



Telcordia Technologies
Performance from Experience

Observations

- Numerical coverage results depend upon many implicit assumptions that can make comparisons misleading.
- White-box code coverage tools can improve the quality of black-box system testing. But, code coverage technology has limitations that prevent the collection of test coverage data for complex systems.
- Code coverage is rigorous, but not always useful.
- Functional coverage may not be objective and repeatable, but it can be very beneficial, if used properly.

Telcordia Technologies
Performance from Experience

Conclusions

- Measuring test effectiveness is key for improving and optimizing the overall testing process.
- Test coverage is a fundamental approach for evaluating the effectiveness of testing.
- Using multiple forms of test coverage together can yield stronger results.
  (e.g. requirements traceability, plus coverage of structural components)

Telcordia Technologies
Performance from Experience

QW2000 Paper 3W1

Mr. Alberto Savoia
(Velogic Inc.)

The Science of Website Load Testing

BACK TO QW2000 PROGRAM

Key Points

- Introduction and description to the key variables and metrics for load testing websites.
- Innovate methodology for developing and validating highly realistic load scenarios using the concept of a WebSite Usage Signature.
- Practical tips, advice, and hard-learned lessons on website load testing.

Presentation Abstract

The concept and practice of software load testing has been around for many years, but the advent and exponential growth of the Internet has created a situation that has taken the magnitude, complexity, and importance of load testing to a whole new level. In this paper we present the overview of a methodology for approaching the very challenging task of load testing Internet websites in a rigorous and systematic manner, in order to collect, realistic and reliable data about a websiteÆs capacity and scalability. In the first part of the paper, we introduce, define, and explain the key variables and metrics that are used to understand, analyze, and generate highly realistic load tests for Internet websites.

In the second part, we introduce and explain the concept of Website Usage Signature (WUS) and use it to describe the process for developing and validating load testing scripts and scenarios that accurately match the way a website is used and navigated by real users. In the final section, we address some of the practical problems and concerns associated with the actual execution of website load tests.

About the Author

Alberto Savoia is founder and CTO of Velogic Inc., a professional services company that specializes in website load testing. In his 15-year career he has been consistently and passionately committed to improve the state of the art in software testing through the use of formal methods and automation, and has been granted several software patents, including 2 US patents in the area of test automation. Prior to Velogic, Mr. Savoia was founder and General Manager of SunTest, a business unit of Sun Microsystems that developed and marketed award-winning Java testing tools for API, GUI, and load testing.

Before SunTest, Mr. Savoia, was Director of Software Research at Sun Microsystems Laboratories

where, among other projects, he started and led the ADL (Assertion Definition Language) project, a highly successful, 3-year, $4M, joint international research effort in automated test generation for object oriented systems.

BACK TO QW2000 PROGRAM

Velogic
The eLoadTesting™ Company

The Science of Website Load Testing

Alberto Savoia
Chief Technical Officer
Velogic Inc.

© 2000 Velogic Inc.

Velogic
The eLoadTesting™ Company

Outline

- How website load testing differs from more traditional enterprise load testing
- Why website load testing is absolutely critical for eBusiness success
- Why realism and accuracy are critical to load testing
- Why most website load tests are wildly inaccurate
- The key principles for developing highly accurate, realistic, and useful website load tests

© 2000 Velogic Inc.

Comparison of Website to Traditional Enterprise Load Testing

Key Factors	Traditional Enterprise Load Testing	Website Load Testing
Transaction volumes	Predictable and limited	Unpredictable and potentially unlimited
User behavior and variables	Predictable and controllable	Unpredictable and uncontrollable
System variables	LAN, centralized HW and SW	Plus firewalls, routers, hosting company, caching systems, …
Risk	Failures are only noticed internally	Failures highly visible

© 2000 Velogic Inc.

Why Websites Need Load Testing

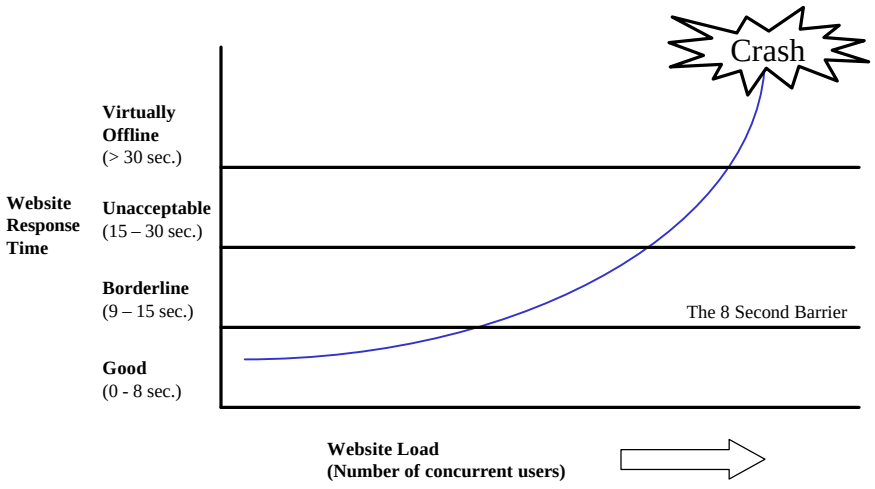
The Fundamental Theorem of Website Performance and Availability

Website Performance And Uptime → eUser Satisfaction → Revenue Profits Market Cap. Brand

Load testing helps you understand, anticipate, and pre-empt performance and uptime problems that can lead to **major eBusiness disasters** and/or **chronic *minor* losses**.

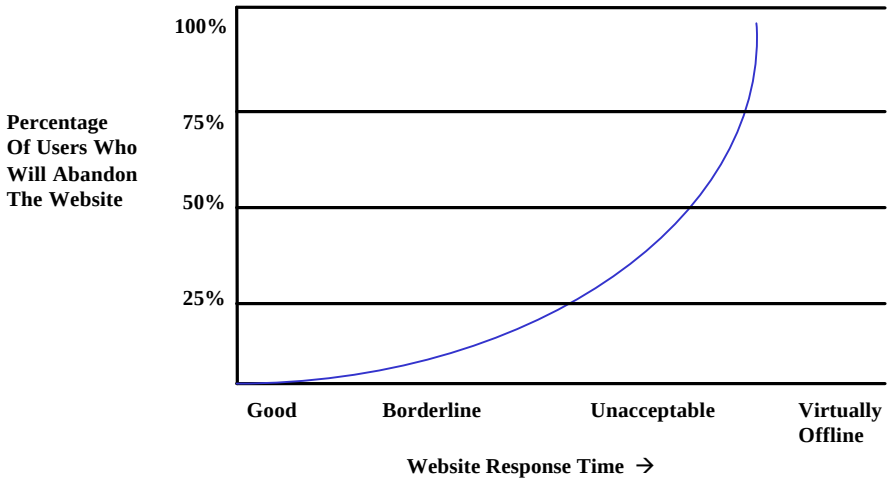
© 2000 Velogic Inc.

Website Load vs. Website Response Time



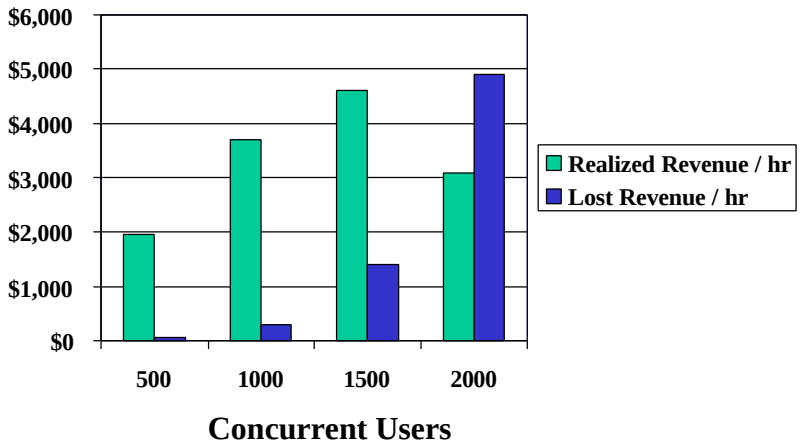
© 2000 Velogic Inc.

User Abandonment vs. Website Response Time



© 2000 Velogic Inc.

The Business Impact of Load



© 2000 Velogic Inc.

Realism and Accuracy Are Critically Important

- Realism
  - The simulated load must closely match the real load
- Accuracy
  - The measurements taken during the test must provide sufficient detail and granularity
- An unrealistic or inaccurate load test can be dangerously misleading:
  - Overestimating your capacity can lead to problems when the real load hits
  - Underestimating your capacity can cause unnecessary delays and cost additional $ for unnecessary HW and resources

© 2000 Velogic Inc.

How Realistic and Accurate Do You Really Have to Be?

- Zona Research experiment*
  - Log files showed that a company's 40Kb (~8 seconds to load) home page had a 30% abandonment rate.
  - The company tweaked the home page to 34Kb (~7 seconds to load).
  - Abandonment rate fell immediately from 30% to 6-8%.
  - All for one (1) tiny second difference in load time.
- Conclusion:
  - Most test scripts are patient, most real users are not.
  - Your load testing measurements should be as accurate as possible. Every second counts!

* "The Economic Impact of Unacceptable Web-Site Download Speeds" , Zona Research Inc. April 1999

© 2000 Velogic Inc.

The Science and Art of Realistic and Accurate Website Load Testing

- Aim for 80% "science" and 20% "art"
- Systematically capture and standardize load parameters from real world usage
  - Identify and track the key load metrics
  - Web site Usage Signature (WUS)
- Develop a realistic and accurate load generation model
  - Online behavior profiling
  - User demographics

© 2000 Velogic Inc.

Key Server-Side Load Metrics

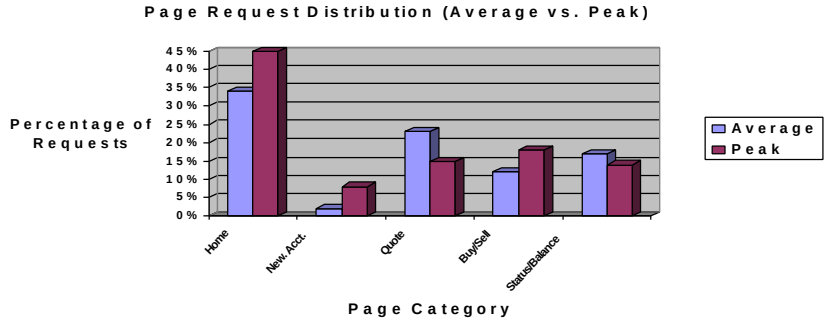
- Examples of server-side website usage metrics
  - Generic
    - Average pages per session
    - Average hits per page
    - Average session duration
    - …
  - Custom
    - Purchases to sessions ratio
    - Stock quotes per sessions
    - Database searches per session
    - …

© 2000 Velogic Inc.

Sample WUS

Average and Peak Load WUS

Metric	Avg. Load Metrics	Peak Load Metrics	Comments
Sessions per hour	10,340	105,232	The average session duration, page views, and hits are all considerably smaller during peak loads.
Page view per hour	97,200	456,708	
Hits per hour	602,640	1,806,343	
Average session duration	6:36	3:23	
Average page views per session	7.20	4.34	Most notably, a 10x increase in number of users caused only a 3x increase in hits per hour.
Average hits per session	58.28	17.16	



© 2000 Velogic Inc.

Gathering Data for the WUS

(Log Analyzers – Use With Care)

Whatever log analysis tool and metrics you use, make sure you use them consistently. The lack of standardization makes log analysis data non-portable.

	WebTrends	Analog	WebAlizer
Total Hits	1,667,256	1,886,952	1,920,405
Total Hits (Successful)	1,651,522	1,869,464	N/A
Total Hits (Failed)	15,734	17,488	N/A
Total Page Views	364,303	112,154	232,931
Total Kbytes Served	5,993,150	6,930,039	6,934,837
Avg. Kbytes served per page	16.45	61.79	29.77
Avg. Hits (Successful) Per Page	4.53	16.67	N/A
Total User Sessions	16,136	N/A	36,612
Avg. Session Length (minutes)	19:17	N/A	N/A
Avg. Pages Per Session	22.58	N/A	6.36

© 2000 Velogic Inc.

Creating Realistic Loads

Load testing realism is achieved by creating scripts and scenarios that take into account a number of key variables about website users and their equipment/location/configuration.

Online Behavior Variables

Read/write/think time
Latency tolerance
Tenacity
Familiarity with site
…

User Demographic Variables

Geographic location
Connection speed
Access time
…

© 2000 Velogic Inc.

Applying Behavior Variables

Velogic
The eLoadTesting™ Company

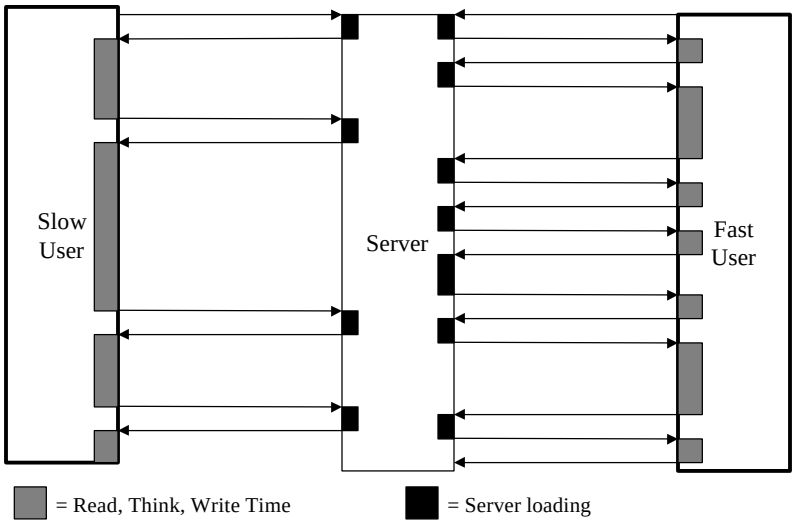
Latency Tolerance Simulation			
Latency Tolerance	**% of Users**	**Page Response Threshold (sec)**	**Effect on Script**
Low	25%	8	Terminate script if 2 consecutive pages exceed the page response threshold
Medium	50%	16	
High	25%	24	

Familiarity Simulation			
Familiarity	**% of Users**	**R/W/T Multi-plier**	**Effect on Script**
First-time users	35%	2.0	The read/write/think time is adjusted for each type using the specified multiplier.
Returning users	55%	1.0	
Experienced users	10%	0.5	

© 2000 Velogic Inc.

User Speed Affects Load

Velogic
The eLoadTesting™ Company

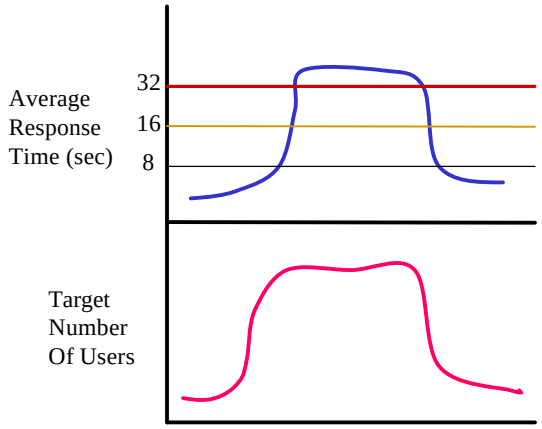


© 2000 Velogic Inc.

User Tolerance Affects Loads

Velogic
The eLoadTesting™ Company

Typical load test results:



This is not a realistic load situation.

What does it really tell you?

It tells you what the load would be if you had incredibly patient users willing to wait a VERY long time to get their pages.

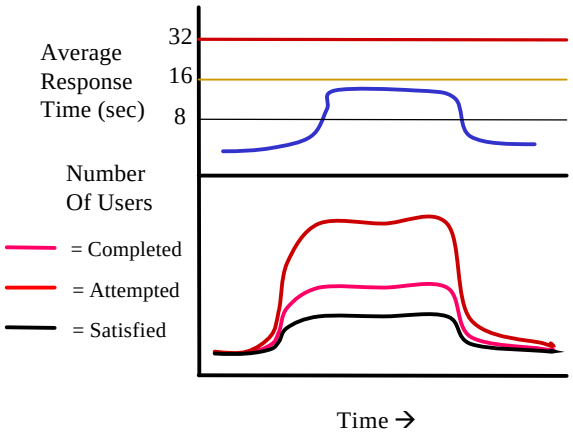
Since this will never happen, what's the value of this?

© 2000 Velogic Inc.

User Tolerance Affects Loads

Velogic
The eLoadTesting™ Company

Realistic load test results:



The number of actual users never reaches the planned number because the slow response turned people away.

This is what happens in the real world.

© 2000 Velogic Inc.

Scripts and Sessions

Velogic
The eLoadTesting™ Company

Definitions

Script: a sequence of page requests by a simulated user
Scenario: a combination of scripts

Scenario Objective: match the target page request distribution

Page Request Distribution	
Page	**% requests**
Home	58%
Product Info	31%
Buy	11%

Script Distribution	
Script	**% execution**
Script 1: Home→Exit	47%
Script 2: Home→Product Info→Exit	35%
Script 3: Home→Product Info→Buy→Exit	18%

© 2000 Velogic Inc.

Conclusions

Velogic
The eLoadTesting™ Company

- Web site load testing is very different from traditional load testing and requires new tools and new approaches.
- Most website load tests are wildly inaccurate and unrealistic and consequently useless and/or dangerously misleading.
- Accurate and realistic website load tests should include: website usage signature + online behavior profiling + user demographics.
- Approach website load testing as 80% science and 20% art.

© 2000 Velogic Inc.

The Science of Website Load Testing

Alberto Savoia
Chief Technology Officer
Velogic Inc.

1 Introduction

The concept and practice of software load testing has been around for many years, but the advent and exponential growth of the Internet has created a situation that has taken the magnitude and complexity of load testing to a whole new level. Traditionally, systems that required load testing existed in private LAN/WAN networks and were accessed at rather predictable times, with rather predictable patterns, by a relatively well known and predictable group of people. A website on the open Internet, on the other hand, is subject to highly unpredictable load patterns by a widely heterogeneous and unpredictable group of users. In the extremely competitive world of the Internet, unacceptable website performance and availability because of excessive loads can cause serious harm to a company's bottom line, market value, and brand. For these reasons, knowing the capacity and scalability of business and mission critical websites is extremely important, and proper load testing is the best way to acquire this knowledge. When load testing is not done properly, the results are at best useless and, in the worst case, misleading, causing a company to either underestimate or overestimate a site's capacity. This wrong result could cause unnecessary expenses, delays or potentially disastrous business decisions.

In this paper, we present the overview of a method for approaching the very new and challenging task of load testing Internet websites in a rigorous, systematic, and repeatable manner. Scientific load testing allows companies to collect realistic, useful, and reliable data about a website's capacity and scalability.

In the first part of the paper we explain the key variables and metrics we use to generate highly realistic load tests for Internet websites. While some of these variables and metrics are the same ones used in more traditional, non-Internet, load testing situations, most have either been tuned for the specific requirements of websites or have been developed specifically for them.

In the second part of the paper, we introduce and explain the concept of Website Usage Signature (WUS), a very effective and systematic way for combining and presenting the above mentioned metrics.

In the final section, we describe the process for developing and testing load scripts and scenarios that accurately match the way a website is used and navigated by real users.

2 Key Variables and Metrics for Internet Load Testing

In this first section of the paper we introduce and explain some of the variables and metrics that are used to analyze and reproduce website loads. We partition these variables and metrics into the following categories:

- Server-side variables and metrics
  - Basic variables
  - Derived variables
- Client-side variables and metrics
  - Online behavior variables
  - Client system variables

Basic Server-side Website Usage Variables and Metrics

This first set of variables and metrics provides a very high-level description of a website's traffic and usage patterns from the server's point of view. This description is easy to understand. Like a car's MPG rating, taken in isolation, these are coarse averages, measured over a wide range of conditions, so "your actual mileage may vary." Nevertheless, like an MPG rating, their usefulness and directness make up for their lack of detail. Furthermore, when used as a set rather than in isolation, their interrelationships weave a picture that provides real insight into the way a website is used. These basics variables and metrics are:

- Total page views per week
- Total hits per week
- Total user sessions per week
- Average session duration
- Average page size
- Average hit size

All of these metrics can easily be derived from log files using a basic log analyzer. They are described below.

Total Page Views Per Week

A page view is the request of a single web page with all its embedded objects. This simple metric is the number you'd like to have if you can only have one number to gauge a website's traffic. We chose page views (rather than page hits or bytes transferred) as the parameter for the numerator because this is the most intuitive and most commonly used unit for describing the traffic or size of a web page. We chose one week as the denominator because using anything less than that for a sample (i.e. a

day or an hour) fails to take into account some very common cyclic patterns that happen within a week, such as a much lighter volume on week-ends for business or financial sites. Longer time samples (i.e. a month or a year) are also inappropriate because in the highly dynamic world of the Internet, things can change dramatically in such time frames.

Total Hits Per Week

A hit is any request for a file received by the server, including images, sound files, and any other type of file that is requested along with a page request.

Total User Sessions Per Week

A user session is the visit to the web site where the interval between page requests from that user does not exceed a pre-determined time (we use 30 minutes).

Ideally, a user session would be defined as a visit by a **unique** user. The problem with this definition is that the concept of unique user is easier to describe than to measure. It's important to understand that, unless you do a retinal scan on your visitors, the information you get is not going to be as accurate as you might like. However, knowing unique users is still extremely useful and necessary for load testing. Having ruled out retinal scans, the best way to know that a user is unique is to request a user name and password every time the user visits the website. This is clearly slow and impractical for most websites. The next most accurate way to determine the uniqueness of a user is to use cookies (pieces of identification information put on the user's hard disk by the website.). The intrusiveness of this method, the fact that it's possible for a single system to be used by multiple users, and the reality that many users disable cookies altogether makes this approach less than ideal and inaccurate. The final way to determine the uniqueness of a user is to use IP addresses. While this approach is the easiest and least intrusive of the three, it is the least accurate because most Internet Service Providers (ISPs) use caching and load balancing schemes that can make a single user appear as multiple users, or multiple users appear as a single user.

Despite possible inaccuracies due to the difficulty in identifying unique users, average user sessions per week is a critical metric for website load testing because the only practical way to load test a website is to simulate actual users and actual user sessions navigating from one page to the next (rather than, say, simply requesting disconnected pages). Fortunately, the inaccuracies introduced by leveraging cookies or IP addresses can be mostly neutralized by being aware of their potential inaccuracies and, more importantly, by standardizing on one method and definition for identifying unique users and using it consistently throughout the entire load testing cycle.

Average Session Duration

This is the amount of time the average user session lasts, measured in minutes and seconds, from the first page request until the last byte of the last requested page is served. All the caveats about user session measurements also apply.

Average Page Size

This is the size, in Kbytes, of a page view. This includes all frames, images, etc. for that page.

Average Hit Size

This is the average size of a hit measured in Kbytes.

Derived Website Usage Variables and Metrics

The basic variables and metrics described in the previous session can be combined to yield an additional set of very useful metrics:

- **Average pages per session—**calculated by dividing total page views by total user sessions.
- **Average hits per page—**calculated by dividing the total number of hits by the total number of page views.
- **Average page viewing time—**calculated by dividing average pages per session by average session duration.

Website Specific Variables and Metrics

The metrics introduced in the previous sections are common and should be used on all websites, but, in addition, each website should track some metrics that are specific to its mission. An eCommerce website, for example, should track what percentage of user sessions result in an actual purchase, since such a transaction exercises and loads very specific subsystems (e.g. credit card authorization, secure server) whose performance is critical to the success of the website. Similarly, an online broker might want to keep track of how many stock quotes per session are requested by the average user, ratio of quotes to trades, etc.

Once the values for these variables have been collected, you begin to form a basic picture of what the website traffic looks like and what a realistic load simulation should look like.

Client-side Variables and Metrics

In our context, clients are people who visit a website and navigate it using a web-browser. The loads generated and the website response time experienced by these people can vary greatly depending on a number of what we call “client-side” variables. These variables fall into two major categories: online behavior variables and client system variables.

Online Behavior Variables

As the name suggests, online behavior variables deal with behavioral differences between users. Some users, for example, read web pages and navigate websites faster than others; we call this difference User Interaction Speed. Interaction speed is very relevant because, in a given amount of time, a fast user is able to go through more web pages than a slower user. This results in more requests, and therefore a higher load, for the website under test. These variables are used in combination with other Website Usage Signature variables to create realistic distributions.

Let’s say, for example, that the average viewing time for a website’s home page is 53 seconds. Although this number is extremely useful, it’s just an average. A slow visitor unfamiliar with the website may spend 2 minutes on the home page, while a fast visitor who is already familiar with the site may stay on it for just a couple of seconds before navigating to another page. A load test that does not consider these variations is simply not realistic and will generate misleading results. By combining the average viewing time with the user interaction speed variable and taking into account the user’s familiarity with the website, you create loads that simulate real usage much more accurately.

Some of the key online behavior variables to consider for website load testing are:

- **Interaction speed—**as already mentioned, this is a measure of how rapidly a user processes a web page and navigates to another page.
- **Latency tolerance—**a measure of how long a user will wait for a web page to load before taking some action (e.g. abandon the website, hit reload). At the present time, there is a widely used rule of thumb which says that most users will wait approximately 8 seconds for a web page to load and, after that time, they will start thinking about taking some other action.
- **Tenacity—**a measure of how determined a user is to accomplish something on the website. A user may have a low latency tolerance, but if the task they want to accomplish is extremely important to them (e.g. sell a stock during a steep market correction), the user will adjust his or her tolerance accordingly and will endure waits longer than usual.

- **Familiarity—**a measure of how well a user knows the website. It can be assumed that frequent visitors to a website know how to navigate it and where to go; they will process certain web pages more rapidly than new users.

Client-System Variables

Online behavior variables deal with differences between humans; client-system variables deal with differences between the hardware, software, and location of the client system.

Some of the key client-system variables to consider for website load testing are:

- **Connection speed—**how quickly the user can access the site. In terms of load and user experience, the difference between a 56K modem and a T1 line are very significant. It's important to know the percentage of users in each connection speed category and use that distribution in the scenarios.
- **Location—**in what geographic region is the user. A user's geographic location affects a number of key variables that have a significant impact on load. The number of "hops" and the backbone speeds in the path between the website and the client system, for example, determines how fast the packets will travel and how many packets are dropped.
- **Software/Hardware configuration—**depending on the website, variables such as type of browser, browser plug-ins, type of OS, or CPU speed, may have a significant impact on load.

Determining The Values of Client-Side Variables

Several methods and assumptions can be used to determine the values and ranges for these variables. Again, a careful analysis of the website log files provides you with some basic guidelines. There are many commercial and freely available tools that directly provide you basic information (e.g. browser type, geographical location). Other variables may require a bit more work. To determine the interaction speed values and distribution, for example, you could select a few specific web pages and extract from the log the average and standard deviation for the viewing time for each of those pages. With this information, you can easily create a statistically significant model for interaction speed. Some of the variables require significantly more work. To identify the variables and distribution for familiarity, for example, you may need to first distinguish new users from returning users and then analyze how their page viewing times vary.

In the absence of log files, or the lack of time or resources for detailed log analysis, you can leverage some of the metrics and statistics collected by companies such as Nielsen//NetRatings, Keynote, or MediaMetrix. Nielsen//NetRatings, for example, provides data on average page viewing times and user session duration based on a very large sample space of users and websites. Although these numbers are not from your specific website, they can work quite well as first approximations.

You can also run simple in-house experiments using employees and their friends and family to determine, for example, the page viewing time differences between new and returning users. As a last resort, you can use your intuition, or best guess, to determine these variables' average value and standard deviation, and assume a normal distribution for their ranges. For realistic load tests, even this last approach is preferable to ignoring these variables and creating a load test where every user comes from the same location, uses the same access speed, spends exactly 57 seconds on each page, and waits until a server timeout to abandon the website.

3 Website Usage Signature

A load test that does not reflect actual usage is at best useless, and it could be dangerously misleading (e.g. causing a company to either overestimate of underestimate the capacity and scalability of their website, with potentially disastrous consequences). In the previous section we introduced and defined some of the key variables and metrics used in website load testing; in this section we describe how such metrics are used systematically to design, develop, and test load scenarios that simulate, as accurately as possible, real load situations.

To measure how closely a load test matches real world usage, we developed the concept of Website Usage Signature (WUS). The WUS is a set of metrics and measurements that, taken collectively, yield a very comprehensive picture of website activity. If the WUS from a load test does not match closely the WUS from actual usage, you should seriously question the validity and applicability of any conclusion drawn from the tests. Conversely, if the two WUS's are well matched, then your level of confidence that any results apply to real world situations increases considerably.

Since every website is unique and needs to track site-specific metrics, there is no standard format or standard set of metrics for the WUS. Factors unique to each website determine which metrics to include. The important thing is to include enough measurements to make it very difficult for two signatures to match closely unless the two loads are also closely matched. If, as a trivial example, we use a WUS that tracks pages/session, but does not track average page size, our scripts may be biased toward substantially smaller pages, and capacity predictions from the load test could be dangerously over optimistic.

The following is an example of a WUS:

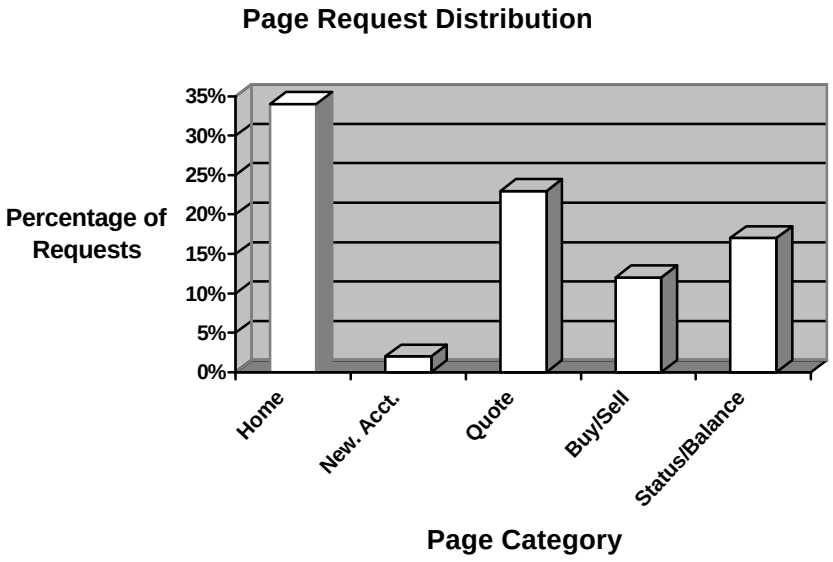
Website Usage Signature for sample.com	
Metric	**Results from Actual Usage** (Averages for 1-hr samples taken during the week of Jan. 24, 2000)
Page view per hour	97,200
Hits per hour	602,640
Average session duration	6:36
Average page views per session	7.20
Average hits per page	6.2
Average number of quotes per session	4.5
Average number of trades per session	0.34

You can increase the precision of a WUS by applying additional statistics to some of the metrics. Instead of simply reporting average session duration, for example, a WUS might include all of the following statistics for session duration.

Session Duration Statistics	
Average	6:36
Median	6:02
Standard deviation	3:18
Minimum	0:32
Maximum	32:56
Percentage less than 2 minutes	12%
Percentage more than 12 minutes	8%

Page Requests Distribution

The WUS metrics we have discussed so far are extremely important to determine the size of the traffic. However, since not all pages or hits are equal, it's equally important to analyze what pages (or types of pages) are requested and in what percentages. To collect and present this information, we use the concept of page requests distribution. A page request histogram, shown below, provides an easy way to look at the distribution of pages.



Page request distribution should be a key part of any WUS, because when planning and designing load test scripts and scenarios matching page requests is one of the most important factors in achieving realistic loads.

Tracking WUS Changes

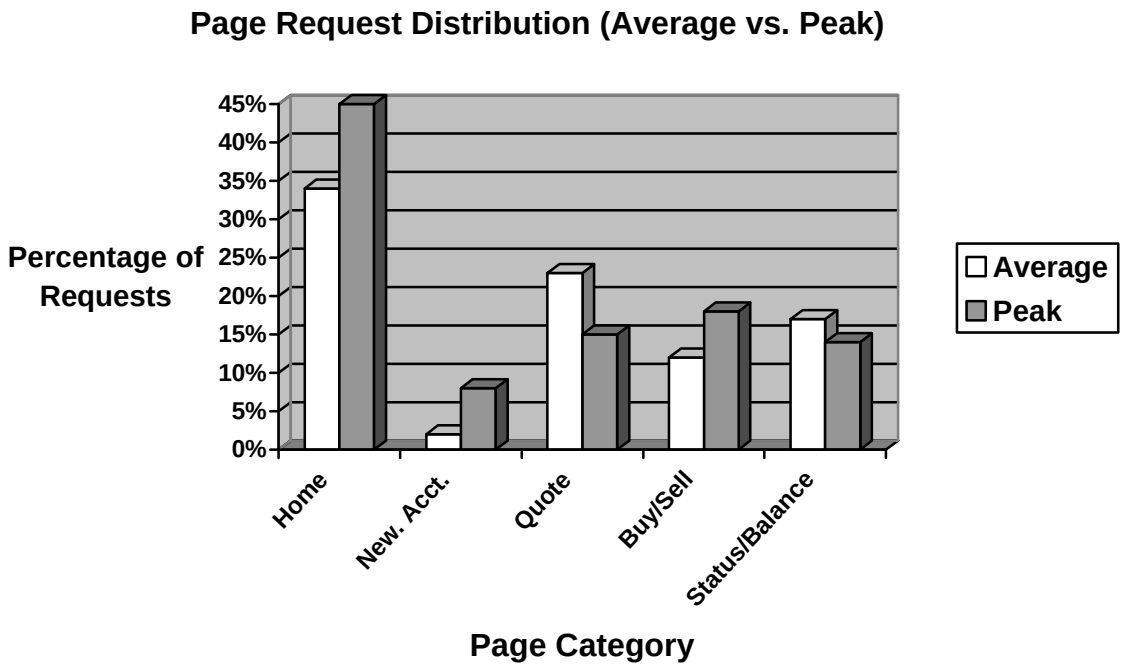
A website's WUS can change dramatically during peak load periods. During a normal trading day, for example, the average session on an online brokerage website may have a ratio of quotes to trades of 14 to 1, because most people are simply checking on their stocks rather than buying or selling them. On a day when the market is dropping sharply, on the other hand, the quotes–to-trades ratio may change to 5 to 1, as many people decide to sell in a panic. Since trades put a significantly higher load than quotes on the website, this would be a key WUS to track and analyze. A load scenario for a sharp market correction should be significantly different than a load scenario for an average high-volume day. Similar dramatic differences in WUS can be expected on eCommerce sites during special promotions or advertising campaigns or on portals or news sites when big news breaks.

Sharp increases in volume are accompanied by substantial changes in WUS. Designing and tracking a WUS with the right variables provides you with invaluable insight for creating highly realistic loads for very different scenarios.

The following table and chart show how the WUS can change as the load shifts from average to peak.

Average WUS vs. Peak Load WUS

Metric	Average Load Metrics	Peak Load Metrics	Comments
Sessions per hour	10,340	105,232	The average session duration, page views, and hits are all considerably smaller during peak loads. Most notably, a 10 time increase in number of users caused only a 3 time increase in hits per hour.
Page view per hour	97,200	456,708	
Hits per hour	602,640	1,806,343	
Average session duration	6:36	3:23	
Average page views per session	7.20	4.34	
Average hits per session	58.28	17.16	



As the table and chart show, the WUS differences between average and peak load can be dramatic; if they are not studied and taken into account, the conclusions from the load test would be highly questionable. In the example above, for instance, there is a difference of 2.86 page views per session between average and peak load. If you do not account for an approximately 40% reduction in page views between these two scenarios, for example, and use the 7.20 page views per session number in a peak load scenario (instead of 4.34) the session capacity reported by the test will be less than the actual capacity under peak load behavior.

4 Creating Realistic Loads

Having a Website Usage Signature reduces the guesswork and greatly simplifies the task of creating and testing realistic load scenarios. This task can be systematically approached as follows:

1. Develop the scripts and scenarios to achieve the desired page request distribution.

2. Modify the scripts and scenarios to take into account client-side variables.

3. Execute the scenarios and compare the resulting WUS to the target WUS. If the differences between the two WUS's are not within your specified tolerances make the necessary adjustment and go back to step 1 and 2.

Developing Scripts and Scenarios

The first step is to develop a set of session scripts that can be combined in various proportions to achieve the desired page request distribution. Many log analyzers report the most common paths through a site and, in many cases, the top 10 or 20 paths through the site are representative of 80% or more, of all visits. If a report on the most common paths is unavailable or too broadly distributed to be helpful, you can deduce the required scripts by using the page request distribution, as the following trivialized example shows.

Assume that you have a website with three pages: Home, Product Info, and Buy, and that an analysis of the log files results in the following page request distribution:

Page Request Distribution	
Page	**% requests**
Home	58%
Product Info	31%
Buy	11%

To be able to achieve this distribution you need to develop at least three scripts:

- Script 1: Home→Exit
- Script 2: Home→Product Info→Exit
- Script 3: Home→Product Info→Buy→Exit

After deciding what scripts to develop, you must decide in what proportions those scripts should be executed to achieve the desired page request distribution. In this case, the script distribution should be:

Script Distribution	
Script	**% execution**
Script 1: Home→Exit	47%
Script 2: Home→Product Info→Exit	35%
Script 3: Home→Product Info→Buy→Exit	18%

If 100 scripts are executed using this script distribution, your page requests will be:

- Home Page: 100 requests (47 from script 1 + 35 from script 2 + 18 from script 3)
- Product Info Page: 53 requests (35 from script 2 + 18 from script 3)
- Buy Page: 18 requests (from script 3)

The total number of pages requested from all these scripts is 171 (100 Home + 53 Product Info + 18 Buy). A simple calculation shows that the target page request distributions will be achieved when the load is executed (e.g. 53 Product Info page requests represents 31% of all page requests, which is the target distribution.)

Clearly, most websites are significantly more complicated than our three-page example, but the same principle can be applied to sites of arbitrary complexity.

Modify Scripts and Scenarios to Account for Client-side Variables

The next step in creating a realistic load test is to take into account what kind of users would be executing these scripts. This is where additional data from the WUS and the client side variables come into play. Let's consider, as an example, two client side variables: latency tolerance and familiarity, and see how they would impact the realism of the load.

Latency tolerance is an indicator of how long a user will wait for a page to load before becoming dissatisfied and, possibly, abandon the website. Different users will have different tolerance levels; some may tolerate loading times of 20 seconds and others will leave the site if they are made to wait more than 8 seconds for a page to load. A load scenario can be made significantly more realistic by simulating the behavior of tolerant and intolerant users. This can be done by assuming a simple distribution and by modifying the scripts to take latency tolerance into account (e.g. by terminating a script

if the simulated user experiences response time past its threshold for two consecutive pages.)

The following table shows a sample latency tolerance distribution and script modification:

Latency Tolerance Simulation			
Latency Tolerance	**% of Users**	**Page Response Threshold (sec)**	**Effect on Script**
Low	25%	8	Terminate script if 2 consecutive pages exceed the page response threshold
Medium	50%	16	
High	25%	24	

Familiarity is a major factor in how quickly a simulated user navigates from one page to the next. As with latency tolerance, different people will behave in different ways: users that are very familiar with the website move more rapidly (therefore creating more load per unit of time) than users who are visiting the website for the first time and need to read and understand how the website is organized to go from one page to the next. You can use a familiarity simulation table to partition the different types of users and define how familiarity impacts the script execution:

Familiarity Simulation			
Familiarity	**% of Users**	**Read, Write, Think Time Multiplier**	**Effect on Script**
First-time users	35%	2.0	The read/write/think time is adjusted for each type using the specified multiplier.
Returning users	55%	1.0	
Experienced users	10%	0.5	

The scope and length of this paper prevents us from going through all the possible client-side variables, but these two examples are indicative of how the simulation of other client side variables can be approached.

Comparing Website Usage Signatures

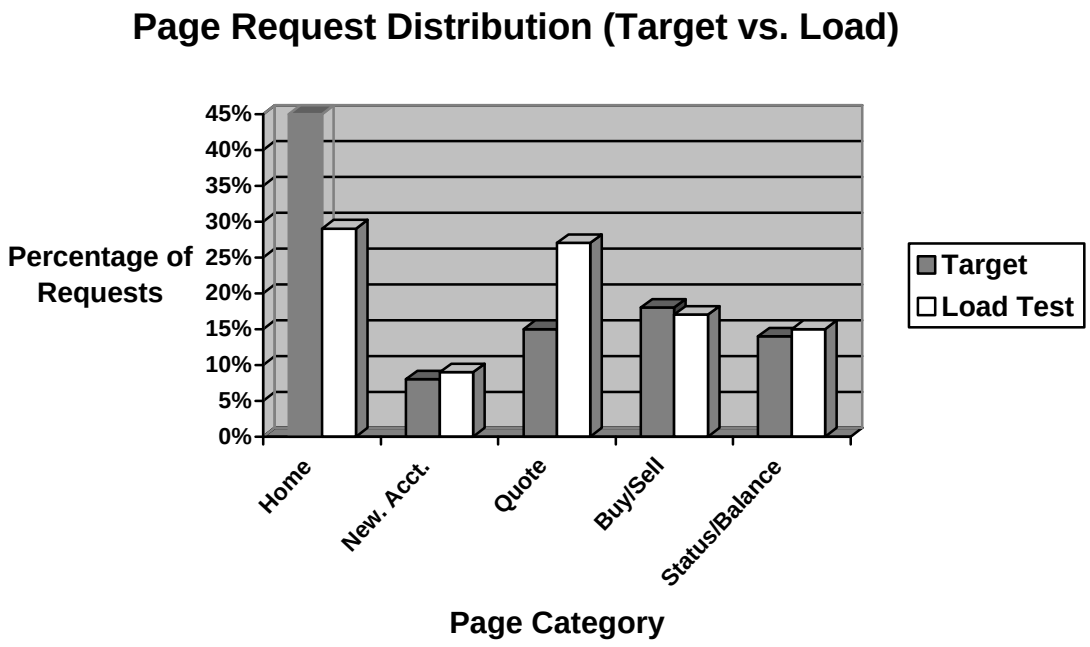
After the scripts and scenarios have been developed and modified to take into account client-side variables, they are executed and the resulting log files analyzed to determine how well the simulated load matches the target WUS. A WUS comparison chart and page request distribution table makes it easy to spot any differences.

In the following WUS comparison chart, for example, we see that although the page views per session and hits per session are both within five percent of each other, the session duration in the load test is 43.7% shorter than during real usage. This major difference would significantly impact the realism of the test and the accuracy of the results. The read/write/think time in the scripts should be modified to make the sessions last longer.

WUS Comparison Chart

Metric	Results from Actual Usage (Averages for 1-hr samples taken during the week of Jan. 24, 2000)	Results from Load Test	Difference
Page view per hour	97,200	95,665	-1.6%
Hits per hour	602,640	580,443	-3.6%
Average session duration	6:36	3:43	+43.7%
Average page views per session	7.20	7.08	-1.6%
Average hits per session	58.28	56.23	3.5%

The following page request distribution chart shows another discrepancy between the load scenario and the target WUS. The percentage of home pages requested by the load test is significantly less than what the target WUS calls for, while the reverse is true for quote pages. The script distribution scenario should be changed to make the simulated load match the target load more accurately.



Achieving a perfect match between load and target WUS is practically impossible, but you should strive to have all the key WUS components within 5-15% of each other.

5 Conclusion

Load testing is the most effective way to gauge a website's capacity and scalability, but load tests that don't simulate real scenarios can be dangerously misleading. In this paper we introduced an approach for planning, developing, executing, and validating load scenarios that can accurately replicate real loads.

Our systematic method for standardizing, collecting, organizing, and comparing load testing variables, makes it possible to approach load testing in a rigorous, scientific, and repeatable, manner. The huge numbers and ranges of variables involved in website load testing will always present challenges and surprises. This framework gives load-testing practitioners the necessary tools to study, understand, and control these variables. Some amount of guesswork will always be required, but we firmly believe that techniques like the Website Usage Signature can be used to reduce guesswork significantly and achieve results that are more reliable.

QW2000 Paper 3M1

Mr. Phil Lones
(Lucent Technologies)

A Practical Approach to Testing Software in an Evolutionary Delivery Environment

BACK TO QW2000 PROGRAM

Key Points

- Brief description of Evolutionary Delivery.
- Introduction to steps required to effectively test in an EVO environment
- Detailed process of testing during evolutionary deliveries.

Presentation Abstract

Are they new processes or just buzzwords? Is it the next "Silver Bullet" or just a passing fad? As a profession, I believe we jump too often for the newest trend to solve our problems without really looking at the tools, techniques, and processes already employed.

The evolutionary model or EVO has been used on several projects with great success including the Message ManagerÖ project here at Lucent Technologies. We have been able to consistently keep the project within budget and meet scheduled deliveries. However, using Evolutionary Delivery invalidates traditional test methodologies. Testing becomes an integrated, continuous process throughout the development life cycle not an "add-on" function at "code complete".

Just as the Evolutionary Delivery model itself requires a different mindset so does "Evolutionary Testing". We must still employ sound concepts and techniques toward testing but we have to think about the process differently.

This presentation does not describe any revolutionary new ideas or processes rather it describes a practical methodology that will produce positive results. I will describe the testing methodology used on Message Manager and our current project and by applying these techniques, rapid deliveries can easily be accommodated. Thinking "Evolutionary" has made the testing effort more productive and has created a better product for our customers.

About the Author

Mr. Philip Lones (Lucent Technologies) [USA] Phil Lones is currently working as the test manager for PC and web based projects with Lucent Technologies. He has several yearsÆ experience testing mainframe and client server applications. He presented his paper on Evolutionary Delivery at the 1998 Lucent Software Symposium, local SPIN and ACM chapters and last yearÆs Quality Week

Symposium.

Mr. Lones holds an MS degree in Information Systems Management from the University of Colorado at Denver, a BS degree in Computer Science from Metro State College of Denver and a BS degree in Electrical Engineering from the University of Washington. He is a member of the Association for Computer Machinery, the Rocky Mt. Information Managers Association, and SQUAD, Software Quality Association of Denver.

BACK TO QW2000 PROGRAM

A Practical Approach to Testing Software in an Evolutionary Delivery Environment

Philip Lones
Lucent Technologies

Introduction

Bug Fixes
Design
Integration
Sanity Test
Beta Trials
Analysis
System Test
Deliverables
Regression Test
Functional Test
Development Life Cycle
Implementation

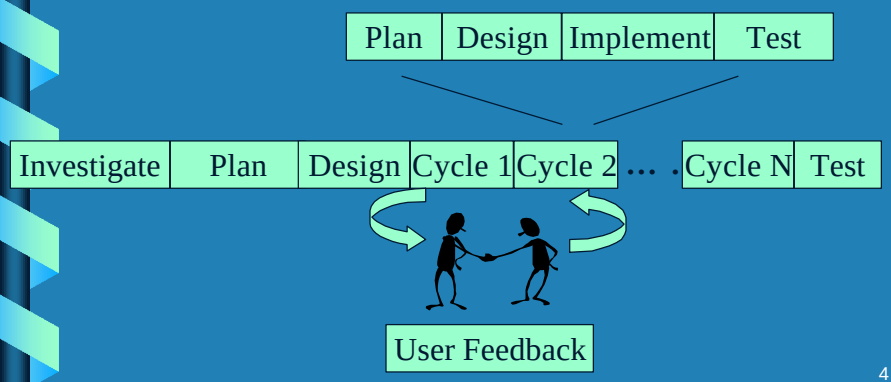
Introduction

- Order out of chaos
- Valid processes
- Evolutionary Delivery -EVO

Evolutionary Delivery

Evolutionary Delivery Model - EVO

Tom Gilb (http://www.Result-Planning.com)



Evolutionary Delivery

Key Concepts

- Early, frequent iteration
- Plan, design, implement and test in each cycle
- User orientation
- Result orientation, not software development process orientation

Evolutionary Delivery

- Message Manager™ , www.messenger™
  - Plan
  - Design
  - Implement
  - Test
- Evolutionary Testing

Evolutionary Testing

- Change Mindset - not an end of project activity.
- Ongoing activity throughout development lifecycle
- Questions
  - Why test?
  - When is enough enough?

Evolutionary Testing

- Test Axioms
  - Can't test everything
  - Will always have some "bugs"
  - NO RISK NO GLORY

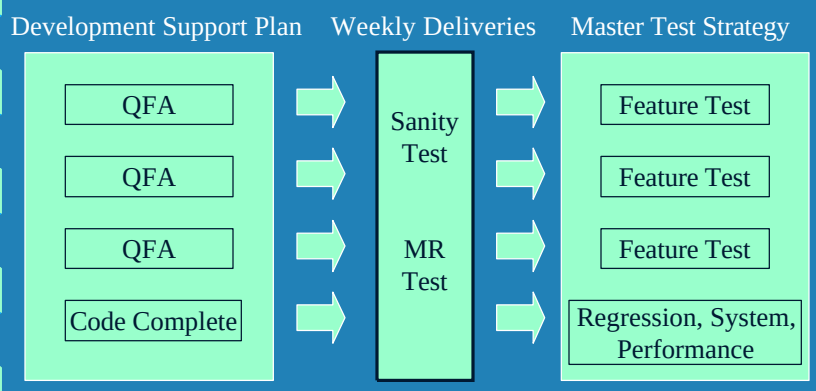
Evolutionary Testing

- Develop test strategy
- Map to delivery schedule
- Execute!!
- Be Flexible

Test Strategy

- Master Test Strategy Plan
  - Identify type of testing
    - Sanity, Bug fixes, Functional, Regression, System, Performance , etc
  - Test Cycles - Map to Development Support Plan
    - Weekly deliverable - Sanity and Bug fixes
    - QFA - functional
    - Code Complete- Regression, System, Performance

Test Strategy

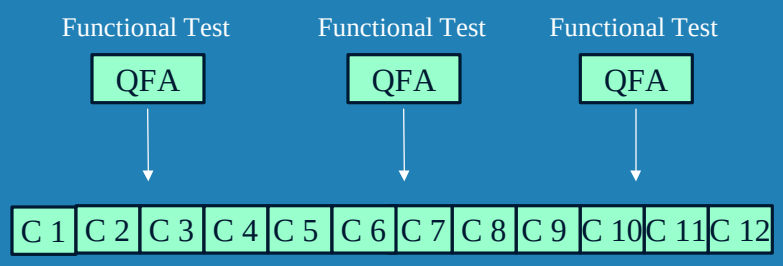


Test Strategy

- Weekly deliverables
  - Sanity test - basic functionality
    - Number of test cases increase as functionality is added - identify areas of risk for test cases.
  - MRs (bug fixes).
    - All MRs introduced into code base are tested.

Test Strategy

Quality Factor Assessment - QFA



Test Strategy

Functional Test

- Full feature functionality
- All supported systems
- Test to the requirements

Test Strategy

- Code Complete
  - Regression test
    - Does everything still work?
    - Supplement Sanity Test
  - System Test - meets requirements
    - Customer scenarios
    - Identify Risks
  - Performance/Load - meets requirements

Conclusion

- Evolutionary Delivery
  - Early and frequent builds to the customer
  - Plan, Design, Implement, Test in each build
- Evolutionary Testing
  - Continuous throughout life cycle

Conclusion

- Testing Process
  - Test Strategy - Tailor to project
    - Delivery cycle - Sanity and MR testing
    - Functional Testing
    - Regression, System, Performance Testing

Conclusion

Development Life Cycle

Design Analysis Implementation

Sanity Test
Bug Fixes
Functional Test
Regression Test
System Test

Deliverables Integration Beta Trials

Conclusion

- Principles of Software Engineering, Tom Gilb, Addison-Wesley, 1988
- The Evolutionary Project Managers Handbook,
- Tom Gilb, manuscript, (http://www.Result-Planning.com)
- Lones@lucent.com

QW2000 QuickStart 3Q

Mr. Robert Binder
(RBSC Corporation)

Best Practices for Object-Oriented Systems

BACK TO QW2000 PROGRAM

Key Points

- Lessons Learned
- State of the Art
- Stat of the Practice

About the Author

Robert V. Binder has over 22 years of software development experience. He is President of RBSC Corporation, providing consulting and training in software engineering and software process improvement since 1984. He is author of "Application Debugging" (Prentice-Hall, 1985). "Testing Object-Oriented Systems" is under contract with Addison-Wesley. He writes a regular column on testing for "Object" magazine. His articles have appeared in American Programmer, Communications of the ACM, Computerworld, CASE Outlook, CASE Trends, Database Programming and Design, IEEE Computer, Journal of Knowledge Engineering, Journal of Software Testing, Verification, and Reliability, and Software Development.

He is the Chair of a newly formed study group to develop an IEEE standard for built-in test for object-oriented software. Mr. Binder has an MS in Electrical Engineering and Computer Science from the University of Illinois at Chicago, and a BA and MBA from the University of Chicago. He is an IEEE Senior Member, a member of the ACM, and holds the CDP and CCP.

BACK TO QW2000 PROGRAM

Testing Object-Oriented Systems: Best Practices

Robert V. Binder
RBSC Corporation www.rbsc.com

Software Quality Week Quick Start Tutorial

May 31, 2000

Copyright © 2000, RBSC Corporation

Overview

- Best Practices
- Test Patterns
- OO Testing Benchmarks
- Improvement Strategies

Copyright © 2000 RBSC Corporation 2

Copyright © 2000 RBSC Corporation

Key Practice Dimensions

- Representation
- Design for Testability
- Test Design
- Test Automation
- Process Integration

Copyright © 2000 RBSC Corporation 3

Representation

- Best Practices
  - UML/OCL 1.0
  - Syntropy, Design by Contract
  - Design Patterns
- Challenges
  - Architecture
  - Limits of cartoons
  - Test design as software engineering
  - Testable representations

Copyright © 2000 RBSC Corporation 4

Copyright © 2000 RBSC Corporation

Design for Testability

- Best Practices
  - Frameworks/libraries with assertions
  - Lakos' levelizable architecture
  - Percolation pattern

Copyright © 2000 RBSC Corporation 5

Design for Testability

- Challenges
  - Seamless language support
  - Object-level BIT services
  - Runtime spec-checking (assertions)
  - OO testability an oxymoron?
  - Loss of intellectual control
  - Entropy horizon about 24 months
  - Frameworks and components

Copyright © 2000 RBSC Corporation 6

Copyright © 2000 RBSC Corporation

Test Design

- Best Practices
  - Test Design Patterns
- Challenges
  - Intra-class coverage
  - Polymorphic paths
  - Testing for reuse (frameworks)
  - Validated failure metrics/fault models

Copyright © 2000 RBSC Corporation 7

Test Design Pattern

- New pattern schema for test design

Name/Intent
Context
Fault Model
Test Model
Entry Criteria
Exit Criteria
Consequences
Known Uses

Test Model
Test Procedure
Oracle
Automation

Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley.

Copyright © 2000 RBSC Corporation

Test Design Patterns

- Method Scope
  - Category-Partition
  - Combinational Function
  - Recursive Function
  - Polymorphic Message
- Class/Cluster Scope
  - Invariant Boundaries
  - Modal Class
  - Quasi-Modal Class
  - Polymorphic Server
  - Modal Hierarchy

Copyright © 2000 RBSC Corporation 9

Test Design Patterns

- Subsystem Scope
  - Class Associations
  - Round-Trip Scenarios
  - Mode Machine
  - Controlled Exceptions
- Intra Class Integration
  - Small Pop
  - Alpha-Omega Cycle
- Integration Strategy
  - Big Bang
  - Bottom up
  - Top Down
  - Collaborations
  - Backbone
  - Layers
  - Star
  - Mesh
  - High Frequency

Copyright © 2000 RBSC Corporation 10

Copyright © 2000 RBSC Corporation

Test Design Patterns

- System Scope
  - Extended Use Cases
  - Covered in CRUD
  - Allocate by Profile
- Regression Testing
  - Retest All
  - Retest Risky Use Cases
  - Retest Profile
  - Retest Changed Code
  - Retest Within Firewall

Copyright © 2000 RBSC Corporation 11

Modal Class Test Pattern

- Intent
  - Develop a class scope test suite for class that has fixed constraints on message sequence.

Copyright © 2000 RBSC Corporation 12

Copyright © 2000 RBSC Corporation

Modal Class Test Pattern

- Context
  - Class modeled with UML Statechart
  - Sequentially constrained behavior can be very complex
  - How can we devise a test suite that exercises all dependencies?
  - E.g, we can't withdraw funds from an overdrawn Account object

Copyright © 2000 RBSC Corporation 13

Modal Class Test Pattern

- Fault Model
  - Missing transition or action
  - Incorrect transition or action
  - Sneak path
  - Incorrect or corrupt state
  - Reaching, Triggering, Propagating
    - All conditions met
    - N-switch transition cover

Copyright © 2000 RBSC Corporation 14

Copyright © 2000 RBSC Corporation

Modal Class Test Pattern

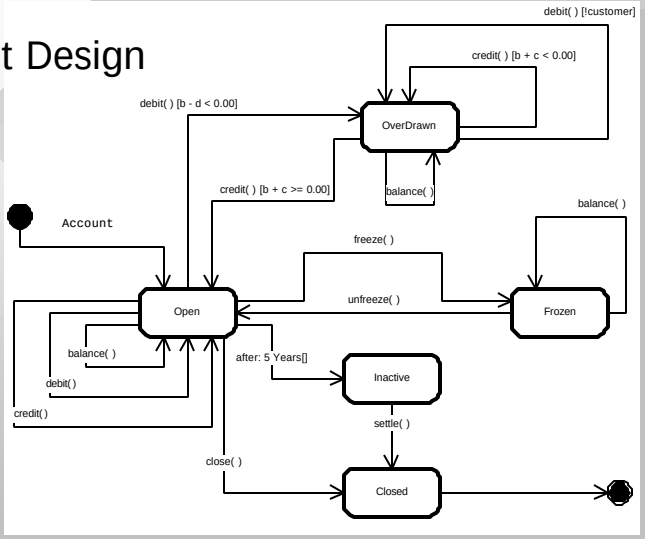
- Test Design

```
class Account extends Object {
/* Java Fragment */
        private      Money  currentBalance;
        private      int    accountNumber;
        private      Date   lastActivity;
        public void  open() {/* */};
        public Money balance() {/* */};
        public void  credit (Money creditAmt) {/* */};
        public void  debit(Money debitAmt) {/* */};
        public void  freeze() {/* */};
        public void  unfreeze() {/* */};
        public Money settle() {/* */};
        public void  close() {/* */};
}
```

Copyright © 2000 RBSC Corporation 15

Modal Class Test Pattern

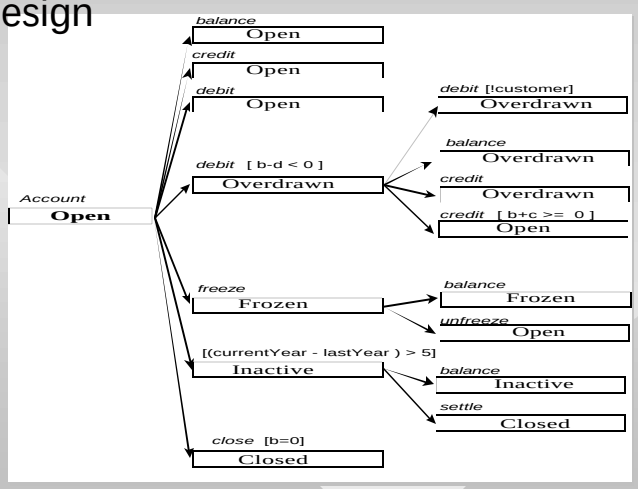
- Test Design



Copyright © 2000 RBSC Corporation 16

Modal Class Test Pattern

- Test Design



Copyright © 2000 RBSC Corporation 17

Modal Class Test Pattern

- Test Design

Events	States				
	Open	Over Drawn	Frozen	Inactive	Closed
Account	PSP	PSP	PSP	PSP	PSP
credit		?	PSP	PSP	PSP
debit	?	?	PSP	PSP	PSP
balance					PSP
freeze		PSP	PSP	PSP	PSP
unfreeze	PSP			PSP	PSP
settle	PSP	PSP	PSP		PSP
5 years		PSP	PSP	PSP	PSP
close	?	PSP	PSP	PSP	PSP

= Valid transition PSP = Possible Sneak Path ? = Conditional Transition

Copyright © 2000 RBSC Corporation 18

Lessons Learned

- Class/Cluster test design
  - Super/subclass interaction must be tested: test at flattened scope.
  - Design subclass test suites to re-run on superclasses.
  - Design superclass test suites to re-run on subclasses.
  - Test polymorphic servers for LSP compliance.

Copyright © 2000 RBSC Corporation 19

Lessons Learned

- Class/Cluster test design
  - Exercise each binding of a polymorphic server message
  - Test all parameters for generics
  - Test interface data flow of non-modal classes

Copyright © 2000 RBSC Corporation 20

Copyright © 2000 RBSC Corporation

Lessons Learned

- Subsystem/system test design
  - Control easily obscured or accidental
    - Complex dependencies between concrete state and message sequence
    - Hierarchic control in state-based subclasses
    - Mosaic modularity at larger scope
  - Model behavior with state machines; achieve transition cover or better

Copyright © 2000 RBSC Corporation 21

Lessons Learned

- Subsystem/system test design
  - Objects don't compose (easily)
  - Producer's framework should not be in consumer's test scope
  - Minimum system/subsystem test includes
    - Testing exceptions
    - Testing class associations
    - Testing use cases (requires testable content)

Copyright © 2000 RBSC Corporation 22

Test Automation

- Best Practices
  - Test Automation Design Patterns
  - Automatic driver generation
  - Simple coverage analyzers

Copyright © 2000 RBSC Corporation 23

Test Automation

- Challenges
  - Intra-class coverage
  - Spec-based test generation
  - Testing for reuse
  - Validated failure metrics/fault models

Copyright © 2000 RBSC Corporation 24

Test Automation Patterns

- Test Case Implementation
  - Test Case/TestSuite Method
  - Catch All Exceptions
  - Test Case /Test Suite Class
- Test Control
  - Server Stub
  - Server Proxy
- Driver Patterns
  - TestDriver Super Class
  - Percolate the Object Under Test
  - Symmetric Driver
  - Subclass Driver
  - Private Access Driver
  - Test Control Interface
  - Drone
  - Built-in Test Driver

Copyright © 2000 RBSC Corporation 25

Test Automation Patterns

- Test Execution
  - Command Line Test Bundle
  - Incremental Testing Framework
  - Fresh Objects
- Built-in Test
  - Coherence assertions
  - Percolation
  - Built-in Test Driver

Copyright © 2000 RBSC Corporation 26

Lessons Learned

- Test Automation
  - Encapsulation and mosaic modularity decrease controllability and observability
  - Design-by-contract/assertions is the only practical counter-measure for inherent non-determinism and loss of testability

Copyright © 2000 RBSC Corporation 27

Lessons Learned

- Test Automation
  - Avoid stubs: increase scope of the IUT or test in bottom-up order
  - Design test harness to exploit the structure and particulars of the system under test
  - Complete app = app components + test components under CM control

Copyright © 2000 RBSC Corporation 28

Copyright © 2000 RBSC Corporation

Test Process

- Best Practices
  - Testing by scope (about 10%)
  - Many embedded/real-time shops
  - Extreme Programming test approach
- Challenges
  - High-frequency/short cycle development
  - Planning formal testing with iterative incremental process
  - Attitudes about testing

Copyright © 2000 RBSC Corporation 29

Test Process Benchmarks

- Testing by Poking Around
- Testing by Use Cases
- Testing by Scope

Copyright © 2000 RBSC Corporation 30

Testing by Poking Around

- About 70% of all organizations
- Characteristics
  - Developer discretion
  - No test entry/exit criteria
  - High tolerance for low quality

Copyright © 2000 RBSC Corporation 31

Testing by Poking Around

- Improvement Strategy
  - Assess limits of improvability
  - Train developers in basic test design
  - Install basic tool set
    - Coverage analyzer
    - Memory leak detector
    - Test Harness generator

Copyright © 2000 RBSC Corporation 32

Copyright © 2000 RBSC Corporation

Testing by Use-Cases

- About 20% of all organizations
- Complies with Unified Process test approach
- Characteristics
  - Assumes objects “just work”
  - System test from use cases
  - Frustrated with chronic bugginess

Copyright © 2000 RBSC Corporation 33

Testing by Use-Cases

- Improvement Strategy
  - Achieve exit criteria for indicated class/cluster test patterns
  - Use appropriate component/subsystem test design patterns
  - Develop testable use cases
  - Implement test automation to support regression testing

Copyright © 2000 RBSC Corporation 34

Copyright © 2000 RBSC Corporation

Testing by Scope

- About one in ten
- Best-in-Class level
- Characteristics
  - Test design corresponds to scope
  - Scope-specific test entry/exit criteria
  - Appropriate testing at all scopes
  - Effective test automation
  - Stable, repeatable process

Copyright © 2000 RBSC Corporation 35

Testing by Scope

- Improvement Strategy
  - Internal test design pattern-mining
  - Design for testability
  - Advanced test automation
  - Quantified closed loop feedback

Copyright © 2000 RBSC Corporation 36

Copyright © 2000 RBSC Corporation

Best Practice Examples

- MPR Teltech
- Stepstone Corporation
- Acme Widget/XP
- Ericsson CEE Project
- *Testing was the primary quality technique*

Copyright © 2000 RBSC Corporation 37

MPR Teltech

- C++/Eiffel telecom network management
- Initially, sporadic class/cluster testing
- Persistent high defect levels
- Instituted systematic class cluster testing
- System test defects reduced by 70:1
- Total development effort cut by half

Copyright © 2000 RBSC Corporation 38

Stepstone Corporation

- IC-Pak Objective-C class library
- Inspections for all classes
- Extensive automated test harness developed for each complex class
- No systematic test design

Copyright © 2000 RBSC Corporation 39

Acme Widget

- 50 KLOC Smalltalk app
- 45% of development used for testing
- Test harness for each class -- semi-daily build
- Same approach used by “Extreme Programming”

Copyright © 2000 RBSC Corporation 40

Ericsson

- 75 KLOC C++ cellular support application
- Systematic testing at class, cluster, and system scope
- No other verification techniques used

Copyright © 2000 RBSC Corporation 41

Lessons Learned

- Process
  - Inspect for omissions and inconsistencies, test for everything else
  - Design for testability
    - Implement hierarchic architecture patterns
    - Eliminate or encapsulate cyclic dependencies
    - Assert class invariants, at least
  - Support reuse with complementary producer/consumer testing strategies

Copyright © 2000 RBSC Corporation 42

Copyright © 2000 RBSC Corporation

Lessons Learned

- Process
  - 3 to 6 development increments
    - Developer class/cluster test -- Run tests locally, design tests globally: “unigration”
    - XP: “Continuous integration, relentless testing”
    - Independent build/integration group tests completed increment
    - Test suites must be regress-able
  - System testing on final increment

Copyright © 2000 RBSC Corporation 43

World Class OO Quality

- Best-in-Class level:
  - An average of “less than 0.025 user-reported defects per function point” in the first year after release
- World Class = 10x Best in Class

Capers Jones. *Software quality: Analysis and Guidelines for Success*.

(London: International Thompson Computer Press, 1997.) p 44.

Copyright © 2000 RBSC Corporation 44

World Class OO Quality

- OO testing best practices can achieve world class software quality

Organization Language	*KLOC*	*FP*	*Major Post Release Bugs*	*Bugs/FP*
Acme Widget Smalltalk	50	2381	11	0.0046
Ericsson C++	75	1364	7	0.0051
Stepstone Objective-C	12	414	5	0.0121

Copyright © 2000 RBSC Corporation 45

Summary

- Lessons learned: unique test design for OO implementations
- State of the art: expressed in patterns
- State of the practice: world class quality can be achieved through testing

Copyright © 2000 RBSC Corporation 46

Copyright © 2000 RBSC Corporation

QW2000 Vendor Technical Paper 3V2

Mr. Christian Hote
(PolySpace)

Runtime Error Detection

BACK TO QW2000 PROGRAM

Key Points

- Efficiency: PolySpace Verifier turns detection of run-time errors on its head: PolySpace Verifier exhaustively and automatically detects sources of run-time errors.
- Quality: It finds run-time errors even in critical applications validated using existing top-level testing and debugging methods.
- Productivity: Reduce verification and validation time and costs while drastically improving software quality. Neither changes to your process, nor constraints on the coding style are imposed.

About the Author

Christian HOTE obtained his PhD in Physics in 1991. He joined Verilog (European leader CASE Tools provider) as product manager and participated in several European Research programs (Eureka, Esprit) for embedded systems design and development. He joined PolySpace Technologies at its creation and manages US business development and operations.

BACK TO QW2000 PROGRAM

PolySpace Technologies

Software Verification and Validation Products

The problem!

- *Run-time errors or non determinism may cause:*
  - *mission loss*
  - *availability losses*
- ***Financial, company image and even human lives consequences***

Finding Errors Costs More and More

- Today, 40% of errors detected during maintenance are RTE
  (IBM T.J. Watson lab. and Berkeley Univ. studies)
- Tomorrow, Software Verification costs will still increase due to increasing applications size and complexity

  (up to 50% of total development costs)

The PolySpace Answer

- Automation of Verification Processes Using New, Simple and Non Intrusive Tools, Satisfying:
  - No Change in Development Processes
  - Quality Gain
  - Productivity Gain

An exercise

```
-- arithmetic exception
procedure arith_2 is
v : long_float;
p : long_float;
y : long_float;
u : float := random.random;
begin
    arith_1 (u, v);
    p := v - 0.75;
    pragma inspection_point(v,p);
    y := sqrt ( p);
end arith_2;

-- unreachable or dead code by linear constraint
procedure unr is
   x : integer := random.random;
   y : integer := random.random;
begin
       if (x > y) then
             x := x - y;
             if (x < 0) then
                    x := x + 1;
             end if;
       end if;
end unr;

Scale  : constant Tab_Float(1..10) :=
(  0.0000000E+00 ,   2.0000000E+05 ,   4.0000000E+05 ,   6.0000000E+05 ,   8.0000000E+05 ,
   1.0000000E+06 ,   1.2000000E+06 ,   1.4000000E+06 ,   1.6000000E+06 ,   1.8000000E+06) ;
-- floating arrays
procedure array_1 is
   Tmp : Float := 0.0;
   Sum : Float := 0.0;
   K : Integer;
begin
   for I in 1 .. 9 loop
      Tmp := Tmp + (1.0 / (Scale (I+1) - Scale (I)));
      for J in 1 .. I loop
         K := I-J+1;
         pragma inspection_point (K);  -- to display variation domain
         Sum := Sum + Scale(J) - Scale (K);
      end loop;
   end loop;
end array_1;
```

Problem:

How many run time errors ?

Existing Methods

- *Compiler, lint-like tools: type check, syntax check, static expressions solving;*
- *Static analysis tools: Coding rules checks, quality metrics, test coverage;*
- *Hours of simulation;*
- *Hours of code reviews.*

PolySpace Verifier

- Non Intrusive : Source code only;
- Exhaustive : Identifies and checks all potential sources of run-time errors;
- Automatic : Predicts where run-time errors occur without execution.

An exercise (following)

```
-- arithmetic exception
procedure arith_2 is
v : long_float;
p : long_float;
y : long_float;
u : float := random.random;
begin
    arith_1 (u, v);
    p := v - 0.75;
    pragma inspection_point(v,p);
    y := sqrt ( p);
end arith_2;

-- unreachable or dead code by linear constraint
procedure unr is
   x : integer := random.random;
   y : integer := random.random;
begin
       if (x > y) then
              x := x - y;
              if (x < 0) then
                     x := x + 1;
              end if;
       end if;
end unr;

Scale  : constant Tab_Float(1..10) :=
(  0.0000000E+00 ,   2.0000000E+05 ,   4.0000000E+05 ,   6.0000000E+05 ,
   1.0000000E+06 ,   1.2000000E+06 ,   1.4000000E+06 ,   1.6000000E+06 ,
-- floating arrays
procedure array_1 is
   Tmp : Float := 0.0;
   Sum : Float := 0.0;
   K : Integer;
begin
   for I in 1 .. 9 loop
      Tmp := Tmp + (1.0 / (Scale (I+1) - Scale (I)));
      for J in 1 .. I loop
         K := I-J+1;
         pragma inspection_point (K);  -- to display variation domain
         Sum := Sum + Scale(J) - Scale (K);
      end loop;
   end loop;
end array_1;
```

Solution:

Use PolySpace Verifier and Look at colors !!

Detected Errors

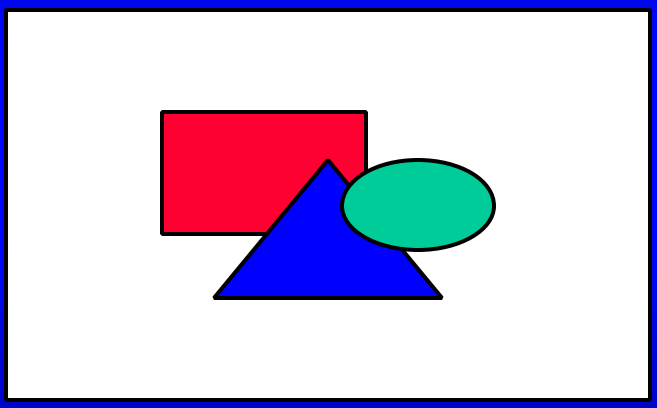
- Run-time errors
  - Division by zero
  - Out-of-bounds array accesses & buffer overflows
  - Arithmetic overflows & underflows (integer, FP)
  - Arithmetic exceptions: sqrt(<0),…
  - Illegal type conversion
- Unreachable code

Detected Errors

- Non deterministic constructs
  - Unsafe global data access
  - Read access to non initialized variables
  - Data corruption
- Multi-tasking
  - Concurrent access on unprotected shared data

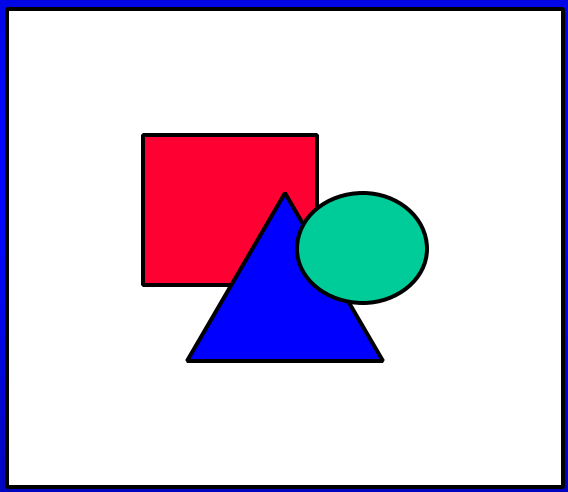
PolySpace Viewer

- Global Data Dictionary



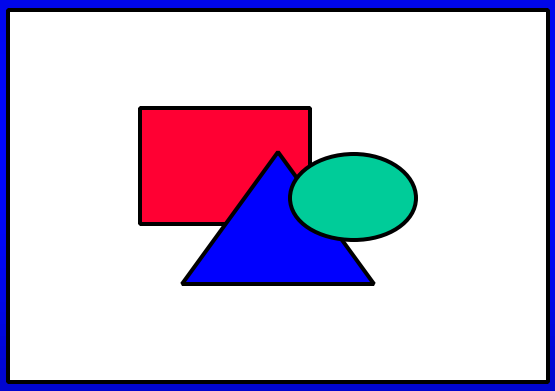
PolySpace Viewer

- Results on Shared Data



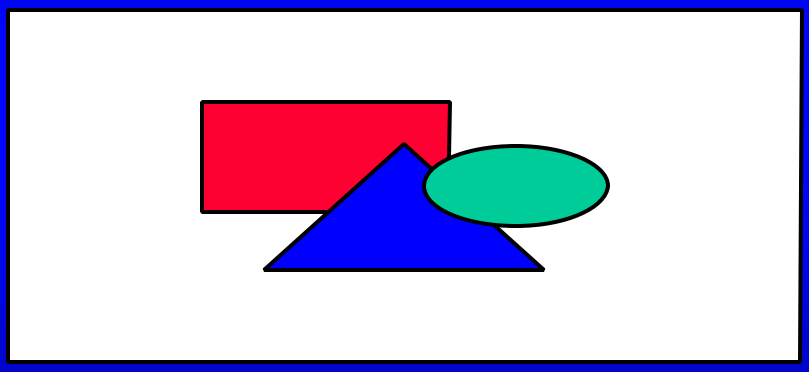
PolySpace Viewer

- *Results on Run-Time Errors*



PolySpace Viewer

- *Results on run-time errors*



Benefits

- ***Efficiency***
  - *Actual detection of run-time errors*
  - *Non intrusive and simple technology*
- ***Quality***
  - *More focused code review*
  - *Exhaustive detection of errors (confidence improvement)*
- ***ROI***
  - *Drastic reduction of verification efforts thanks to high selectivity rate (97-99% of code is checked)*
  - *Early detection of errors*
  - *Repeat your gains for each new software release*
- *1 m.m cost instead of 16m.m for code review*

Conclusion

- ***Technological breakthrough*** *in 1997: successful automated verification of the 502 flight software is a* ***world première***
- *High quality and productivity gains*
- *No modification of development processes*

PolySpace Technologies

Company Profile

Business Mission Statement

- Provide software validation and verification tools allowing actual and valuable benefits in terms of quality and productivity

Market

- Embedded Software Industries
  - avionics, space, automotive, energy, telecommunications, transport, defense, medical…

Customer References

- Ariane 502, 503
  - Mission critical software: navigation, guidance, pilot
  - 5 interacting parallel tasks
  - 70000 LOC

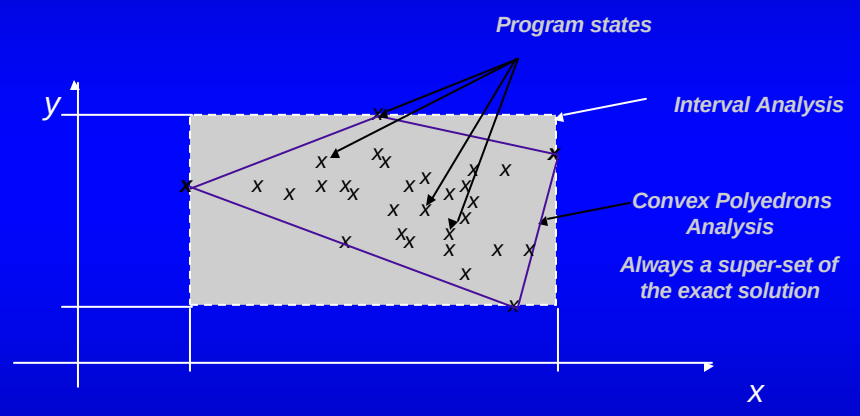
Customer References

- Atmospheric Reentry Demonstrator
  - Mission critical software: navigation, guidance, pilot
  - 3 interacting parallel tasks
  - 26000 LOC

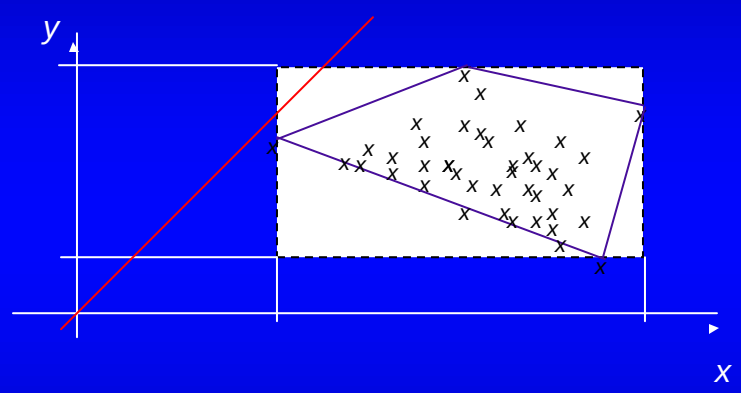
PolySpace Technologies

Background Technology

Abstract Interpretation



No Error

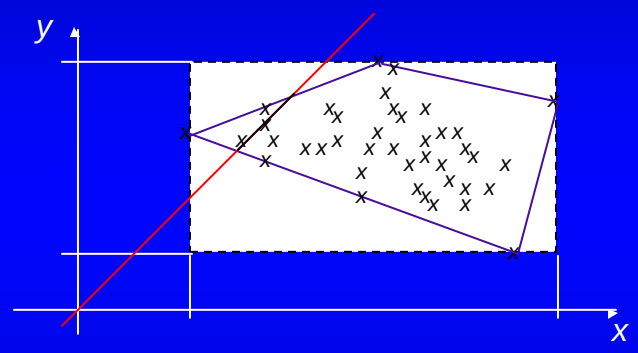


Expression: a = x / (x-y);

Correctness condition: x ¹ y

The intersection between the set of program states and the set of erroneous states is empty.

Potential Error

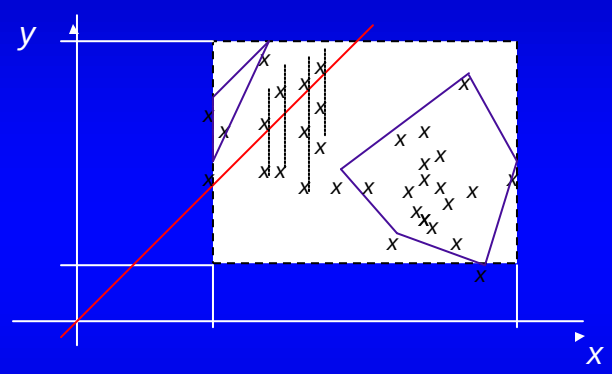


Expression : a = x / (x-y);

The intersection between the set of program states and the set of erroneous states is not empty.

A peculiar execution may lead to an error.

Optimizations



Several optimization algorithms are used to reach a high selectivity rate:

- *Polyedrons union*
- *Integer lattices*
- *Gröbner bases...*

Conclusion

- *Abstract Interpretation isn' t rocket science.*

QW2000 Paper 3T2

Mr. Robert Oshana
(ObjectSpace, Inc.)

Performance Engineering of an Embedded System Application

BACK TO QW2000 PROGRAM

Presentation Abstract

This paper presents lessons learned performance engineering a CPU throughput utilization estimate in a Raytheon Systems Company program developing a digital signal processing application concurrently with a next generation DSP-based array processor. Algorithmic performance and an efficient implementation were driving criteria for the program. As the processor was being developed concurrently with the software application a significant amount of the system and software development would be completed prior to the availability of physical hardware. This led to incorporation of performance engineering techniques into the development life-cycle. This article relates the mechanisms used to performance engineer the system and experiences with the approach.

About the Author

Robert Oshana is a Project Lead at ObjectSpace, Inc. ObjectSpace is a business-to-business integration (B2Bi) company whose products enable organizations to streamline their collaboration, communication and integration with trading partners over the Internet. He is responsible for leading software teams throughout all stages of business-to-business web-based development projects. Robert has masters degrees in electrical engineering, computer science, and business administration. He is also an adjunct professor at Southern Methodist University where he teaches several software engineering and E-Commerce courses in the Graduate Software Engineering program. He has over 30 publications in the areas of process improvement, software development methodology, software testing, and real-time systems.

BACK TO QW2000 PROGRAM

Performance Engineering of an Embedded System Application

Robert S. Oshana
Project Manager
ObjectSpace, Inc
roshana@objectspace.com
(972) 726-4530

1

Avoiding Disasters

- Expensive disasters can be avoided when system performance evaluation takes place relatively early in the software development lifecycle
- Applications will generally have better performance when alternative designs are evaluated prior to implementation

2

What is SPE ?

- A set of techniques for;
  - gathering data
  - constructing a system performance model
  - evaluating the performance model
  - managing risk of uncertainty
  - evaluating alternatives, and
  - verifying the models and results

3

Project Overview

- Digital signal processing application
- Software being developed concurrently with a next generation DSP-based array processor
- Algorithmic performance and an efficient implementation are driving criteria for the program
- A significant amount of the system and software development would be completed prior to the availability of physical hardware

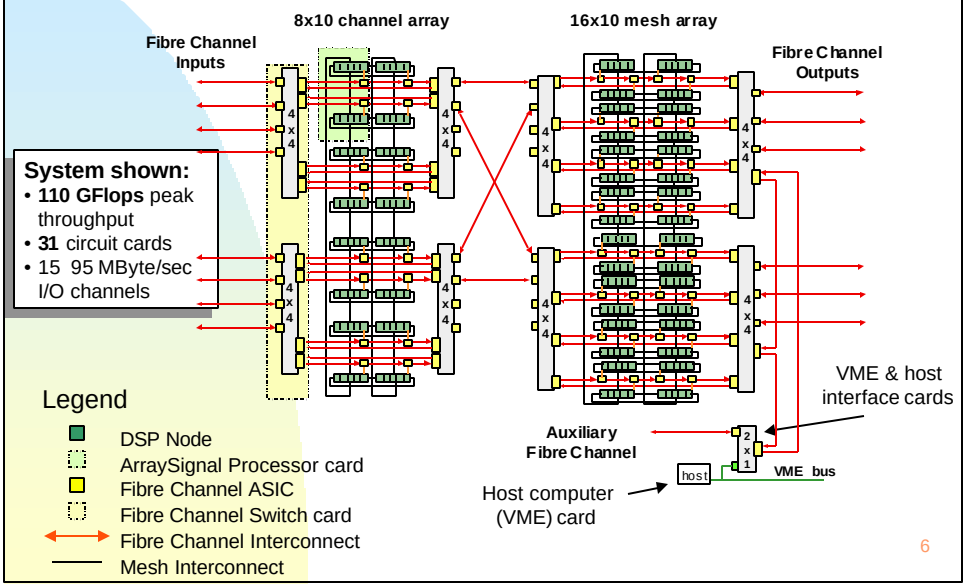
4

Project Overview

- SPE techniques were incorporated cross-functionally;
  - Systems engineering organization
    - responsible for developing the signal processing algorithms
  - Software and hardware engineering organizations
    - responsible for implementing the algorithms in an embedded real-time system
- Application is a large, distributed, multi-processing embedded system

5

DSP Array Architecture in Mesh Configuration



6

Risks

- Concurrent H/W - S/W development
- Development tools not immediately available
- Delivered system not meeting performance requirements was a serious concern
- Algorithm stream was being enhanced and revised as part of the development effort
- SPE critical to mitigating these risks.

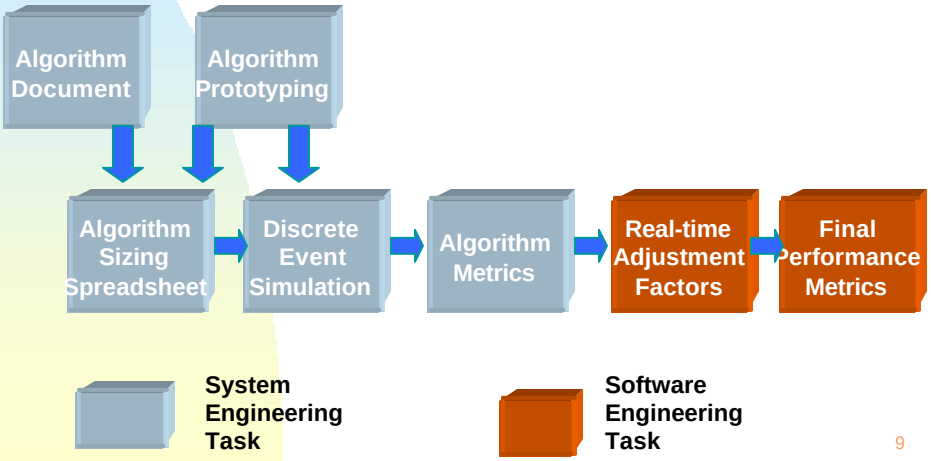
7

Information for SPE

- Workload
- Performance objectives
- Software characteristics
- Execution environment
- Resource requirements
- Processing overhead

8

Performance Metric Calculation Flow



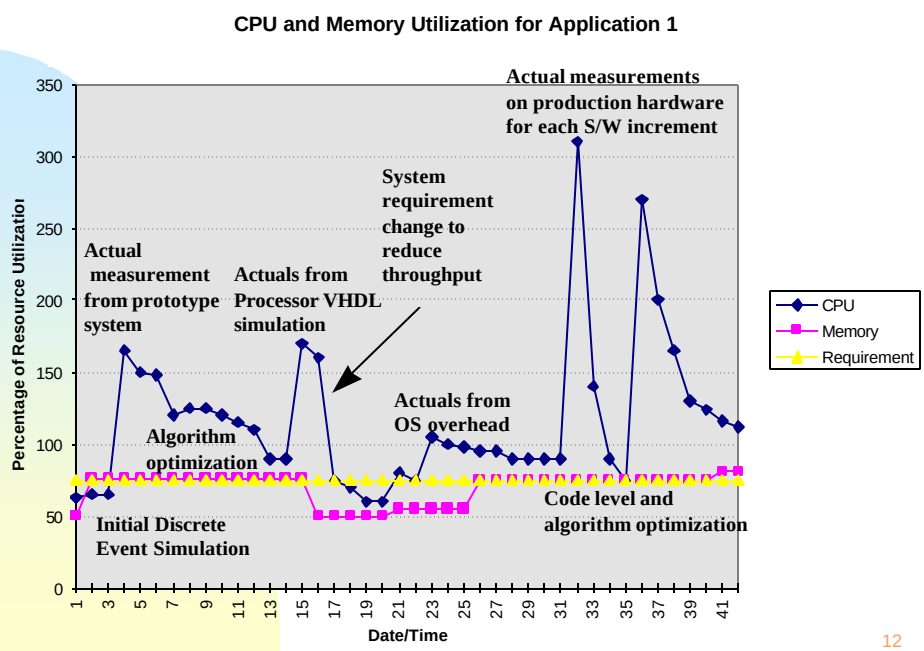
Factors that Influence Throughput

- The quantity of algorithms to implement
- Elemental operation costs (measured in processor cycles)
- Sustained throughput to peak throughput efficiency
- Processor family speed-up

Factors that Influence Memory

- Size and quantity of intermediate data products to be stored
- Dynamic nature of memory usage
- Bytes/data product
- Bytes/instruction
- size and quantity of input and output buffers based on worst case system scenarios (workloads)

11



12

Chronology of CPU Throughput Reduction for Application 1

Increase or Decrease in Metric	Explanation
Initial discrete event simulation was used as the starting point for the metric estimation	Discrete event simulation was built using algorithm cycle estimations and first order modeling for task iterations due to context switching, etc
Measurement on prototype C40 based array	Prototype code was ported to a C40 based DSP small scale array and measured. The measurement was then scaled based on the speedup of the C67 based DSP full scale array
Algorithm level optimization	Algorithms were made more efficient using algorithm re-structuring methods and reducing complexity in other areas of the algorithm stream.

13

Chronology of CPU Throughput Reduction for Application 1

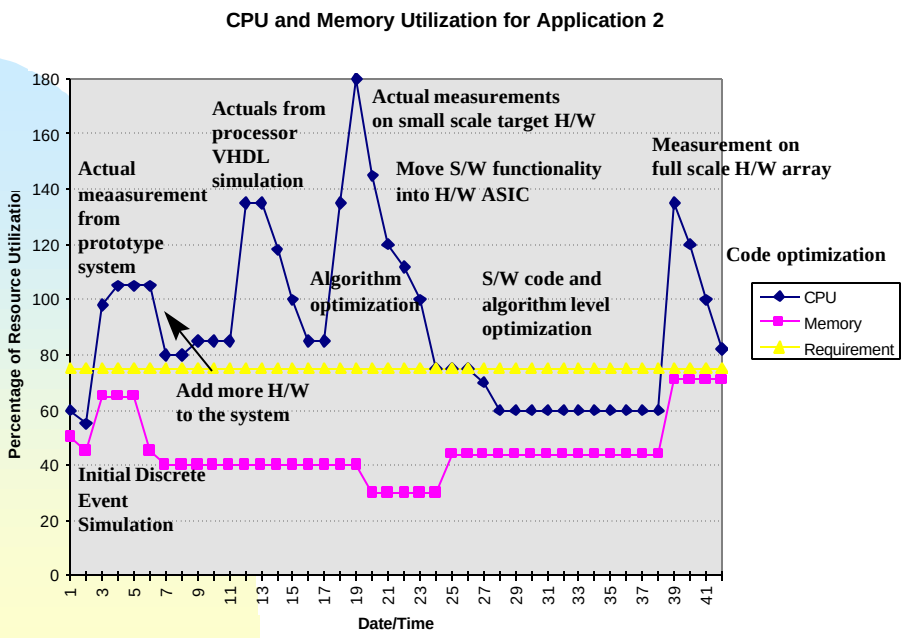
Processor VHDL measurement	Big increase in throughput measurement was due to unexpected high cost of accessing data from external (off-chip) memory. Several benchmarks were performed and scaled to the entire application
System level requirement change	Project decision was made to change a system level parameter. This caused significant algorithm restructuring and was an unpopular decision with the customer
OS level overhead measured	Because the processor was new, the COTS OS was not immediately available. This point indicated the first time to run the application in a multi-tasking environment with the OS.

14

Chronology of CPU Throughput Reduction for Application 1

Actuals on production hardware array of DSPs for each software increment	The production code was initially developed without code optimization techniques in place (Make it work right and then make it work fast). Initial measurement for the full algorithm stream was not entirely optimized when we first took the measurement
Continued code and algorithm level optimization	Dedicated team in place to work code optimization and other algorithm transformation techniques to reduce CPU throughput (i.e. taking advantage of symmetry in the algorithms and innovative techniques to reduce communications between DSPs which were expensive)

15



16

Chronology of CPU Throughput Reduction for Application 2

Increase or Decrease in Metric	Explanation
Initial discrete event simulation was used as the starting point for the metric estimation	Discrete event simulation was built using algorithm cycle estimations and first order modeling for task iterations due to context switching, etc
Measurement on prototype C40 based array	Prototype code was ported to a C40 based DSP small scale array and measured. The measurement was then scaled based on the speedup of the C67 based DSP full scale array
Add more hardware to the system	Number of DSP nodes was increased by adding more DSP boards. Good hardware design made scalability relatively easy.

17

Chronology of CPU Throughput Reduction for Application 2

Processor VHDL measurement	Big increase in throughput measurement was due to unexpected high cost of accessing data from external (off-chip) memory. Several benchmarks were performed and scaled to the entire application
Algorithm optimization	Because of the nature of the algorithms, we were able to significantly cut CPU throughput utilization by restructuring the algorithms to pipeline the major loops of the algorithm stream.
Actual measurement on small scale target Hardware	In our hardware/software co-design effort, we did not have full scale hardware until late in the cycle. Initial benchmarking for this application was performed on a single node prototype DSP card.

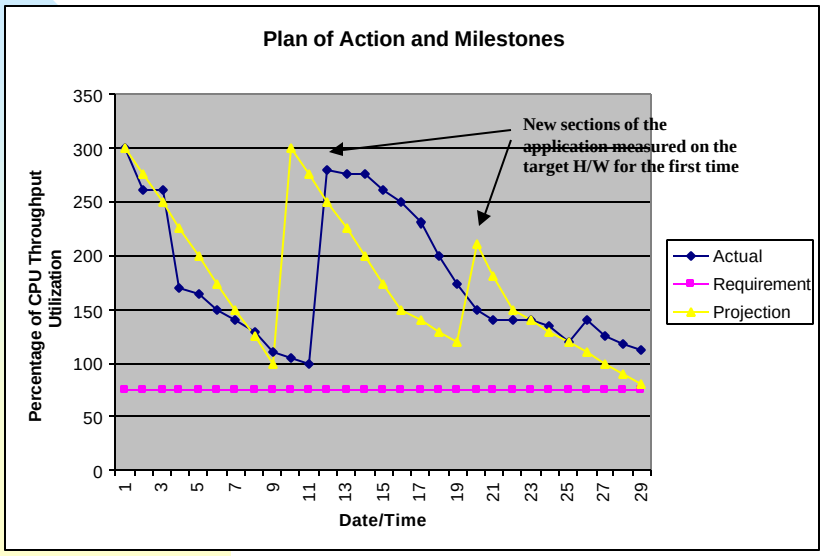
18

Chronology of CPU Throughput Reduction for Application 2

Move software functionality into hardware ASIC	Decision was made for risk mitigation purposes to move part of the algorithm stream into a hardware ASIC in another sub-system, saving significant CPU cycles in the application software.
Software code and algorithm level optimization	Dedicated team in place to work code optimization and other algorithm transformation techniques to reduce CPU throughput
Measurement on full scale hardware	Measuring the application CPU throughput on the full scale hardware showed that we had under estimated the overhead for communication among all the array nodes. We developed a tailored comm API to perform intra-node communications more quickly.

19

Plan of Action and Milestones for Application 1



20

Tools Identified in Performance Plan and Errors Resolved

Tool	Error Factors Resolved
Code Generation Tools (Compiler, Assembler, Linker)	Compiler efficiency Quality of generated assembly code Size of load image
Instruction Level Processor Simulator	Utilization of dual processor pipelines Cycle counts for elemental operations
Cycle-accurate Device Level VHDL Model	Effect of external memory access times Instruction Caching effects Device resource contention between processor and DMA channels
Single DSP Test Card	Validate VHDL results Runtime interrupt effects
Multi-DSP Test Card	Inter-processor communication resource contention effects

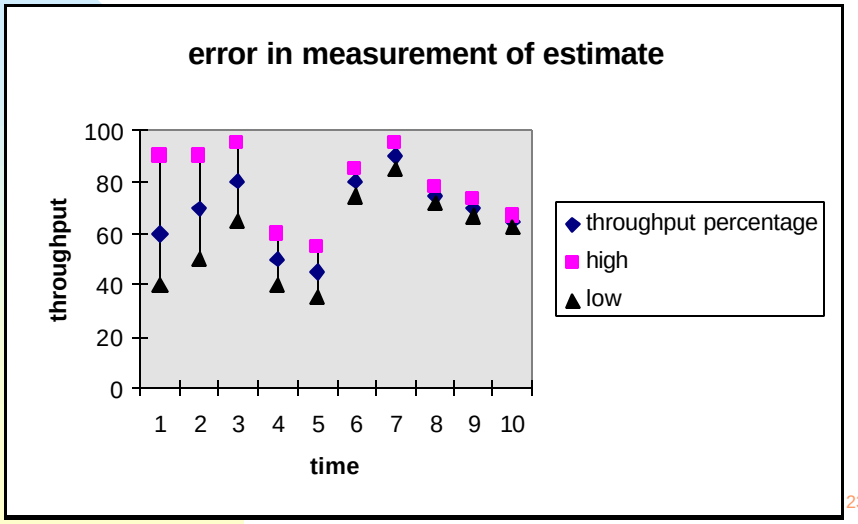
21

Make it Work Right - Then Make it Work Fast

- Optimization learning curve
- Complex algorithms that are hard to understand and comprehend
- Must profile the algorithm stream and optimize the “hot spots”

22

Improvement in Accuracy in Estimate over time



23

Conclusions and Lessons Learned

- Prototype early in the development life cycle
- Ignore the processor marketing information
- Analyze the functions executing most often
- Don't ignore the interfaces
- Benchmarks of discrete algorithms don't scale well to real-time systems
- Keep management informed
- Budget accordingly

24

Where to find more

- IEEE Computer, May 2000, "Performance Engineering of an Embedded System Application"

25

Speaker Bio

- **Robert Oshana has over 17 years of software project management experience. He has managed various software development efforts in both the defense and commercial industries. His current assignment is Project Manager for ObjectSpace, Inc., a business-to-business integration (B2Bi) company whose products enable organizations to streamline their collaboration, communication and integration with trading partners over the Internet. Robert is also an adjunct professor at Southern Methodist University where he teaches several courses in the Graduate Software Engineering program, including Software Testing and Quality Assurance, as well as an E-Commerce curriculum in the graduate Telecommunications program, and short courses in the Distance Education program. Robert has over 30 publications in the areas of software process improvement, software testing, software performance engineering, and real-time systems. He has masters degrees in Electrical Engineering, Computer Science, and Business Administration.**

26

Performance Engineering of an Embedded System Application

Robert S. Oshana
ObjectSpace, Inc.

roshana@objectspace.com

Abstract: This paper presents lessons learned performance engineering a CPU throughput utilization estimate in a defense program developing a digital signal processing application concurrently with a next generation DSP-based array processor. Algorithmic performance and an efficient implementation were driving criteria for the program. As the processor was being developed concurrently with the software application a significant amount of the system and software development would be completed prior to the availability of physical hardware. This led to incorporation of performance engineering techniques into the development life-cycle. This article relates the mechanisms used to performance engineer the system and experiences with the approach.

Introduction and Project Description

Expensive disasters can be avoided when system performance evaluation takes place relatively early in the software development lifecycle. Applications will generally have better performance when alternative designs are evaluated prior to implementation. Software Performance Engineering (SPE) is a set of techniques for gathering data, constructing a system performance model, evaluating the performance model, managing risk of uncertainty, evaluating alternatives, and verifying the models and results [1,2]. SPE also includes strategies for the effective use of these techniques. Software performance engineering concepts have been incorporated into a Raytheon Systems Company program developing a digital signal processing application concurrently with a next generation DSP-based array processor. Algorithmic performance and an efficient implementation were driving criteria for the program. As the processor was being developed concurrently with the software application a significant amount of the system and software development would be completed prior to the availability of physical hardware. This led to incorporation of SPE techniques into the development life-cycle. The techniques were incorporated cross-functionally into both the systems engineering organization responsible for developing the signal processing algorithms and the software and hardware engineering organizations responsible for implementing the algorithms in an embedded real-time system.

The customer and application are proprietary. The application is a large, distributed, multi-processing embedded system. One of the sub-systems consists of two large arrays of digital signal processors (DSP). These DSPs execute a host of signal processing algorithms (various size FFTs and digital filters, and other noise removing and signal enhancing algorithms). The algorithm stream being implemented included both temporal decomposition of the processing steps as well as spatial decomposition of the data set. The array of mesh-connected DSPs was used because the spatial decomposition required mapped well to the architecture and the required throughput of the system drove the size of the array (Figure 1). The system is a data driven application, using interrupts to signal the arrival of the next sample of data. This system is a “hard” real-time system in the sense that missing one of the data processing deadlines results in a catastrophic loss of system performance.

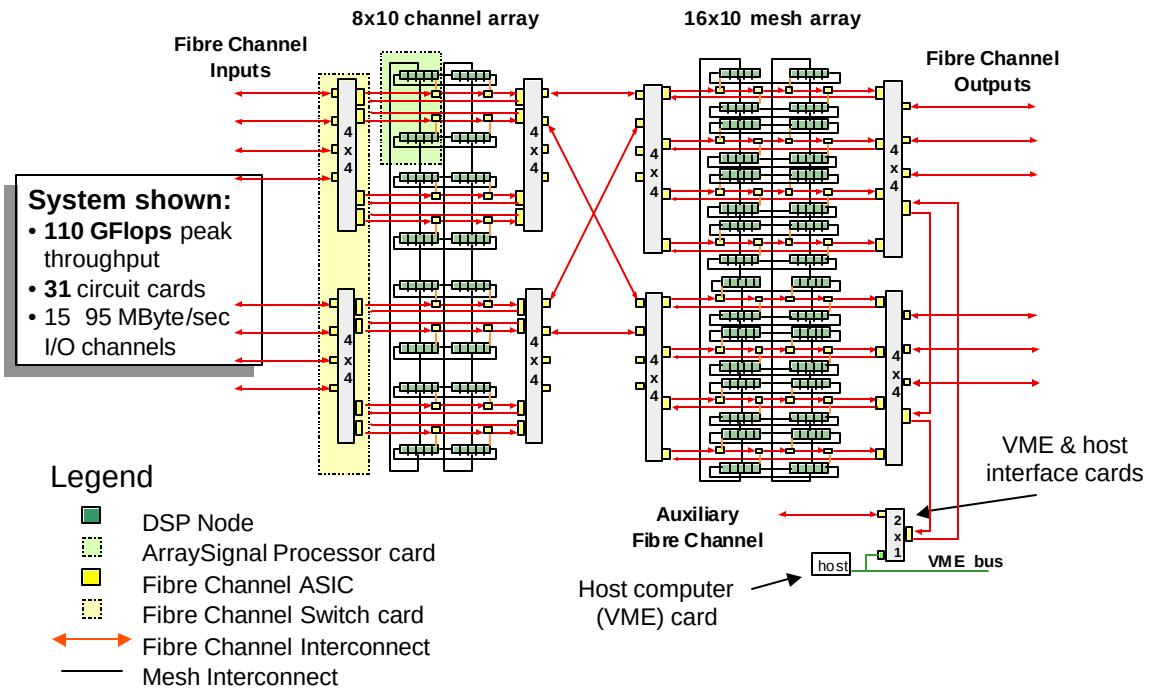


Figure 1. DSP Array Architecture in Mesh Configuration

The concurrent development of a new DSP-based array processor using a next generation DSP device still being developed and whose software development tools were not yet available posed a significant challenge to the development team. The risk of the delivered system not meeting performance requirements was a serious concern. To further complicate matters the algorithm stream was being enhanced and revised as part of the development effort. The incorporation of SPE techniques into the development processes of the various functional organizations was deemed critical to mitigating these risks.

The issue of performance was addressed from the inception of the program throughout its development phases. The main measures of performance are captured in three metrics:

- processor throughput,
- memory utilization,
- I/O bandwidth utilization.

These are the metrics of choice because monthly reporting of these metrics is a customer requirement for the program. Initial estimates of these metrics were made prior to the start of the program and updated monthly during the development effort. Uncertainties associated with key factors driving these estimates were identified. Plans for resolving these uncertainties during the development effort were developed and key dates identified. Updating the metrics and maintaining the associated risk mitigation plans was a cross-functional collaborative effort involving systems engineering, hardware engineering and software engineering.

Initial Performance Estimates and Information Requirements
The information generally required for a SPE assessment are [1];

Workload; The expected use of the system and applicable performance scenarios. We chose performance scenarios that provided the array processors with the worst case data rates. These worst case scenarios were developed by interfacing with the users and our system engineers.
Performance objectives; This represents the quantitative criteria for evaluating performance. We used CPU utilization, memory utilization, and I/O bandwidth because of the customer requirement that we report on these monthly.
Software characteristics: this describes the processing steps for each of the performance scenarios and the order of the processing steps. We had accurate software characteristics due to an earlier prototype system using a similar algorithm stream. We also had an Algorithms Description document detailing the algorithmic requirements for each of the functions in the system. From this a discrete event simulation was developed to model the execution of the algorithms.
Execution environment; this describes the platform on which the proposed system will execute. We had an accurate representation of the hardware platform due to involvement in the design of the I/O peripherals of the DSP as well as some of the DSP core features as well. The other hardware components were simulated by the hardware group.
Resource requirements; this provides an estimate of the amount of service required for the key components of the system. Our key components were CPU, memory and I/O bandwidth for each the DSP software functions.
Processing overhead; this allows us to map software resources onto hardware or other device resources. The processing overhead is usually obtained by benchmarking typical functions (FFTs, filters) for each of the main performance scenarios.

CPU throughput utilization was the most difficult metric to estimate and achieve. Therefore, the rest of this paper will focus primarily on the methods we used to develop an accurate estimate for the CPU throughput utilization metric..

Developing the Initial Estimate
The process used to generate the initial performance metric estimates is shown in Figure 2. This flow was used throughout the development effort to update the metrics. The algorithm stream is documented in an algorithm document. From this document the systems engineering organization developed a static spreadsheet model of the algorithm stream which provided estimates of throughput and memory utilization for each of the algorithms in the algorithm requirements document. The spreadsheet includes allowances for operating system calls and inter-processor communication. The systems engineering organization used a current generation DSP processor to perform algorithm prototyping and investigation activities. The results of this work influenced

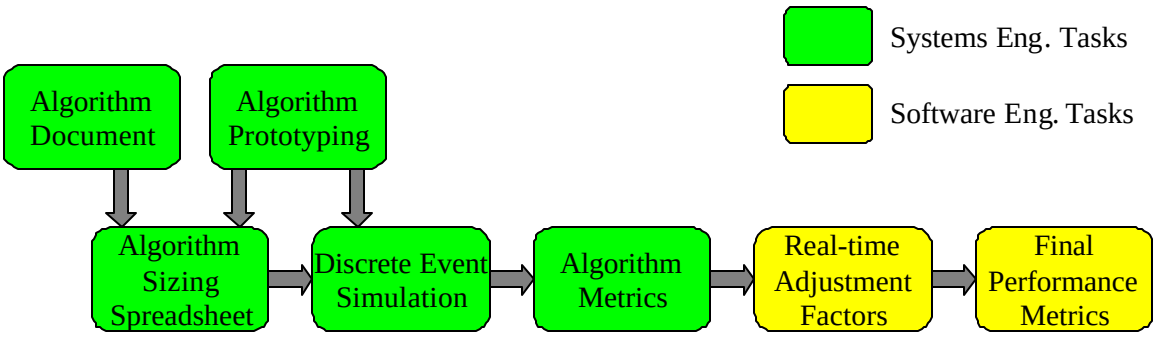


Figure 2. Performance Metric Calculation Flow.

algorithm implementation decisions and were used to develop the discrete event simulations used to estimate the performance metrics. A discrete event simulation was used to model the dynamic performance of the algorithm stream. The simulation model included allowances for operating system task switches and associated calls. The initial algorithm spreadsheet of resource allocations for each algorithm and discrete event simulation processes provide the system engineering 'algorithm' performance metrics. At this point the metrics reflect the throughput, memory, and I/O bandwidth required to perform the algorithms defined in the algorithm document and implemented using the prototype implementations. The software engineering organization then updates the performance metrics to reflect the costs of embedding the algorithm stream in a robust, real-time system. These metric adjustments include the effects of system-level real-time control, built-in-test, formatting of input and output data, and other 'overhead' functions (processing overhead) required for the system to work. The results of this process are the reported processor throughput, memory utilization and I/O utilization performance metrics.

Tracking the Metrics

The software development team is responsible for estimating and reporting metrics related to processor throughput and memory. These metrics are reported periodically to the customer, and are used for risk mitigation. Reserve requirements are also required to allow for future growth of functionality (our reserve requirement was 75% for CPU and memory). Throughout the development life cycle, these estimates varied widely based on the different modeling techniques used in the estimation and hardware design decisions which influenced the amount of hardware available to execute the suite of algorithms as well as measurement error. Figure 3 shows the metric history for throughput and memory for the first array processor application. There is a wide variability in the throughput throughout the life cycle, reflecting a series of attempts to lower the throughput estimate followed by large increases in the estimate due to newer information. In Figure 3, the annotations describe the increases and decreases in the estimate for the CPU throughput measurement.

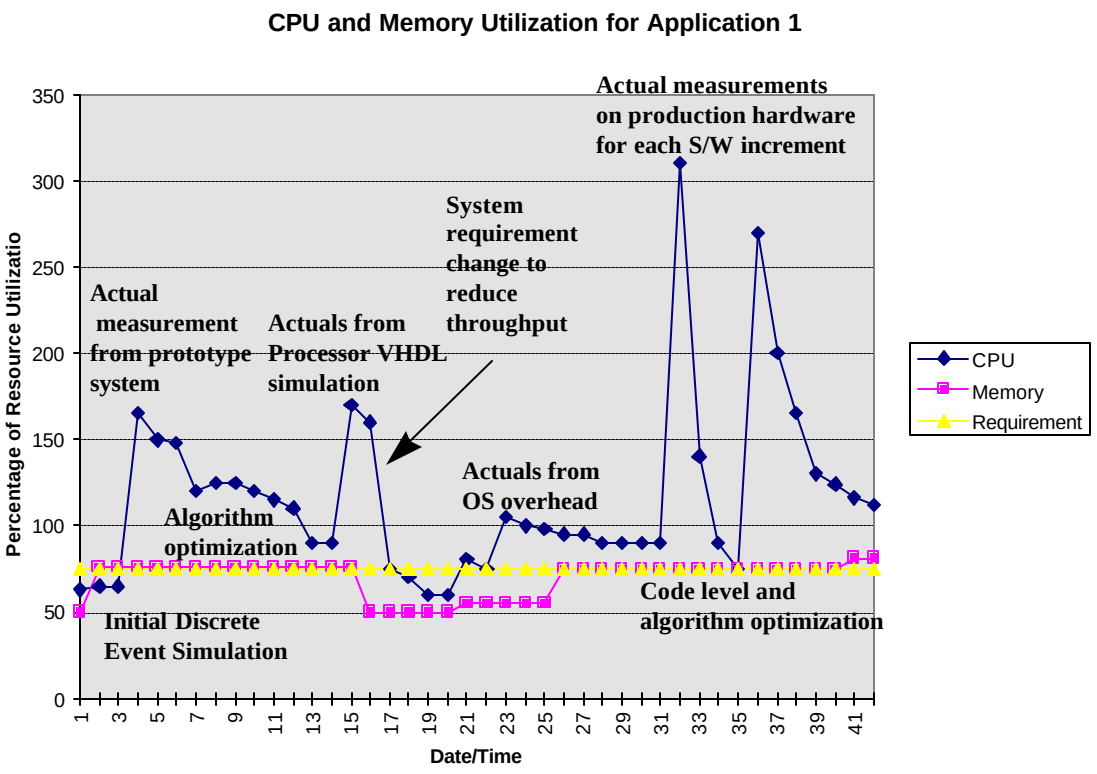


Figure 3. Resource Utilization metric for Application1

The first large increase in the estimate came as a result of implementing the algorithm stream on a prototype current generation processor. These measurements were then scaled based on the anticipated performance of the next generation processor. An effort was then undertaken to optimize the implementation of the algorithm stream to lower the throughput estimate.

The next unexpected increase came from running representative benchmarks on the next generation cycle accurate simulator. This simulation allowed us to estimate the true cost of external memory accesses, pipeline stalls, and other processor characteristics that increased the cost of executing the algorithms. These results led the development teams to undertake another significant effort to optimize the algorithm stream for real-time operation. The main techniques undertaken during this phase included instrumentation of the Direct Memory Access (DMA) to stage data on and off chip, re-structuring of code to allow critical loops to pipeline, assembly language implementation of critical algorithm sections, and efficient use and management of on-chip memory where memory access time is much shorter.

The representative benchmarks showed us that we could reduce the throughput using code-level optimization techniques (use of on-chip memory, pipelining of important loops, etc) but we were still in danger of not meeting our overall throughput requirement. It was at this time that a system requirement was modified to reduce throughput.

The third major increase came when we measured the full application on the target array of DSPs. The main reason for the increase was due to the fact that many of the algorithms were not optimized. Only a small percentage of algorithm were benchmarked on the processor VHDL simulator (representative samples of the most commonly used algorithms such as the FFTs and other algorithms called inside major loops in the code). The software group still needed to employ the same optimization techniques for the remaining code for each of the software increments being developed. By this time the optimization techniques were familiar to the group and the process went fairly fast.

The lifecycle throughput estimates for the second array processor application is shown in Figure 4. A similar pattern in the reported numbers is seen here due to the same basic issues.

Once again the initial discrete event simulation proved to be inaccurate and the prototype system measurements were much higher than anticipated due to overly aggressive estimates of the CPU throughput, failure to account for realistic overhead constraints, etc. A long process of code and algorithm optimization was able to bring the estimate back down close to the goal before the VHDL simulation measurements uncovered some other areas that made us increase the estimate. The increase in the estimate in this application resulted in several risk management activities to be triggered;

The estimate in month 5 was high enough and was made early enough in the program schedule that the program was able to add more hardware resources to reduce the algorithm distribution and lower the throughput estimate. This was made at the expense of more power and cooling requirements as well as more money for the hardware (no new designs were required, just more boards). These increases in power and cooling had to be offset by sacrifices elsewhere to maintain overall system requirements on these parameters.

The measurement in month 19 was caused consternation among the managers as well as the technical staff. Although we felt continued optimization at the code level would drop the number significantly, meeting the application requirement of 75% CPU throughput (25% reserved for growth) would be hard to accomplish.

One contributor to the CPU throughput estimate increase was a result of an under-estimation of a worst case system scenario which led to an increase in data rate for the processing stream,. This resulted in several algorithm loops being executed more frequently which increased the overall CPU utilization.

The decision was made to move some of the software functionality being done in the DSPs into a hardware ASIC to reduce the throughput significantly (there were a sufficient number of unused gates in the ASIC to handle the increased functionality). With this decision coming so late on the development cycle, however, significant re-design and re-work of the ASIC and the interfaces was required which was extremely expensive for the hardware effort as well as delays in the system integration and test phase.

The last increase in CPU utilization was a result of scaling the algorithms from the small (single node) DSP benchmark to the full array of DSPs. The increase was mainly due to a mis-estimation in the overhead associated with inter-processor communication. Once again, the development teams were faced with the difficult challenge of demonstrating real-time operation given these new parameters. At this late date in the development cycle, there are not many options left for the system designers. The main techniques used at this point to reduce the throughput estimate were additional code optimization, assembly language implementation of additional core algorithms, additional limited hardware support, and a significant restructuring of the algorithm control flow to circumvent the use of slow operating system functions. For example, we eliminated some of the robustness in the node to node communication API in order to save valuable CPU cycles.

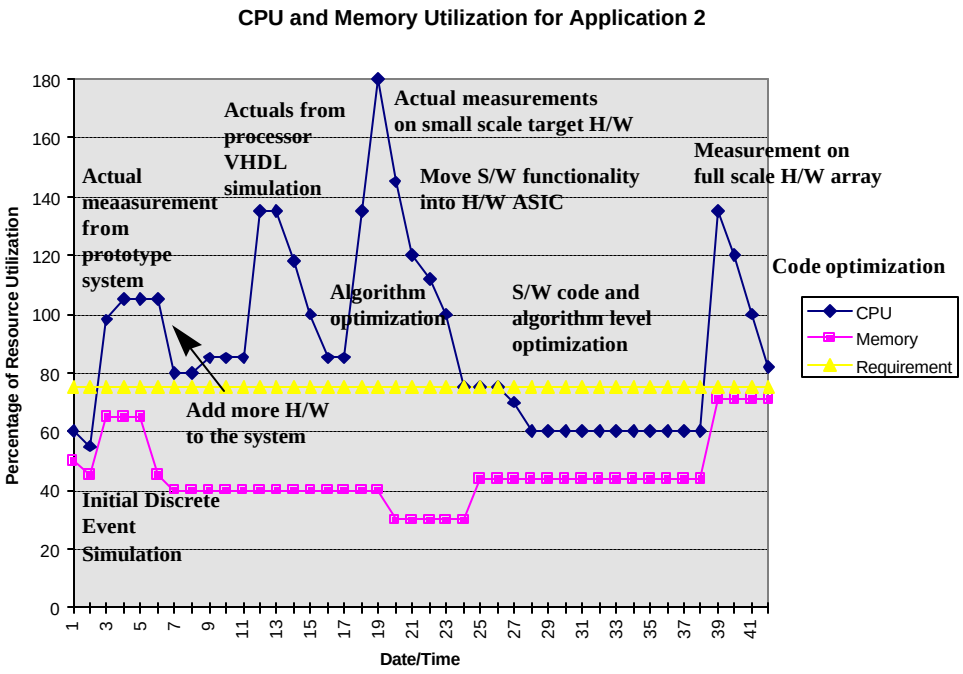


Figure 4. Resource Utilization metric for Application 2

Conclusions and Lessons Learned

There were several lessons learned in this experience;

- *Prototype early in the development life cycle*; several of the surprises we encountered could have been uncovered earlier if the proper level and type of prototyping was performed.
- *Ignore the processor marketing information*; Most processors will never achieve the throughput proposed in the literature.
- *Analyze the functions executing most often*; these areas are where the hidden cycles can come back to haunt a development team.
- *Don't ignore the interfaces*; real time systems carry an inherent "overhead" that never seems to be accounted for in throughput estimates.
- *Benchmarks of discrete algorithms don't scale well to real-time systems*; benchmarking an individual algorithm inherently implies that algorithm has complete control and use of all the processor resources including internal and external memory, the DMA controller, and other system resources.
- *Keep management informed*; as we approach the completion of the code level optimization effort, it appears the model we established early in the project was a relatively accurate estimate.
- *Budget accordingly*; the two pass approach of functional correctness followed by code optimization will take more time and more resources to accomplish.

REFERENCES

1. Smith, Connie U. "Performance Engineering for Software Architectures", 21st Annual Computer Software and Applications Conference, 1997, p 166-167.
2. Baker, Michelle and Warren Smith, "Performance Prototyping: A Simulation Methodology for Software Performance Engineering", Proceeding of the Computer Systems and Software Engineering", 1992, p 624-629.
3. Oshana, Robert , "Rate Monotonic Analysis Keeps Real Time Systems On Track", EDN, September 1, 1997.
4. Liu, C and J Layland, "Scheduling algorithms for multiprogramming in a hard real time environment", Journal of the Association for Computing Machinery, January 1973.
5. Obenza, Ray, "Rate monotonic analysis for real-time systems", March 1993

QW2000 Paper 3A2

Ms. Linda Hayes
(WorkSoft)

Table Driven Testing: Making Automation Accessible

BACK TO QW2000 PROGRAM

Presentation Abstract

This presentation will explain how to use advanced scripting techniques including modularity, reusability, encapsulation, indirection and data-driven design to make automation easily accessible to domain experts with a minimum of training and technical expertise. The ultimate goal is to make test automation available, persistent and consistent across the organization, over time and change.

About the Author

Linda G. Hayes BBA, CPA, MS, JD

Linda holds degrees in accounting, tax and law, is a CPA and member of the Texas State Bar. She is the founder of three software companies including AutoTester, where she pioneered automated test tools for the PC. Her new company, WorkSoft, offers the next generation of enterprise-level test automation.

A frequent industry speaker and award-winning author, she publishes the monthly Quality Quest column for Datamation, wrote the Automated Testing Handbook and co-edited *Dare to be Excellent* with Alka Jarvis on best practices in the software industry. Her article "Quality is Everyone's Business" won the Most Significant Contribution award from the Quality Assurance Institute and was published as part of the Auerbach Systems Development Handbook.

BACK TO QW2000 PROGRAM

Table-Driven Testing:

Making Automation Accessible

Quality Week 2000
Linda G. Hayes

Business Case for Test Automation

- Market pressure drives schedules
  - Product cycles are accelerating
- Budget pressure constrains funds
  - Time, resources, skills are limited
- Business risk demands quality
  - Testing is critical and complex
- Automation is best solution
  - Reduce time, cost, increase quality

Obstacles

- Script skills are scarce, expensive
  - Tools are specialized languages
- Development effort is prohibitive
  - 5 to 10X manual effort
- Maintenance offsets productivity
  - Applications are moving targets
- **Undocumented, non-transferable**
  - **Limited useful life**

Automation Evolution

- Capture/Playback
- Data-Driven
- Table-Driven
- Test Repository

Capture/Playback

- Most commonly recognized technique
  - Easy to demonstrate, sell
- Perform test manually, capture steps
  - Implies no difference in approach
- Script is generated automatically
  - Appears to hide script language

Capture/Playback Example

Select File>>Open
Select ListBox Files "Accounts"
Enter TextBox Name "Joe C. Customer"
Enter TextBox Address "1234 Main"
Enter TextBox City "Anytown, USA"
Select RadioButton Type "Individual"
Click PushButton "OK"
Verify Message "Account Added"

Capture/Playback Advantages

- Little training, setup time
  - Useful in a crisis
- Develop tests on the fly
  - Requires little process
- Captures audit trail
  - Recreate manual steps

Capture/Playback Disadvantages

- Late in development cycle
  - Software must be working
- Requires manual capture
  - Limited leverage
- May not replay reliably
  - May still require some scripting
- Difficult to maintain
  - Offsets productivity

Data Driven

- Inputs and outputs stored as data
  - Each record is a test case
- Sequence of actions scripted
  - Variables substituted for data values
- Capture manually then modify
  - Add variables
  - Add file processing
  - Logic and branching

Data Driven Script Example

```
Open File "TestData"
Select File>>Open
Select ListBox Files "Accounts"
Next: Read Record until EOF:
  Enter TextBox Name [Name]
  Enter TextBox Address [Address]
  Enter TextBox City [City]
  Select RadioButton Type [Type]
  Click PushButton "OK"
  Verify TextBox Message [Status]
EOF: End of File
```

Data Driven Data Example

Name	Address	City	Type	Message
Joe C. Customer	1234 Main	Anytown, USA	Individual	Account Added
Mary B. Buyer	9876 Elm	Othertown, USA	Wholesale	Account Added
Little Shoppe	12 Main Mall	Shoptown, USA	^	Type Required

Data Driven Advantages

- Create test cases earlier
  - Data values defined in advance
- Create tests in utility of choice
  - Database, spreadsheet, editor
- Leverage
  - Import, generate data
- Reduced maintenance
  - Single script per process

Data Driven Disadvantages

- Technical tool skills required
  - Use of variables, external text files
  - Logic to handle data variations
- Scripts still recorded
  - Delays test process
- Data file management needed
  - Data files are part of test repository

Table Driven

- Test cases stored as data
  - Standard file format for all tests
  - Each step is a separate record
- Single main processing script
  - Single point synchronization
- Generic function scripts
  - Input, verify functions for each class
  - Shared across all instances

Table Driven Data Example

Window	Object	Action	Value
Main	Menu	Select	Accounts
Accounts	Name	Enter Text	Joe C. Customer
Accounts	Address	Enter Text	1234 Main
Accounts	City	Enter Text	Anytown, USA
Accounts	Type	Select Radio	Individual
Accounts	OK	PushButton	
Accounts	Message	Verify Text	Account Added

Table Driven Main Script

```
Open file TESTDATA
  NEXT: Read file TESTDATA until EOF
    Does @Window have focus?
      If no, wait for timeout
        If timeout, call Recover
    Does @Object have focus?
      If no, set focus to @Object
        Does @Object have focus?
          If no, Call LogError
   Call @Function
END: End of file
```

Table Driven Functions

Menu Select script

```
Does menu item Value exist?
        If no, Call LogError
Is menu item Value enabled?
        If no, Call LogError
Select menu item Value
Resume
```

Enter_Text

```
Press [Shift-End]
Enter Value
Resume
```

Select Radio

```
Select Radio @Object, Value
Resume
```

Verify Text

```
Get value of @Object into ActualValue
Verify ActualValue=Value
        If no, call LogError
Resume
```

Table Driven Advantages

- No tool expertise to develop tests
  - Test cases easy to write, read
- Develop test cases, scripts earlier
  - Ready to execute when software is
- Minimized maintenance
  - Single occurrence of each function
- Portable architecture
  - Across applications
  - Across tools

Table Driven Disadvantages

- Need technical skills
  - Identify object classes, actions
- Need tool skills
  - Develop and maintain scripts
- Need data repository
  - Tie tests, data together

Test Repository

- Analyst friendly front-end
  - Short learning curve
- Open architecture
  - Works with any tool
- Centralized repository
  - Tests, data, results
- Documented, maintainable
  - Easy to transfer, manage

Certify™ Implementation

Overview &
Demonstration

Certify™ Modules

- **Organize**
  - Environments, Applications, Versions, Users
- **Plan**
  - **R**equirements, Schedules, Cycles, Processes
- **Define**
  - Map, Variables, Data
- **Perform**
  - Develop, Execute, View Results
- **Analyze**
  - Coverage, Status, Trends

Execution Engine

- Function library
  - Written in tool of choice
  - Mapped to object classes
  - Linked to repository
- Infrastructure
  - Synchronization
  - Error-handling
  - Recovery
  - Reporting

Summary

- Evolution of Test Automation
  - More sophisticated use of tools
  - Easier to use for testers
- From Tool to Application
  - Tests stored in database, not scripts
  - Organized approach instead of do-it-yourself
- Focus on Tests, not Tool

QW2000 Paper 3W2

Dr. B.M. Subraya & Mr. S. V. Subrahmanya (Infosys)

Performance Testing: A Methodical Approach to E-Commerce Applications

BACK TO QW2000 PROGRAM

Presentation Abstract

Performance of many Web sites depends on the load on the site at peak time under varying conditions. Traditionally, client-server testing does not ensure the performance of Web sites, as there are unknown users with many uncertainties. Under these circumstances performance testing must be carried out in such an environment which reflects the real users. Since it is difficult to provide such an environment before the deployment of the application, it is more appropriate to follow a methodical approach that ensures the required performance under varied circumstances.

1.0 Introduction: The rapid rise of Web-based E-commerce applications has led to a plethora of design and development tools designed to minimize the time to market for many e-business applications. Designing and developing, however, are only two phases of application development. E-commerce applications demands rapid development and deployment. Testing, a critical third phase occupies the dominant role in the life cycle of the E-commerce application.

Successful e-business applications testing depends on a comprehensive test plan, functionality and performance testing by using incrementally developed scripts, and using these scripts to test under varying assumptions and parameters. This combination provides a thorough and methodical test process.

The first step in testing the e-business architecture is to test the application layer. This type of testing is often performed early in the lifecycle, before the actual GUI for the application is built, to ensure that the essential foundation is present and sound before you add more complex application components. During the deploying an e-business application, one of the most important issue need to be addressed is application's performance. Unlike traditional client/server applications, where user loads can be easily predicted, performance problems are inherent to e-business systems, where the volume of users and traffic levels can change in a matter of seconds.

2.0 What is Performance Testing?

Performance testing is directly reflect the behavior of the complete Web site. Visitors expect the fast response with in a short period of time. Therefore, a rigorous performance testing must be carried out on each site. Performance testing can be viewed as a "Black Box" testing, which focuses on application and system behavior from outside, with no knowledge of the program code that supports the system. The main objectives of the performance testing is:

* Maximum number of concurrent users that can be supported while offering "acceptable performance."
* Maximum number of concurrent users that can be supported prior to causing a system failure.
* Location of bottlenecks within the application architecture.
* Impact of a software or hardware change on the overall performance of the application.
* Scalability issues

The performance testing is done by recording or scripting the actions that real users perform, and then playing those actions back against the System Under Test (SUT), in an automated and controlled manner. Performance testing also depends on certain issues like, Graphics, Load Time and Connection Methods.

About the Authors

Dr. Subraya B.M. currently working as Consultant to Education and Research department of Infosys Technologies Limited, Bangalore, India. Before he comes to Infosys on sabbatical, he was working as Professor and head of Computer Centre, at S.J. College of Engineering, Mysore, a reputed engineering college in southern part of India, since more than 20 years.

Dr. Subraya holds Ph.D from Indian Institute of Technology, Delhi from Computer Science & Engineering Department, in the area of hardware verification. He has guided many projects at graduate and under graduate level. His area of interest includes Software testing, Operating System and distributed databases. He can be reached at subraya@inf.com.

Mr. S.V. Subrahmanya holds B.E degree in Electrical Engineering from University of Bangalore, Bangalore, India and M.Tech in Computer Science from IIT, Kharagpur. He is currently working as Project Manager at Infosys Technology limited, Bangalore. He started his carrier as programmer then moved to academics where in he worked as Assistant Professor in the department of Computer Science & Engineering, BMS College of Engineering, Bangalore. Prior to joining to Infosys he was working at Ashok Leyland Information Technology as Asst project Manager.

His areas of interest include Enterprise Application Integration(EAI), E-commerce End-to-End testing and Software Engineering. He can be reached at subramanyah@inf.com.

BACK TO QW2000 PROGRAM

PERFORMANCE TESTING: A METHODICAL APPROACH TO E-COMMERCE APPLICATIONS

Dr. SUBRAYA BM
subraya@inf.com
Education & Research Deptt.
ITL Infosys, Bangalore

SESSION PLAN

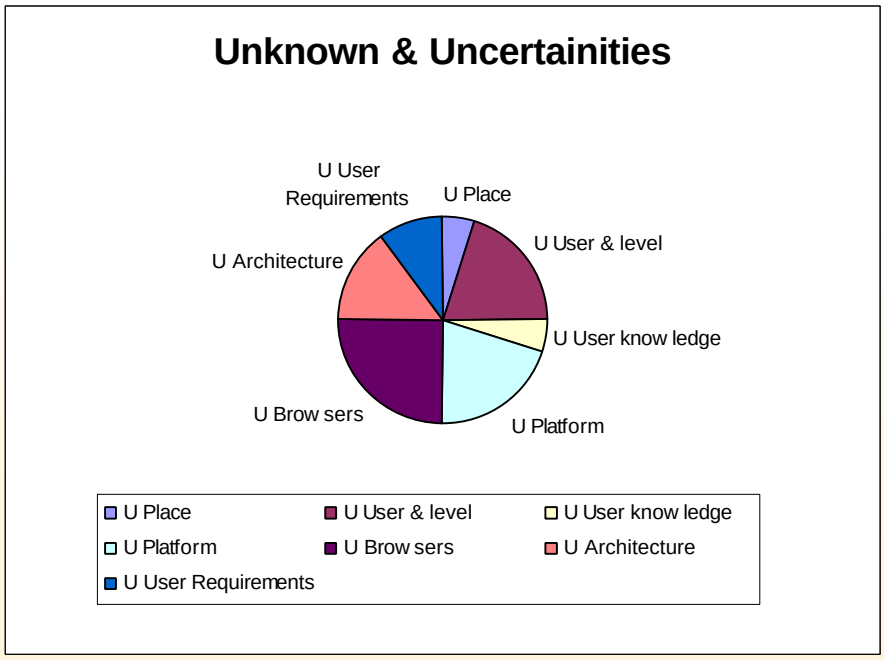
- Introduction to Web Testing
- Web Site Quality and Reliability
- Performance Model & Parameters
- Approaches for Performance Testing
- Performance Indicators & Analysis
- Conclusion & Future Work

WEB APPLICATION TESTING

- It is relatively a new area
- There is a difference in testing Client/ Server and Web Applications
- Automated testing of WebSites is an opportunity and a Challenge

UNKNOWN AND UNCERTAIN

- Unknown place
- Unknown platform
- Unknown Browsers
- Unknown Architecture
- Unknown User and level
- Unknown User knowledge
- Unknown User Requirements



WHAT TO TEST

- All components on client side
- All components on server side

WEB TEST REQUIREMENTS

- More graphics oriented compare to standard applications
- More multimedia functionality to be considered
- Performance requirements for database driven applications

WEBSITE QUALITY AND RELIABILITY

- Dimensions of Quality
- Impact of Quality

DIMENSIONS OF QUALITY

- Performance
- Time
- Structural
- Content
- Accuracy and Consistency
- Response Time and Latency

IMPACT OF QUALITY

- Poor Quality Web Site

  - broken pages and faulty images, CGI-Bin error messages, etc

- Very Complex WebSites

  -Complex colour combinations, no back link, multiple and frequent links

HOW to ENSURE QUALITY AUTOMATICALLY?

- Conduct set of tests automatically and repeatably
- Create Test Sessions
- Plan Test Context

USABILITY

TYPES OF TESTING

- Easy to navigate
- Consistency in look and feel of the site
- Provide Table of Content
- Home button or link on every page
- Avoid too many pop up windows
- Download file and its size must be displayed
- Requirement of plug-in components
- Contact information(email etc)

PERFORMANCE TESTING

TYPES OF TESTING

- Similar to Black box testing
- Looks like an end user to the system under test
- Performed by recording or scripting the actions

PERFORMANCE TEST OBJECTIVES

- Support of maximum number of concurrent users
- Identify location of bottlenecks within the application architecture
- Impact of a software or hardware change
- Scalability issues

Perception, Model, Measures & Parameters

- Perception of performance
  - Web Users
  - Web Masters

- Performance depends on three models
  - Workload Model
  - Performance Model
  - Cost Model

Perception, Model, Measures & Parameters(Contd)

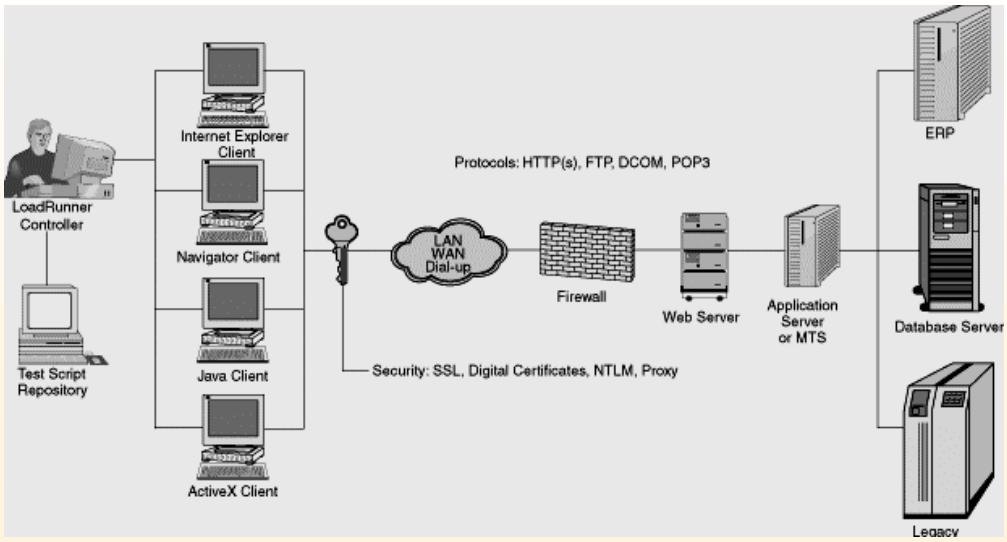
- Performance measures
  - Response Time
  - Throughput
  - Resource utilization
  - Resource queue length

- Performance Parameters
  - System
  - Resource
  - Workload

Performance Indicators

- Form of performance measures and provides basic information about the performance
- Category of Performance Parameters
  - Server side
  - Client side
  - Network side
- Important Performance Indicators are:
  - Processor
  - Physical Disk
  - Memory

TYPICAL TESTING ENVIRONMENT



VARIATIONS OF PERFORMANCE TEST

- Load Testing
- Stress Testing
- Endurance Testing

Approaches for Performance Testing

- Traditional Approach or Conventional Approach
- Transactional Cost Analysis(TOC)

Traditional or Conventional Approach

- **Set the hardware & software environment with required testing tool**
- **Set the performance parameters**
  - **Maximum number of users**
  - **Maximum number of requests**
  - **Best response time**
- **Tackle one variable at a time by keeping other variables constant**
- **Incomplete and time consuming**
- **Used for benchmarking or post-mortem analysis**

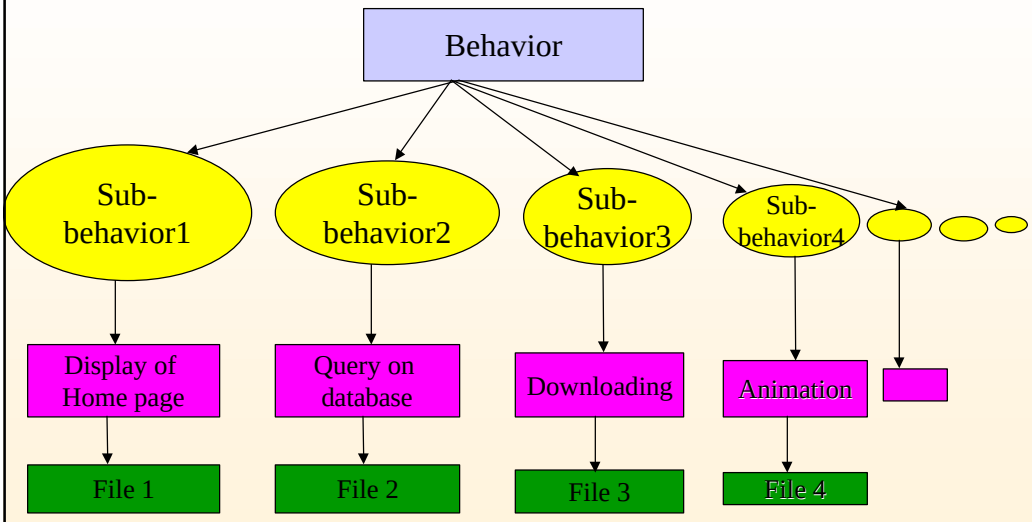
Transaction Cost Analysis(TCA)

- Provides the computing power available
- Framework used to estimate resource cost
- Used for Capacity Planning

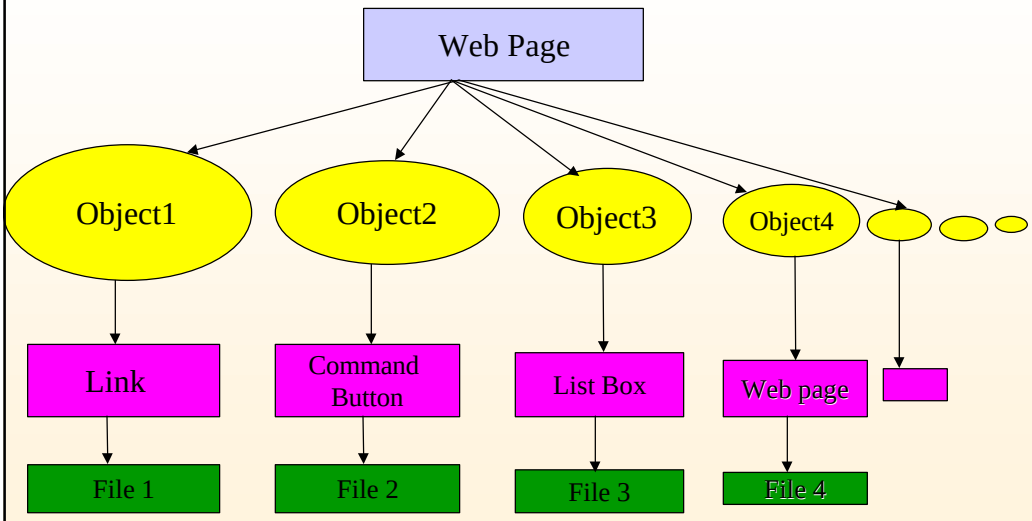
Web Application & Its Structure

- Web site is composed of many web pages
- Web page is
  - linked with many other web pages
  - linked to another web site
- Definition of Behavior of a Web site
- Definition of Web page
  - as an object
  - an object may be a link, a command button, list box, images, audio or another web site itself

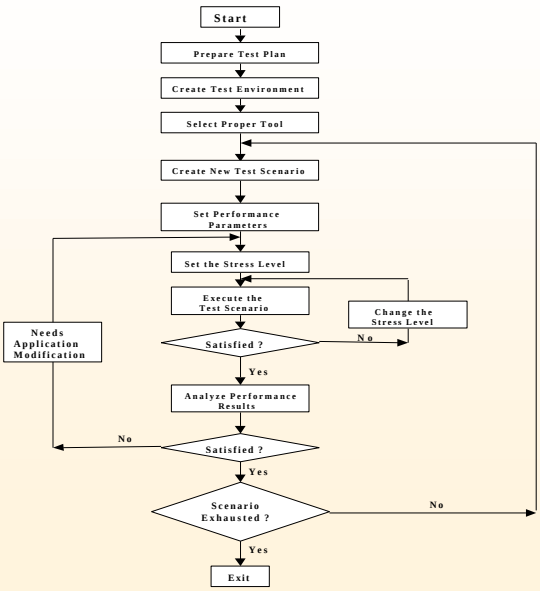
Behavior of a Web site



Definition of a Web Page



Testing Process



TEST PLANNING

- Define test objectives
- Define test tools
- Identify test team members
- Assess risks
- Define critical success factors

Testing Environment : An example

- Name of the Web/Application Server
- Name of the Database Server
- Server machine RAM
- Server machine Processor Speed
- Number of Clients
- Number of User groups
- Number of Users
- Number of Threads
- Number of Sockets/Thread
- Name of the Tool selected for testing
- Test Session Duration

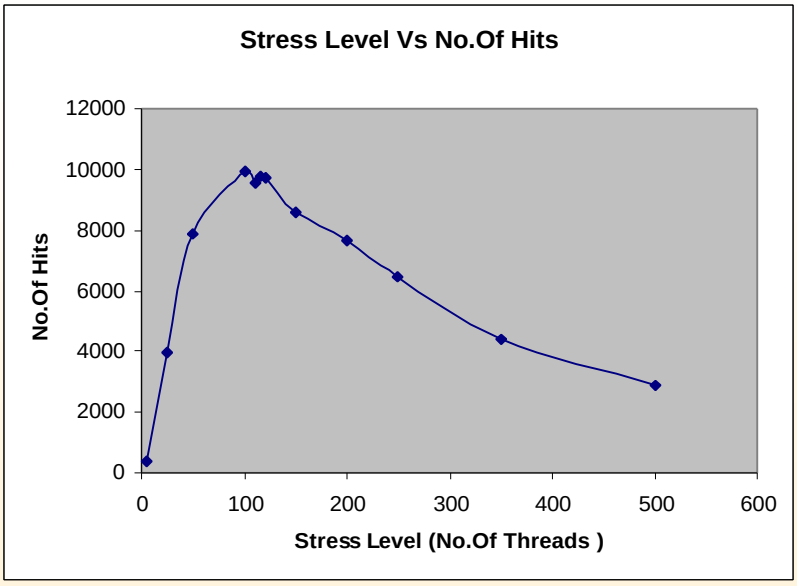
Performance Parameters & Stress Levels

- Parameters varies from object to object
  - Processors
  - Disks
  - Memory etc,
- Stress Levels
  - Number of Users
  - User groups
  - Number of clients
  - Managing threads and sockets

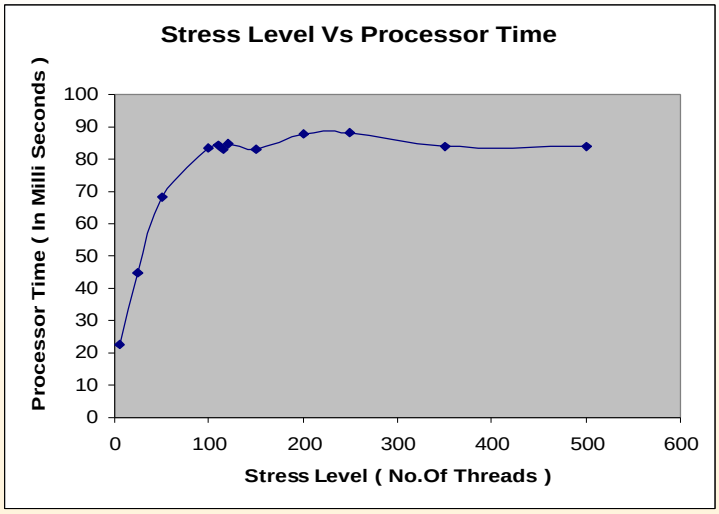
TEST EXECUTION

- Set the duration/ session
- Analyze test cases
- Setup applicable test tools
- Keep track of Configuration mgmnt process
- Identify the test run time(day, week, month)
- Perform repeated test execution

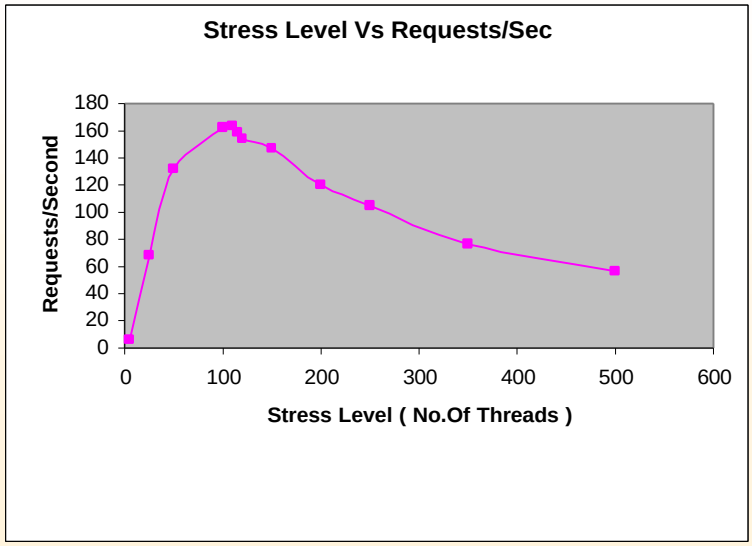
Analysis of Performance Parameters



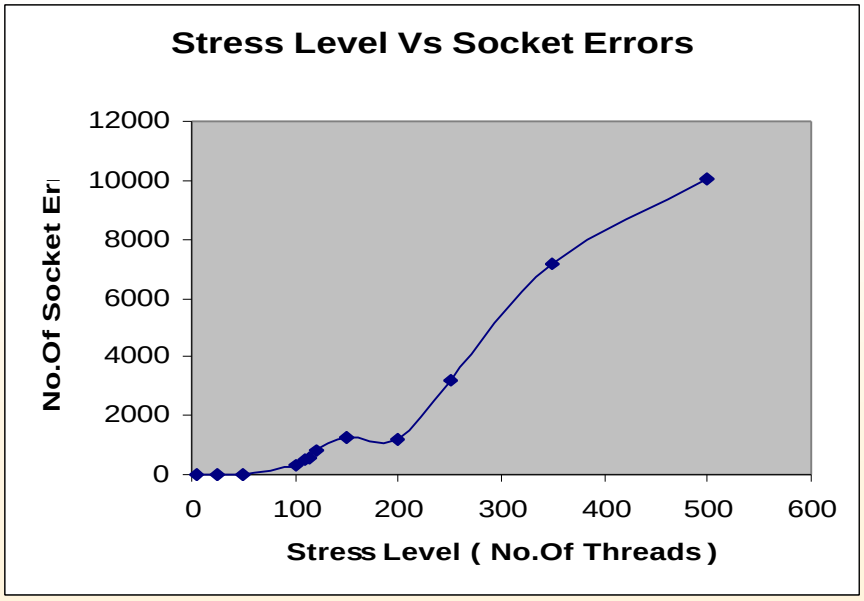
Analysis of Performance Parameters(contd)



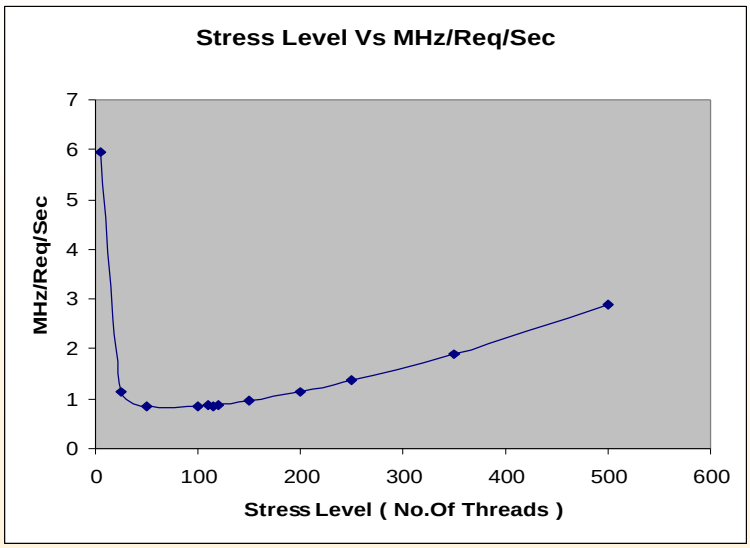
Analysis of Performance Parameters(contd)



Analysis of Performance Parameters(contd)



Analysis of Performance Parameters(contd)



TEST STOP CONDITIONS

- When to stop testing
- Number of test cycles to be carried out

CONCLUSION

- Functional Testing is a must
- Plan for Performance testing from the beginning
- Choose appropriate method for testing
- Select suitable Performance parameters for testing
- Analyse the results for varying stress levels

Future Work

- Modelling business component with the Behavior of the site
- Mapping Behavior with Performance Testing

Thank you

{subraya, subrahmanyasv} @ inf.com

Performance Testing: A Methodical Approach to E-Commerce Applications

by

Dr. B.M. Subraya and Mr. S.V. Subrahmanya

Infosys Technologies Limited
Electronics City, Hosur Road
Bangalore 561 229 India

subraya@inf.com **and** subrahmanya@inf.com

Abstract:

Performance of many Web sites depends on the load on the site at peak time under varying conditions. Traditionally, client-server testing does not ensure the performance of Web sites, as there are unknown users with many uncertainties. Under these circumstances performance testing must be carried out in such an environment which reflects the real users. In order to satisfy the user, it is more appropriate to view the performance testing from the business perspective than the implementation. The business perspective can be captured by the behavior of the Web site as user feels and provide similar environment for testing. Since it is difficult to provide such an environment before the deployment of the application, it is more appropriate to follow a methodical approach that ensures the required performance under varied circumstances. This paper addresses the new testing process, which uses the concept of dividing the behavior of the Web site into testable components with varied performance parameters and stress levels.

1.0 Introduction:

The rapid rise of Web-based E-commerce applications has led to a plethora of design and development tools designed to minimize the time to market for many e-business applications. Designing and developing, however, are only two phases of application development. An e-commerce application demands rapid development and deployment. Testing, a critical third phase occupies the dominant role in the life cycle of the E-commerce application.

Successful e-business applications testing depends on a comprehensive test plan, functionality and performance testing by using incrementally developed scripts, and using these scripts to test under varying assumptions and parameters. This combination provides a thorough and methodical test process.

The first step in testing the e-business architecture is to test the application layer [10]. This type of testing is often performed early in the lifecycle, before the actual GUI for the application is built, to ensure that the essential foundation is present and sound before you add more complex application components. During the deployment of an e-business application, one of the most important issues need to be addressed, is application's performance. Unlike traditional client/server applications [1,2,7], where user loads can be easily predicted, performance problems are inherent to e-business systems, where the volume of users and traffic levels can change in a matter of seconds.

2.0 What is Web Testing?

Conventional application software testing is well defined with a set of methods and established over the years [4,7]. Each method is implemented with a proper strategy. However, one can use some heuristics to test the specific part, which can not be done by the available methods due to many practical aspects of the software. But, the conventional methods may not work with the Web based applications. These applications vary with functionality, presentation, and target users. Thus, Web applications are dynamic in nature and require strong testing methodology. Web testing mainly involves testing functionality of the system like in conventional system [4], presentation layer and performance [5]. During testing of the Web applications, many issues must be addressed as the web site is subjected to many unknown and uncertainties.

3.0 Unknown and Uncertainties(U&U)

Unlike in conventional system, Web based application is targeted for many unknowns and uncertainties. Typically, in a scale of 100, the pie chart shown in fig 3.1 gives approximate distribution U&U, which must be addressed during Web design and testing. Some of the main issues are addressed in the following sections.

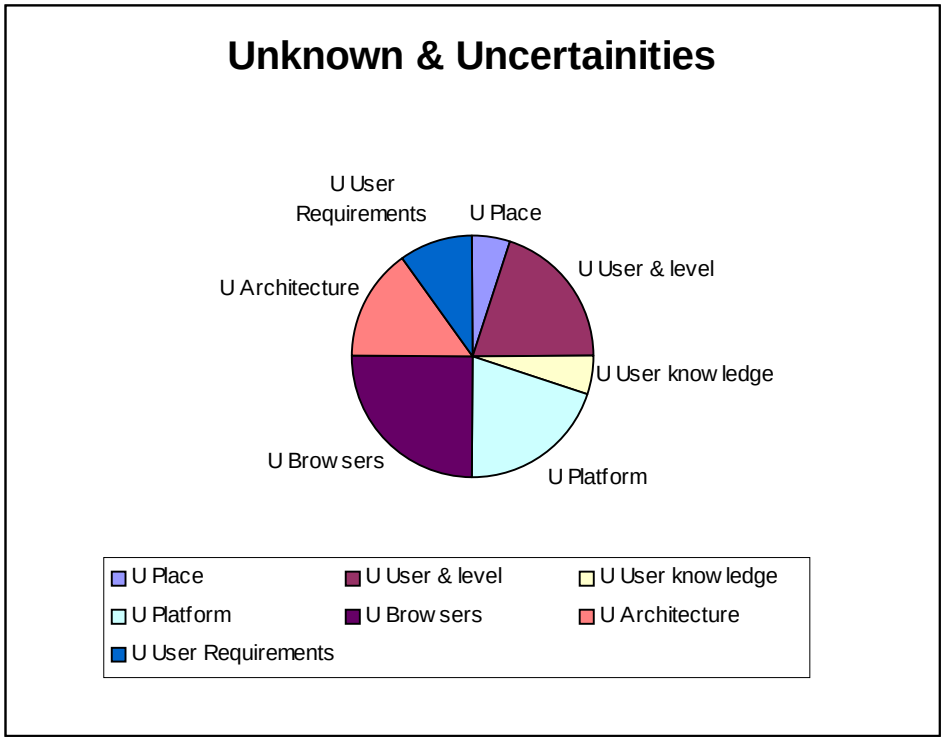


Fig. 3.1 U&U distribution

3.1 Unknown place

Though the Web site is hosted at one place, its users may be spread across the entire globe. Different regions will have different culture, style of functioning and accepting the target application. For example, a particular colour may represent peace in some region whereas it may signify differently in some other region. If the colour used is unpleasant, users may not visit that site for business. Therefore, the place of usage must be taken into consideration while designing the application.

3.2 Unknown User and level

Since the conventional application is targeted towards the known organization with known users. A typical airline reservation system will be used by known users who have sufficient knowledge and training. However, the Web based applications targeted for everybody who may or may not be having sufficient knowledge. The target users may not know the organizational structure of the system. Their level of understanding may be different. A Scientist may be accessing a Web site whereas a teacher may also be accessing the same site and their requirements are different. Their level of thinking may be different. Their level of approach and using the site may be different. How to ensure the unknown users with different levels will not switch to other sites? How to leverage the site among many users that caters to their levels.

3.3 Unknown User knowledge

Many users are visiting the Web site for business but their knowledge and expectations are unknown. A businessman who is proficient in the business may not be having sound knowledge in using the Web site. It may be difficult to learn the technology to use the Web site due to lack of time and interest. During the development of the Web based applications, designers must keep track of the users and their knowledge level.

3.4 Unknown platform

Deciding the specific platform to develop a Web based application is a critical activity in the software development. The application may be developed in some specific Operating System but user may be using a different platform to access the site. Designers must ensure that the site can be accessible from all the standard Operating Systems. It is more common that the data be stored in the legacy system where as the Web server will be in different platform.

3.5 Unknown Browsers

Browser is necessary to access any site and it is integral part of the user requirement. Users must be allowed to use any browsers to visit and doing business at any sites. Web sites should not be browser specific and must be independent of browser dependent platform. Though designers might have tested with a specific browser, it may not work with all browsers. Some of the technologies may not work with all browsers. These browsers will require plug-in components during browsing pages. Many browsers are platform dependent and needs to be configured. Some of the pages may not look clear when viewed through different browsers. Issues like, frames may not be working properly, scroll bars may not be appearing, fonts may not be proper must be addressed.

3.6 Unknown Architecture

The main issues related to the architecture is to find out how the developed application can be accessed by all systems with different types of architecture. For example, it may be Pentium, an old 486 bases system, workstation or risk based system. During the development, it is not guaranteed that all users will have latest technology architecture. There may not be sufficient memory or cache as assumed by the designer during the development. Issues related to scalability must also be addressed.

3.7 Unknown User Requirements

Since it is difficult to satisfy user expectations, efforts must be taken to fulfil most of the requirements. Users always will insist the size of the downloadable file to be displayed. Sufficient width must be provided while accepting the user profile. Table of Contents must be on the left side of a page. Too much pop up windows may not be acceptable. Web sites with more complexity will slowly loose its attractions among visitors. Though the user requirements are unlimited, it is desirable to follow the commonality among many sites.

Inspite of the above U&U, testing of Web applications is a challenge and an opportunity to design a quality Web applications.

4.0 How to ensure the Quality Testing?

As Web sites becoming a fundamental component of businesses, quality of service will be one of the top management concerns. Users, normally, does not care about site failures, traffic jams, network bandwidth[9], or other indicators of system failures. To an online customer, quality of service means fast, predictable response service level of a Web site noted in a real time. User measures the quality by response time, availability, reliability, predictability, and cost.

Since the Web based applications are entirely different from the conventional systems, there is a need to ensure the quality of the final deliverable product. In conventional system, testing is followed as per the guidelines of the test plan. The test plan is mainly concentrating on the functionality of the application and is guided by the requirement specification. However, test plan alone is not sufficient to test the Web based applications. This is due to the dynamic nature of the applications. Therefore, it is necessary to view the quality from different dimensions.

4.1 Dimension of Quality

In the dynamically changing Scenario of the world of Web sites, there are many ways in which one can view the dimensions of the quality. They are:

4.1.1 Time

The Web site always keeps changing dynamically. Some parts of the site may be removed or a new part may be added. There may be a separate link created externally to integrate other sites. Contents of the Web site may be changed or updated periodically. The business may demand the reorganisation of the Web site afresh. The Time parameter mainly deals with the changes in the Web site since the last update.

4.1.2 Structural

A complete Web site is made out of many building blocks. A block may be a frame, a page or multiple pages, an input text box, another Web site itself. These blocks are connected through links. Each page may have many links and links may be repeated in multiple pages also. There are two types of links, viz. Internal and External. Internal links are the ones, which integrate various modules with in the site connected through Intranet. External links are responsible for connecting modules, which are external to the site and connected through the Internet. The structural parameter mainly deals with how well do all of the parts of the Web site hold together. Whether all links are working properly or images will be displayed with out any errors.

4.1.3 Content

The importance of the Web site lies in the relevance and quality of the information it contains. A Web site delivers contents of many forms, such as HTML documents, images, sound, and video clips. Content size, structure, and links affect the performance of the site. Sometimes highly used pages may have to be mirrored either in the same place or at different places.

Some of the Web sites are mainly concentrating on content publishing and give more importance to the contents. Keeping in view that a third party search engine may be used to get information by searching key phrases, the content must be created carefully. Therefore, key phrases must exist continually in highly changeable pages. Maintaining quality content from version to version is really a challenging task.

4.1.4 Accuracy and Consistency

Accuracy decides the correctness of the Web sites where as consistency measures the reliability of information. It is a necessary condition to present accurate data and maintain the consistency of data during multiple access. If a page is downloaded multiple times, the same page must be obtained at all downloads. If the data content is wrong during multiple downloads, the reliability of the site will be lost. The Web site must ensure the same data or information during multiple retrievals of the same page. Links must navigate to the destination page that is intended too.

4.1.5 Response Time and Latency

A customer who visits Web sites always expects responses for a specific query with no or minimum delay (latency) time. The end-to-end response time must be as fast as possible after a query is submitted. This parameter mainly helps to identify parts of the Web site, which are slow.

4.1.6 Performance

Performance is an important dimension, which measures how a Web site can retain the customer permanently. If more than one concurrent user access the site through the browser, how quickly one can satisfy the query of many users. Is the connections between various components are performing as expected? What is the measure of the performance of an E-commerce site? Performance varies by time of day, by load and usage. Taking ten minutes to respond to an E-commerce purchase is not acceptable.

4.2 Impact of Quality

Quality of the Web site, though depends on the above parameters, it has to be maintained periodically to ensure the quality permanently. Poor quality implies that the customer will no longer visit the site and hence the organization may loose business. The issues that affects the quality are, broken pages and faulty images, CGI-bin error messages, complex colour combinations, no back link, multiple and frequent links, etc.

4.3 Ensuring Quality automatically

In a changing Scenario of the Web based applications, it is necessary to maintain the Web sites with out sacrificing the quality. To ensure the quality, one has to conduct set of tests automatically and repeatedly through test sessions. To have proper test sessions, test plan and test context must be well prepared. Apart from keeping the quality conscious, the Web site must address the usability issues. The usability issue

mainly depends on the psychology of the users which varies from one user to another. However, following issues may be addressed during the designing and testing of the Web applications:

- Easy to navigate
- Consistency in look and feel of the site
- Provide Table of Content
- Home button or link on every page to guide to a proper destination
- Avoid too many pop up windows
- Downloadable file and its size must be displayed
- Requirement of plug-in components must be known
- Contact information(email etc)

Quality aspects of any Web site must be addressed carefully and completely. A systematic planning to conduct various tests to satisfy the quality parameters described above is unavoidable. There are many ways in which tests are conducted and organised. However, an important quality parameter, performance, has to be addressed in order to ensure the reliability and speedy accessing of web sites.

5.0 What is Performance Testing?

Modern computer systems are becoming more complex and dependent on many factors such as the network technologies on the Internet. Computing is distributed between various processes such as in the client/ server paradigm or clients and Web servers. Deploying applications that rely on Web servers, Intranets, and client/server technologies is a challenge both in assuring that the functionality will be maintained and in guaranteeing that the functionality will be delivered with an acceptable performance. Performance problems can bring all sorts of undesired consequences, including financial and sales loss, decreased productivity, and a bad reputation for a company. It is important to plan ahead of the system deployment and capacity of networked systems. Capacity planning involves being able to predict when the existing applications will fail to meet the required performance levels. Performance prediction may be accomplished through performance models [5].

Performance testing is directly reflecting the behavior of the complete Web site. Visitors expect the fast response with in a short period of time. Therefore, a rigorous performance testing must be carried out on each site. Performance testing can be viewed as a “Black Box” testing, which focuses on application and system behavior from outside, with no knowledge of the program code that supports the system. The main objectives of the performance testing is:

- Maximum number of concurrent users that can be supported while offering “acceptable performance.”
- Maximum number of concurrent users that can be supported prior to causing a system failure.
- Location of bottlenecks within the application architecture.
- Impact of a software or hardware change on the overall performance of the application.
- Scalability issues

The performance testing is done by recording or scripting the actions that real users perform, and then playing those actions back against the System Under Test (SUT), in an automated and controlled manner. Performance testing also depends on certain issues like, Graphics, Load Time and Connection Methods.

6.0 Performance Modeling

Performance of a system in internet can be analyzed from different viewpoints. For instance, a Web user's perception of performance has to do with fast response time and no connections refused. On the other hand, a Webmaster's perception of performance is oriented towards high connection throughput and high availability. The Web server performance depends on several factors: hardware & software platform, operating system, server software, network bandwidth [9] and workload. There are various well-known methodologies for performance evaluation of computer systems [6].

The overall performance of any system mainly depends on three models: a workload model, a performance model, and cost model [5]. The workload model captures the resource demands and workload intensity characteristics of the load brought to the system by the different types of transactions and requests. The performance model is used to predict response times, utilizations, and throughputs, as a function of the system description and workload parameters. The cost model accounts for software, hardware, network-communications, and support expenditures.

An important aspect of performance modeling involves predicting whether a system will deliver performance metrics like response time and throughput. Performance prediction is the process of estimating performance measures of a computer system for a given set of parameters. Typical performance measures include response time, throughput, resource utilization, and resource queue length.

Parameters are mainly divided into the following categories:

6.1 System Parameters:

Some of the characteristics at the server end normally affect the performance. Some of the parameters include, load-balancing at the mirrored Web-server end, the type of Network protocols, maximum number of connections supported by a Web server, and maximum number of threads supported by the database management system.

6.2 Resource Parameters:

Some of the resources affect the performance by virtue of its intrinsic features. For example, disk seek times, latency and transfer rates, network bandwidth, router latency, and CPU speed ratings.

6.3 Workload Parameters:

Workloads on sever at any instant affects the Performance of the system. Some of the parameters like number of hits/day to the Web proxy server, number of requests/sec submitted to the file server, number of transactions submitted per second to the database sever plays a dominant role in performance testing.

Performance prediction requires the use of models. To predict the performance of a system we need to be able to solve the performance model that represents the system.

Two types of models are used: simulation and analytical model. Analytic models are based on a set of formula or algorithms used to generate performance metrics whereas simulation models are computer programs that mimic the behavior of the system. However, for each model, set of performance indicators must be defined.

7.0 Performance Indicators

To ensure the stability of the Web site and provide a good measure for the performance testing, it is necessary to define a set of indicators. These indicators will provide the performance behavior of the Web site. One can categorize the indicators from the client and server side. Though these indicators provide basic information about the performance, it does not provide true picture of the real environment. Some of the indicators are defined below:

7.1 Processor

Processor is an important resource, which affects the performance of the application. It is necessary to measure the amount of time spent in processing the threads by one or many CPU's. If the number consistently above 90% on one or more processors indicates that the test is too intense for hardware. This indicator is applicable only when the system has multiple processors.

7.2 Physical Disk

Since the disk is a slow device with more capacity to store, the parameters like latency and elapsed time provides more information on improving the performance. Performance of the system depends on disk queue length also. It shows the number of outstanding requests on the disk. Sustained queue length indicates a disk or memory problem.

7.3 Memory

Memory in any computer system is an integral part of the hardware system. More memory will speed up the I/O process during execution but burdens on the cost parameter. All data and instructions to be executed must be available on the memory comprised on pages. If more pages are available, faster will be the execution. The observation on number of page faults will give information related to the performance. It is worth considering how many pages are being moved to and from the disk to satisfy virtual memory requirements.

7.4 Network Traffic

It is not possible to directly analyse the network traffic on the Internet that depends on the bandwidth [9], type of network connection and other overheads. However, it is possible to find out the time taken for number of bytes reaching the client from the server. The bandwidth problem will also affect the performance as the type of connections varies from client to client.

Performance indicators provide the basis for testing and in each category, different types of counters can be set. The values provided by these counters will greatly helps in analysing the performance. To set the performance counters, a proper environment must be established for testing.

8.0 Right Environment for Performance Testing:
Since unknown users and uncertainties are surrounding the usability of the Web site, it is necessary to provide the proper hardware and software environment for the performance testing. The main focus of the performance testing is to generate multiple virtual users and each virtual user performs a set of activities simultaneously. It is not possible to create a real life environment at the production centre but one can create a virtual environment by using testing tools and proper hardware and software. Sometimes it is necessary to create cluster of clients as a representative of different Operating Systems, multiple browsers and architectures. It may be required to create different stress level associated with different clients.

A typical architecture[11] for such simulations may be viewed as shown in fig 8.1.

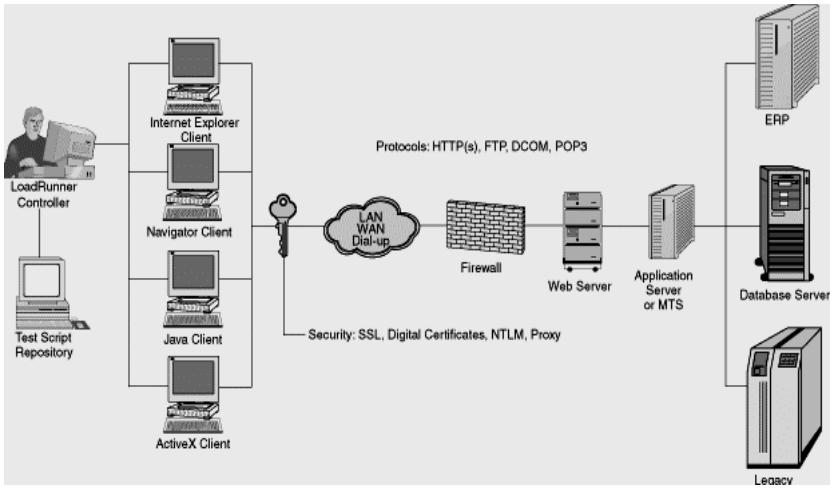


Fig 8.1 Typical Architecture for Performance Testing

A single load generator (fig 18.1) can typically simulate the behavior of tens of hundreds of users through the use of very small Web clients known as virtual users. Virtual users communicate directly with the Web server or through any type of browser available with the client. These virtual users may be grouped and mapped on to a specific client and such many clients may be created. However, all the clients can be controlled from one master client. The delays for the users think time and network latency can also be programmed. The test sessions can be planned and during the session, the type of activities can be programmed by using the record & play or scripting the behavior of the site.

The application servers may be typically either Microsoft Transaction Server (MTS) or any other server[3]. If the applications are available on legacy servers, it is necessary to build a proper mapping. Thus migration of data from such legacy system to the Web server must be carried out automatically. In some cases, the environment may demand cluster of application servers spread across in different locations of the Intranet. To test the security aspects of the site, it is also necessary to build up the security environment so that proper test case can be generated.

9.0 Variations of Performance Testing:
The performance testing is done by recording or scripting the actions that real users perform, and then playing those actions back against the system under test in an

automated, controlled manner. Such scripts can be edited to suite the specific environment, if required. Though the performance testing depends on the specific objective set by the customer, there are three common variations of performance tests that routinely used[1].

9.1 Load tests

Load testing provides insight into the working of the Web site in the day to day conditions. It models the behavior of users in the real world. Test script mimics the behaviors that the users commonly perform, and include think time delays and arrival rates reflective of those in the real world. When selecting think time delays and arrival rates to be used in the load test, it is best to choose values observed during peak load. The behavior of the applications selected will depend on the type of the system under test. In some of the applications, the test may run for a short period of time depending on the functionality being tested. Load test mainly provides insight in to the application under test.

Most Web actions should complete in seconds, thus a 10-30 minute test duration is usually adequate to bring the load up to a steady state, exercise the desired functionality, and get a large result set.

9.2 Stress tests

Stress testing is load testing with the user think time delays removed. In stress tests, scripted actions are executed as quickly as possible. Stress testing reflects the worst case scenario likely to be experienced by a Web site. It can bring to light performance issues that might not otherwise be seen, such as database record contention or the impact of too few connection service processes. In some of the applications, it is essential to increase the stress and find out whether the application under test breaks or not. Some of the Web site applications may not require stress testing as intensive because of its nature.

9.3 Endurance /Durability tests

Endurance testing can be thought of as a long duration load or stress test. Instead of test execution period lasting for tens of minutes, tests are executed for hours or even days. Endurance tests can reveal obscure system defects such as slow memory leaks, the accrual of uncommitted database transactions in a rollback buffer, queuing of downstream systems, or other gradual impacts upon system resources. This type of tests are required for applications which is of high importance and tenable for a long time. It is difficult to conduct endurance tests at the production centre but can be done as part of the acceptance test.

10.0 Approaches for Performance Testing:

Many approaches and strategies are available for the performance testing. However, the type of approach to be followed depends on the type of application and the criticality of the application. Web based applications are dynamic in nature and requires methodical approach to test its functionality and performance. The approach must be continuous and repeatable as performance testing is a continuous process. Some of the basic approaches may be used to test the performance of the application.

10.1 Traditional or Conventional Approach

In this approach, we first set the hardware and software environment with the required testing tool. Performance testing is then carryout keeping in view of the required performance parameters. The parameters may be either the maximum number of users or requests that can be handled or the best response time that can be achieved. At any time, only one variable is tackled by keeping the other variables constant. For example, set the response time as 1 sec and keep increase the number of users so that the response time will not exceeds its limit. This approach determines the number of users that the site can serve with acceptable response.

This approach may be incomplete and time consuming as it may not be possible to perform the test with all permutations. It is better to work with a representative number of combinations and then attempt to estimate the results by extrapolating these results.

Conventional approach may be best used for benchmarking purposes or a *post-mortem* analysis to obtain statistics on the various performance parameters. It does not help in a practical environment where-n many external factors are involved.

10.2 Transaction Cost Analysis (TCA)

Since traditional conventional method lacks the dynamic nature of the application, TAC provides a framework to estimate the cost based on the workload. The workload is based on many factors which includes the following [5]:

- Number of clients that requests pages,
- Type of page reflecting size & access frequency,
- Number of pages available on the server under test,
- Number of client machines or client processes,
- Number of hardware resources used.

The framework estimates each resource cost as a function of usage profile, service mix, or hardware configuration. Hardware resources include, but are not necessarily limited to, CPU, cache, system bus, RAM, disk subsystem, and network. TCA also helps in reducing the potential for over-provisioning hardware since the entire (linear) workload range is measured. TCA captures all costs present during run-time including operating system overheads.

To understand the costs better let us assume an ASP based application being tested for performance. Suppose the system at optimum performance levels is able to handle n (say) ASP requests per second.
Let us assume the average computing power of the system as y MHz. The cost of one ASP requests per second becomes (y/x) in MHz/ASP requests per second.

If more than one processor is used as resources, the average computing power must be computed accordingly.

i.e. Average processor time spent on specific task =

(Number of Processors) * (Speed of Processors) * (percentage of processor utilization)

For example, consider 70 ASP requests per second on four processors with 200Mhz speed with 70% processor utilization gives the cost as:

The total computing power becomes, 4 processors * 200 MHz = 800 Hz.

If we assume 70% processor utilization, the available computing power is,

(800) * (0.70) => 560 MHz

If 70 ASP requests per second to be satisfied, the cost of one ASP per request becomes,

560/70 => 56/7 => "8 MHz per ASP request per second"

This provides a picture about the cost utilization of an object during multiple requests. Based on the above framework, simulate load over linear operating regime to cost each transaction and interpolate resource costs over operating regime. Cost of each resource as a function of usage profile is computed which reflects the performance of the system.

TCA is a more proactive methodology and can be used for Capacity Planning exercises. Once TCA has been performed, it can be applied to capacity planning using the following procedure:

1. Define usage profile and calculate throughput targets.
2. Calculate total resource costs.
3. Calculate capacity requirements.

The effectiveness and flexibility of the capacity planning model depends critically on a careful assessment of the expected usage profile for each service. This profile consists of both individual and aggregated user behavior, and site profile information.

10.3 Testing Process:

Once the methodology and the requirements are known, it is necessary to prepare the test process. The Test Process depends mainly on the required resources and proper environment associated with the test plan. The testing process can be defined only when the structure of the Web application is properly defined.

10.3.1 Web application and its structure

Web based application is composed of many web pages and users can interact with Web pages through browsers. Web pages may be linked with many other web pages or external sites through links. A web page is defined as composed of many objects. An object may be a link, a command button or a list box in a page, a message, an image, audio or another Web site itself. Therefore, a Web page may be defined recursively in terms of objects. Object is a fundamental entity used to construct the Web site. Once the Web application is defined, the application as a whole is subjected to the testing process. However, the whole application can not be tested. The application must be sliced into testable objects. These objects reflect the behavior of the Web page. In general, the behavior of the Web site is divided into set of sub-behaviors represented by many integrated objects. The division of the behavior depends on the criticality of the complete or part of the Web site. Sub-behaviors are

independent on its own and may be clubbed together to represent the overall behavior of the site. These behaviors may be captured by using play and record concept and collected in the form of scripts known as Scenarios. The Scenarios are then subjected to testing with a set of performance parameters and stress levels. The overall process of testing sub-behaviors is represented in the flow chart shown in fig. 10.1.

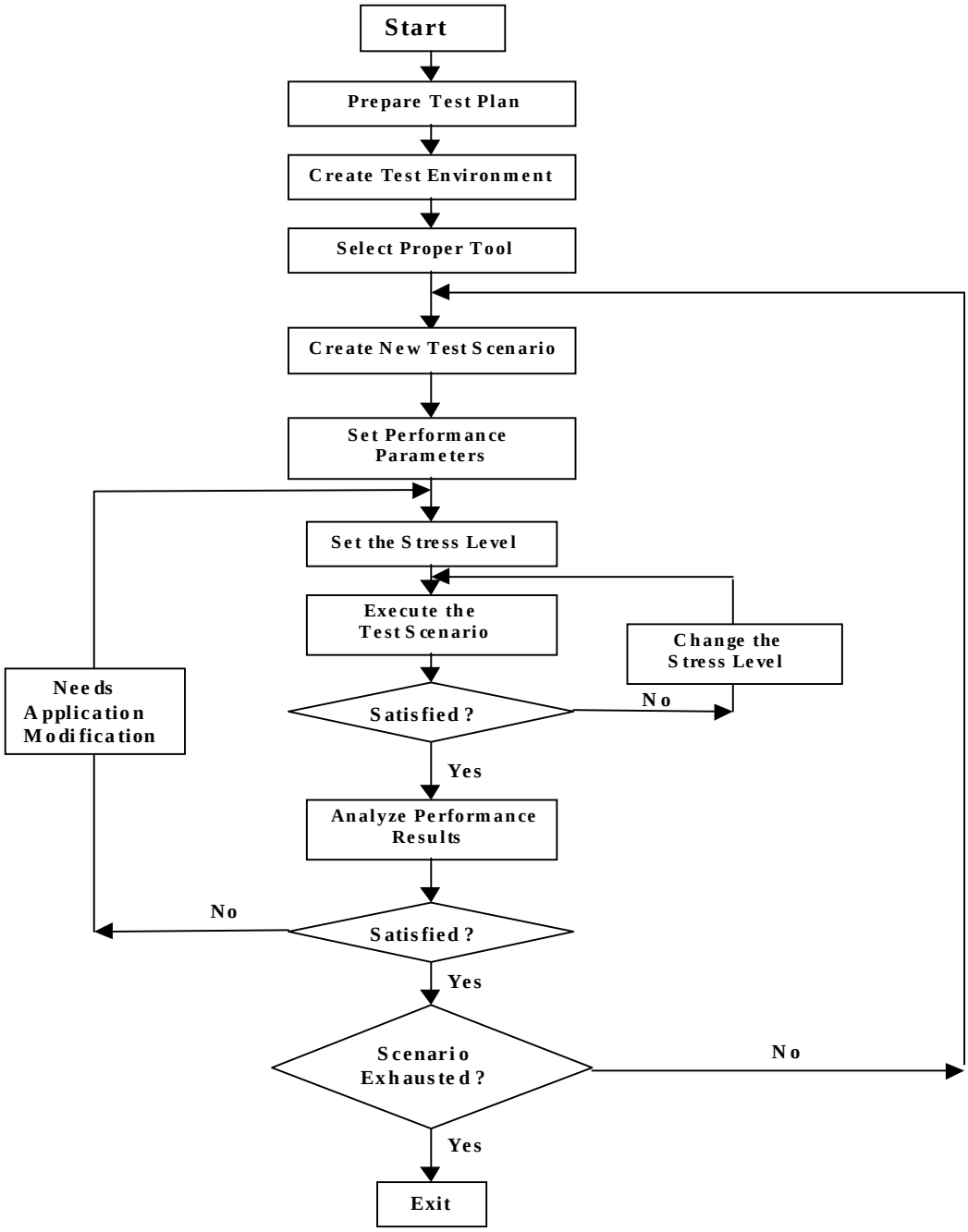


Fig 10.1 Testing Process of a Web application.

The primary activity of the testing process is to create the testing environment as described in section 8.0. Proper testing tool must be procured and installed before starting of the test. There are many tools available in the market, but it is important to find the right tool for the application. One can use some standard parameters to select the proper tool [7]. Majority of the performance testing tools provides the facility to capture the behavior of the application under test. The behavior may be of the whole page or in parts/ slices that is to be decided by the test designer. A set of behaviors may be grouped in the form of Scenarios. Once the behavior is captured in the form of Scenarios, then the performance parameters are set for the selected Scenario. These performance parameters vary from one Scenario to another. The system is then subjected to test under varying stress level for a specific duration against the test plan as described in fig 10.1.

10.3.2 Test Plan

The test plan for performance testing differs from the conventional test plan of the software testing. The conventional test plan tries to bring about the various test cases to test the functionality of the system. In case of performance testing, the functionality of the system is already tested and the correct system is subjected to next level of testing from the user perspective. Thus the test plan for performance testing mainly deals with performance indicators under varying conditions. The test cases includes the stress levels for each performance indicators, number of users, type of grouping the users, architecture which includes clients machine, browsers etc,. Purpose of each test must be specified for each performance indicators and probable performance levels have to be provided.

The output of the test based on the given test plan is analyzed. The results may be represented in the form of graphs for different stress levels that will reflect the true behavior of the system. Based on the analyzed reports, if the user is not satisfied then the application can be modified and subjected to the same cycle of operation. Here the modification of the application may be reconfiguring the servers, fine tuning the database servers, and suitable enhancement to the architecture or software modifications.

Each Scenario can be tested individually or as a test suite for specific interval of time, known as test sessions. The test sessions may be one minute or hours or days that depends on the application and the type of test conducted on the application.

10.3.3 Set the parameters and stress level

Performance parameters are important yardsticks to measure the performance of the system. However, all parameters are not relevant to each and every application. Even for a specific application, the parameters vary with the sub-behaviors of the Web application. It is better to set the relevant performance parameters for a specific behavior that is under test. These parameters vary with the resources and these resources are called as objects. For the specific object, there may be different type of counters to be set. For example, if we consider memory as an object then the counters may be page faults/sec, cache hits/sec and so on. Each counter may not be relevant to all behaviors of the applications.

The stress level depends on the number of users to be simulated for any specific Scenario. The stress level is set to an integer value based on the Scenario that is under

test. The Scenario is captured for similar type of activities. For example, those pages, which are critical and accessed by more visitors, may have to be tested with maximum stress level. These pages may be of query type, content publishing or any other type. Once the stress level is set for a Scenario, it must be executed for a specified period or session. The same Scenario will be repeated for multiple stress level till the required performance level is reached. If the testing process fail to reach the required performance level, generate the results that can be analyzed.

10.3.4 Analyze Performance Parameters

The main issue in performance testing is to set the proper parameters and analyzing the results of the testing. Though setting the parameters depends on the behavior of the specific Scenario, analysis of the result may have to be done by a standard benchmark. However, it is too difficult to take decisions based on the benchmarks available but proper analysis will provide more in-depth knowledge about the performance of the Web site. Graphical representation of performance results may provide global picture for analysis and further improvement. A methodical approach is required to analyse the results. Inorder to help analysing the result, we have used a prototype design and used HOMER performance tool to test the application. Some of the important parameters are analysed by keeping the stress level as base.

Stress Level vs No. of Hits

The performance test is normally conducted for specific period. Keeping this as the base, the performance results are analyzed. During the testing session, the client may generate requests which will be sent to the server in the form of hits. These hits at the server end or requests from the client side depends on the number of socket connections. During the initial stage of the stress level the total number of hits in a session increases with the stress level and reaches a maximum level as shown in the fig 10.2. Then the number of hits decreases with the stress level due to the collisions among many requests. As the number of request increases, the collision among request also increases. Further, number of socket errors also increases which leads to the decrease in the number of hits.

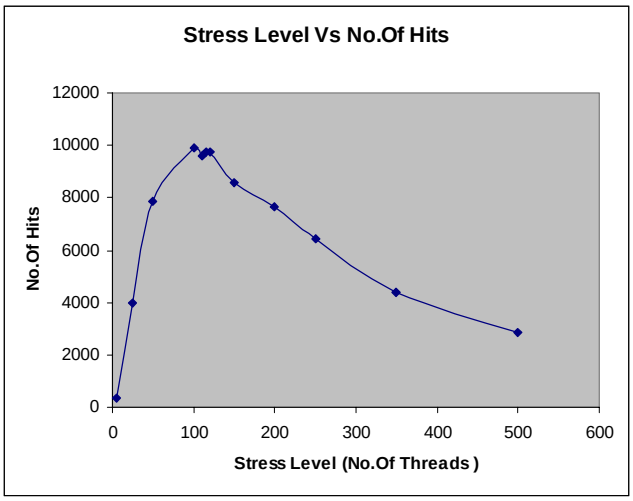


Fig 10.2 Stress level vs No. of Hits.

Stress Level Vs Processor Time

Processor Time is an important parameter in which the stress on the processor can be measured. It is the amount of time spent processing threads by one or many CPU's. To find the sustainability of the processor, it is important to increase the stress on the processor above 80% level as shown in the fig 10.3. It gives how much percentage time spent on executing each thread. As the stress level increases, the process time also increases, reached a maximum and maintains at the same level.

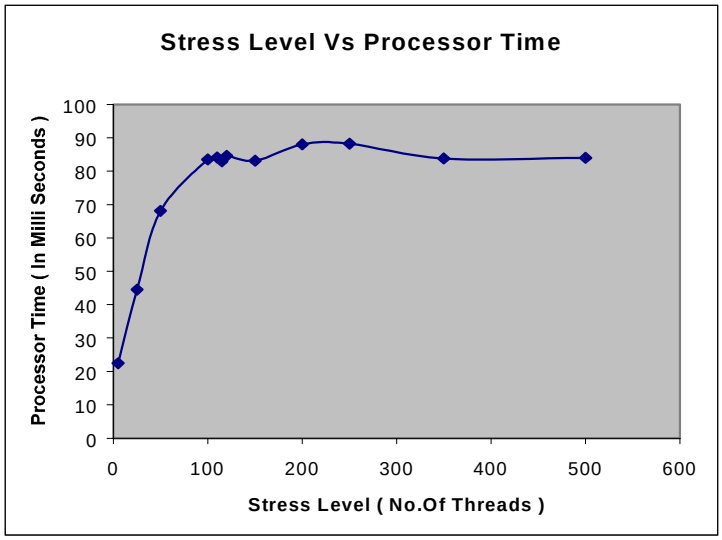
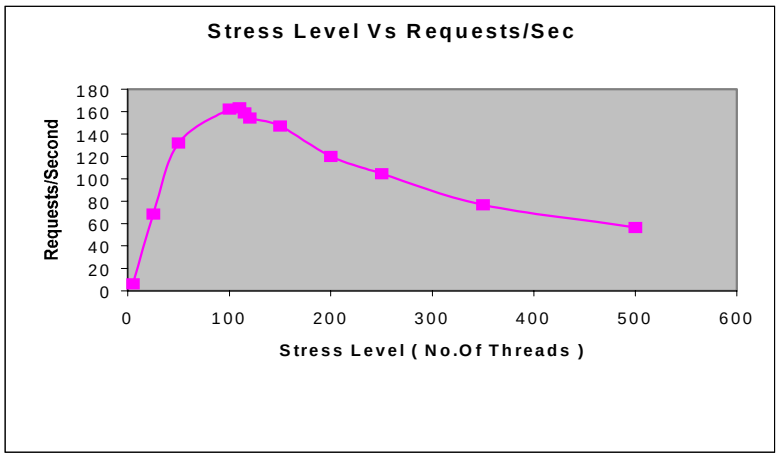


Fig 10.3 Stress level vs Processor Time

Stress Level Vs Requests/Sec

Sometimes the number of requests executed per second from the server gives the in-depth information about the behavior of the sites. The number of requests per second represents the throughput of the server. The number of requests executed per second increases as the stress level increases and reaches a maximum, which reflects the capability of the server. As the stress level increases the execution of request/ sec decreases and reaches a saturation level as shown in the fig 10.4. This is due to the fact that as stress level increases from one client, local processing time is more and at any instant the number of requests received at the server side decreases. After sometime, irrespective of stress level, sever gets the constant flow of requests. The benchmark suggests that the value more than 80% reflect more stress on the server.



Stress Level Vs Socket Errors

Socket is a connection at the server side in which the client can interact through threads. Each thread can create multiple sockets and each socket is a concurrent request. From the web server's perspective, it just sees lots of incoming socket connections. The server doesn't care what the thread architecture is behind the incoming requests. Thus the stress is equal to the multiple of threads and sockets, not simply threads alone. Increasing the sockets per thread allows a single client machine to more efficiently generate concurrent connections. As the number of socket connections increases there is a possibility of generating the socket errors at the server end. These socket error increases as the stress level increases as shown in fig 10.5. Attempt must be made to decrease the number of socket errors with the stress level.

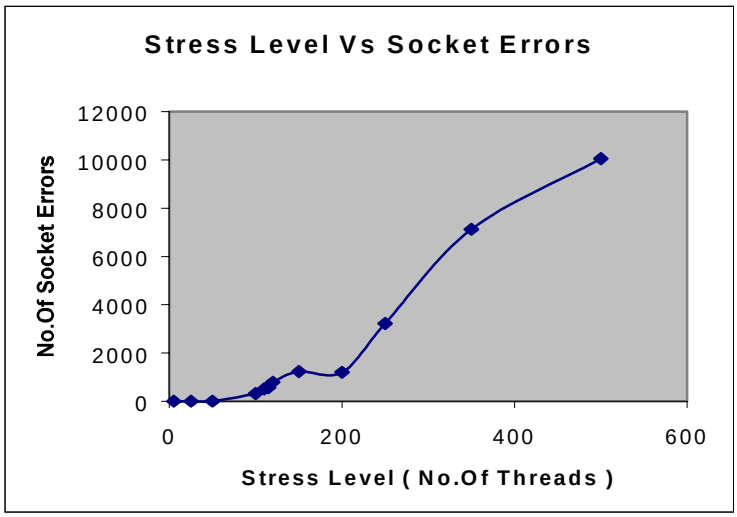


Fig 10.5 Stress level Vs Socket errors

Stress level Vs Computing Power

The computing time required to process a request in a single processor environment depends on the percentage of time allocated to a process as explained in section 10.2. For a fixed percentage of time allocated, the number of requests processed per second decreases as the stress level increases.This means, the computing time allocated to a specific process decreases as the stress level increases or the time taken to process the request with the stress level requires more computing power as shown in fig 10.6. The initial spike in the graph represents the overheads on the processor.

The set of performance parameters considered is to highlight importance of the analysis of the results for various performance objects. This is only an illustration of a sample prototype design that is subjected to a set of performance parameters.

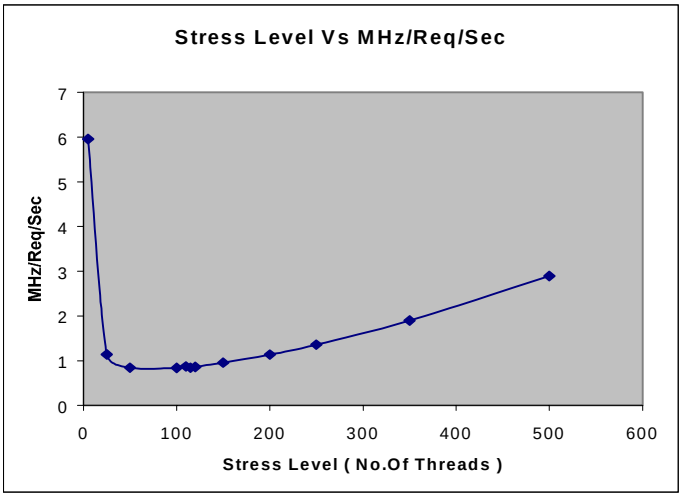


Fig 10.6 Stress level Vs Computing power

Performance testing is not just testing the web site for the required performance but guide the designers for a better optimum performance. This is evident from the above results, which provides in-depth information about the performance of the system. We have injected few errors in the application and tested the application with the same setup. The results were encouraging and supplement the importance of the performance testing.

11.0 Conclusion

Web based applications are more complex compare to the conventional client-server applications due to many unknowns and uncertainties. Among many complexities, performance testing is one of the main activities in E-commerce application development life cycle, which has to be tackled with more vigor and aggression.

Performance testing, which enhances the customer confidence on the web site, is based on many approaches and strategies. Testing the site as a whole is cumbersome and tedious due to many complexities including the behavior. The behavior of the site can be divided into testable sub-behaviors. These sub-behaviors may be part of the web page, a query, a downloadable object or any other meaningful business entity. Each sub-behavior can be captured by using any performance-testing tool as a Scenario. The Scenario is then tested for a specific test session for different test levels. The result is analysed and decision is taken about the performance. This is repeated for all sub-behaviors, which ultimately ensures the testing of the whole site. This methodical approach not only ensures the structured way of testing but also provides step by step enhancement of the quality of the site.

Experimental results shows that the whole Web site need not be subjected to rigorous performance testing with all performance indicators set for each sub-behavior. One can identify critical behavior and test with intensive stress for a long duration to ensure the required performance and stability. However, modeling the behavior of the

Web site before starting of the design in a structured pattern, which helps for the methodical testing, must be addressed.

Acknowledgments:
We sincerely thank Mr. Chandramouli who has helped in conducting the tests and drawing various graphs during the performance analysis.

Reference:

1. Bill Jaeger, *Minimizing Risk with Proactive Performance Testing*, by, E-Business Advisor Article, April 1999.

2. *Microsoft Site Server 3.0 Commerce Server*, Capacity and Performance Analysis.

3. *Optimizing ASP Performance in Site Server 3.0*, Commerce Edition, April 1999

4. Pressman & Roger S, *Software Engineering: a Practionar's Approach*, McGraw Hill International.

5. Daniel A. Menasce and Virgilio A.F. Almeida; *Capacity Planning for Web Performance Metrics, Models, & Methods*; Prentice Hall, PTR, New Jersey 07458, 1998.

6. Menasce D.A. and Dowdy L.W.; *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems*.; Prentice Hall, NJ, 1994.

7. William Perry, *Effective Methods for Software Testing*, John Wiley, New York, 1995.

8. James A. Whittaker, *What is Software Testing? And why is it so hard?*, IEEE Software, Jan/Feb 2000

9. Ralph Barker, *Optimizing Internet Bandwidth*, Performance Computing, January 2000.

10. Dan Sullivan, *e-Business Application Testing in Action*, e-Business Advisor, July 1999.

11. Internal Report, Infosys Technologies Ltd, Bangalore

QW2000 Paper 3M2

Mr. Cem Kaner

Yes, But What Are We Measuring?

BACK TO QW2000 PROGRAM

Presentation Abstract

A theory of measurement must take into account at least 8 factors:

- The attribute to be measured
- The appropriate scale for the attribute
- The variation of the attribute
- The instrument that measures the attribute
- The scale of the instrument
- The variation of measurements made with this instrument
- The relationship between the attribute and the instrument
- The likely side effects of using this instrument to measure this attribute

Software-related measurements often involve psychological or subjective components. They are questionable when they fail to take these factors into account. This paper describes the factors and uses the history of perceptual measurement to illustrate the application of them.

About the Author

Speaker bio to be supplied.

BACK TO QW2000 PROGRAM

Yes, But What Are We Measuring?

Cem Kaner
Quality Week
May 31, 2000

For a related paper, e-mail me at
kaner@kaner.com

Overview

- A theory of measurement involves at least 9 factors:
  - **Attribute to be measured**
    - appropriate scale for the attribute
    - variation of the attribute
  - **Instrument that measures the attribute**
    - scale of the instrument
    - variation of measurements made with this instrument
  - **Relationship between the attribute and the instrument**
  - **Likely side effects of using this instrument to measure this attribute**
  - **Purpose of the measurement**

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Yes, but what ARE we measuring?

- Do bug counts measure testers? Thoroughness of testing? Nearness to ship-ready?
- Does "complexity" measure complexity?
- Does "coverage" measure extent of testing?

How would we know?

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Note what we're measuring

- Note that many of the variables that we're interested in are ***psychological*** variables. (eek!)
  - Competence or effectiveness of testers
  - Complexity of software
  - Even completeness of testing (because completeness is relative to the tester's model).

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

What is measurement?

- Is measurement really "the assignment of numbers to objects or events according to a clear cut rule"?
  - No, it can't be. If it was, then many inappropriate rules would do.
- **Measurement is the assignment of numbers to objects or events according to a rule derived from a model or theory.**

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Definitions from other fields

- "Measurement is the assigning of numbers to individuals in a systematic way as a means of representing properties of individuals."
- "Measurement theory is a branch of applied statistics that attempts to describe, categorize, and evaluate the quality of measurements, improve the usefulness, accuracy and meaningfulness of measurements and propose methods for developing new and better measurement instruments."
  - Allen & Yen, *Introduction to Measurement Theory.*

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Definitions from other fields

- "Measurement is the process of gathering information from the physical world. . . . Measurement is achieved by means of sensors, also called transducers, and involves the gathering of data and its comparison with an agreed standard. . . . A signal is . . . a symbolic representation of some attribute (or combination of attributes) of the system under observation. . . . Data are the numbers associated with a signal. . . . Information is the representation of attributes of the event or object being measured by a known and agreed symbolism."
- "Measurement science is the systematic study and organization of the methods by which information is gathered from the physical world."
    - Sydenham, Hancock & Thorn, *Introduction to Measurement Science and Engineering.*

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Simple measurement

- You have a room full of tables that appear to be the same length. You want to measure their lengths.
- You have a one-foot ruler.
- You use the ruler to measure the lengths of a few tables. You get:
  - 6.01 feet
  - 5.99 feet
  - 6.05 feet
- You conclude that the tables are "6 feet" long.

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Simple measurement (2)

- Note the variation
  - Measurement errors using the ruler
  - Manufacturing variation in the tables
- Note the rule:
  - We are relying on a direct matching operation and on some basic axioms of mathematics
    - The sum of 6 one-foot ruler-lengths is 6.
    - A table that is 6 ruler-lengths long is twice as long as one that is 3 ruler-lengths long.
- These rules don't always apply. What do we do when we have something hard to measure?

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Next measure: A race

- Sandy, Joe and Susan run in a race. Sandy comes in first, Joe second, and Susan third.
  - We assign Sandy the number 1 (first place) and give her $10,000.
  - We assign Joe the number 2 and give him $1000.
  - We assign Susan the number 3 and give her $100.
- Questions:
  - Is Sandy twice as fast as Joe and three times as fast as Susan?
  - Is Sandy 10 times as fast as Joe and 100 times as fast as Susan?
  - We assigned these numbers to Sandy, Joe and Susan according to a rule. Isn't that assignment based on their speed?

Did we measure their speed in this race or not?

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

The race (slide 2)

- The rankings (1st, 2nd, 3rd) could be called measures of speed but they are on an **ordinal scale**. What you can tell from them is that Sandy (1st place) was faster in this race than Joe, who was faster than Susan. Sandy might have been lots faster or a little faster— we can't tell.
- There are five main types of scales (categorical, ordinal, interval, ratio, absolute). A tremendous amount of mathematical arm-waving can be (and has been) done about the measurement problems and properties of scales.
- The scaling problem is moderately interesting, but it is hardly the most difficult problem in a theory of measurement.
- ***The much more challenging problem lies in the relationship between the measure and the underlying attribute.***

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Measurement scales

- ***Absolute scale:*** You have four computers, I have two.
- ***Ratio scale:*** You are twice as wealthy as me— you have $200, I have $100. The relationship is the same if we switch to Canadian dollars, $300CDN vs $150CDN. Multiplying doesn't affect the relationship.
- ***Interval scale:*** (Temperatures of 70, 75 and 80 degrees (Fahrenheit) differ from each other by 5 degrees. The difference (the interval) between 10 and 15, 70 and 75, and 75 and 80 is the same. But a 70 degree day is not 7 times as hot as a 10 degree day.
- ***Ordinal:*** (Position in the race. Bug severity: fatal, serious, moderate, minor)
- ***Categorical:*** (This is a design bug (category 1); that is a control flow bug (category 2).)

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Attributes and Instruments

Length	Ruler
Duration	Stopwatch
Speed	Ruler / Stopwatch
Sound energy	Sound level meter
Loudness	Sound level comparisons
Tester goodness	??? Bug count ???
Code complexity	??? Branches ???
Test completeness	??? Line / branch coverage ???

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Surrogate measures

- "Many of the attributes we wish to study do not have generally agreed methods of measurement. To overcome the lack of a measure for an attribute, some factor which can be measured is used instead. This alternate measure is presumed to be related to the actual attribute with which the study is concerned. These alternate measures are called surrogate measures."
  - Mark Johnson's MA Thesis
- "Surrogates" provide unambiguous assignments of numbers according to rules, but they don't provide an underlying theory or model that relates the measure to the attribute allegedly being measured.

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Relationship between attribute & instrument

- What is the underlying mechanism, or fundamental relationship that justifies the use of a certain observation or statistic as a measure of an attribute?
- Suppose that you increased the underlying attribute by 20%. What would happen to the reading on the instrument?
- In the bug count case, 20% better testing might reflect:
  - More subtle bugs that required more thorough investigation and analysis
  - Bug reports that are more thorough, better researched, and more descriptive of the problem (and more likely to yield fixes)
  - Superb testing but of an area that is relatively stable.
- The bug counts might even go down, even though tester goodness has gone up.

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

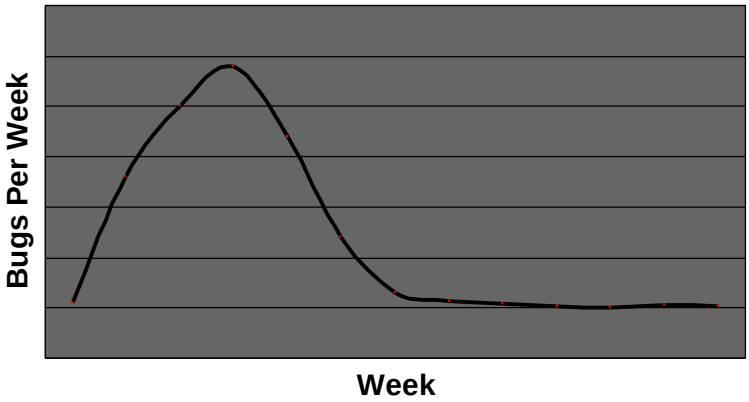
Side effects

- **What if you increase the count of reported bugs by 20%? Does this mean that tester goodness has gone up by 20%**
- If you reward testers for higher bug counts, wont you make changes like these more likely?
  - Testers report easier-to-find, more superficial bugs
  - Testers report multiple instances of the same bug to raise their bug count
  - Programmers dismiss design bugs as non-bugs put in the system to raise the bug count
  - No one will work on the bug tracking system or other group infrastructure.
  - Testers are less willing to spend time coaching other testers.

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

More bug count mischief

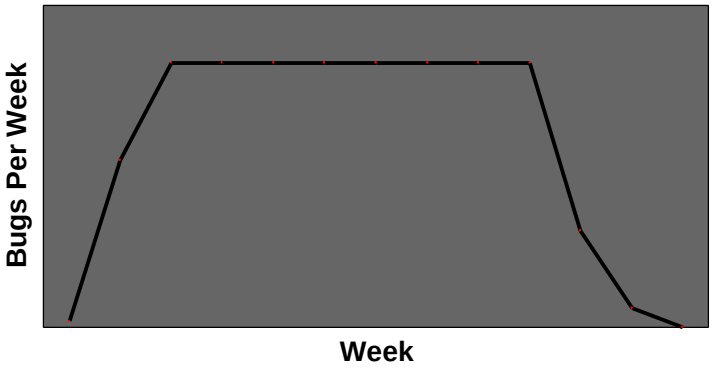
What Is This Curve?



Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

More bug count mischief

Shouldn't We Strive For This ?



Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Code coverage

Coverage measures of the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors:

» ***100% line coverage*** means that you tested for every bug that can be revealed by simple execution of a line of code.

» ***100% branch coverage*** means you will find every error that can be revealed by testing each branch.

» ***100% coverage*** *should* mean that you tested for every possible error. This is obviously impossible.

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

The Problem of Coverage

- Several people seem to believe that complete statement and branch coverage means complete testing. (Or, at least, sufficient testing.)
- Part of the rationale comes from IEEE Std. 982.1-1988, § 4.17, "Minimal Unit Test Case Determination"
- IEEE Unit Testing Standard is
  100% Statement Coverage
  and 100% Branch Execution
- Most companies don't achieve this (though they might achieve 100% of the code they actually write.)

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

But line / branch coverage just test the flowchart

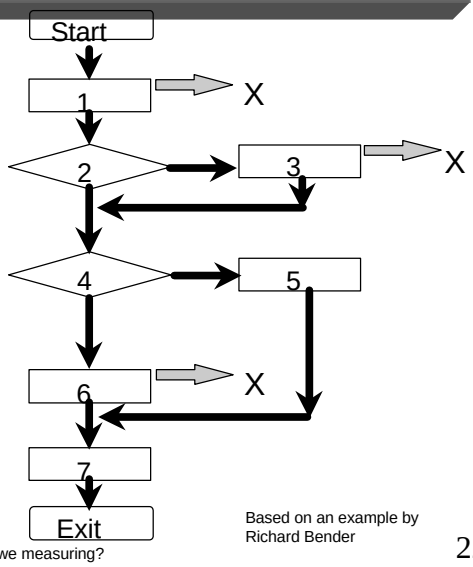
You're not testing:

- » data flow
- » tables that determine control flow in table-driven code
- » side effects of interrupts, or interaction with background tasks
- » special values, such as boundary cases. These might or might not be tested.
- » unexpected values (e.g. divide by zero)
- » user interface errors
- » timing-related bugs
- » compliance with contracts, regulations, or other requirements
- » configuration/compatibility failures
- » volume, load, hardware faults

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Line coverage and data flows

X means this routine changes variable X




```
1(x) 2 3(x) 4 5 7
1(x) 2 4 6(x) 7
```

Now we have 100% branch coverage, but where is 1(x) 7?

```
1(x) 2 4 5 7
```

Based on an example by Richard Bender

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

If we use "coverage"?

- If we improve testing by 20%, does this result in a 20% increase in "coverage"? Does it necessarily result in ANY increase in "coverage"?
- If we increase "coverage" by 20%, does this mean that there was a 20% improvement in the testing?
- If not, then we have a "measure" that is ripe for abuse.
- People will optimize what is tracked. If you track "coverage", the coverage number will go up, but (as Marick has often pointed out) the quality of testing might go down.

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Lacking a theory, consider a balanced scorecard

"Coverage" measures are popular because they provide management with essential (if incorrect) feedback about the progress of testing.

Rather than reporting a single not-very-representative measure of testing progress, consider adopting a "balanced scorecard" approach. Report:

- a small number (maybe 10) of different measures,
- none of them perfect,
- all of them different from each other, and
- all of them reporting progress that is meaningful to you.

Together, these show a pattern that might more accurately reflect progress.

For 101 examples of possible coverage measures, see "Software Negligence and Testing Coverage" at www.kaner.com

Robert Austin criticizes even this approach. It will still lead to abuse, if the measures dont balance each other out.

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Postscript: Measurement and Science

- Hetzel, "The sorry state of software practice measurement and evaluation" in Fenton, Whitty & Iizuka's *Software Quality Assurance & Measurement*
  - Unable to obtain detailed descriptions of experiments underlying published conclusions
  - Unable to obtain reference data
  - Unable to get answers to questions like, "How was testing effort calculated and measured?" In response to claims like, "Inspections found 1 defect per hour of effort compared to testing that typically was .2 or .3 per hour."
- Much of the data is either proprietary or from toy programs

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

Postscript: Measurement and Science

- In other fields, we obtain original data from other researchers in order to:
  - Reanalyze (possibly to reinterpret, possibly to check compatibility with an entirely different theory)
  - Replicate (run the same experiment yourself).
- With most data presented in CS, we cannot obtain original data or thorough description of the original details.
- **Such data are called case studies, not scientifically reported experiments.**
- **If engineering involves the application of science, and our "science" lacks a broad, shared data base, then how can we have genuine, licensable, Software Engineering?**

Copyright © Cem Kaner, 2000. Yes, but what are we measuring?

CEM KANER, J.D., Ph.D.

www.kaner.com **kaner@kaner.com**
P.O. Box 1200 **408-244-7000 (Voice)**
Santa Clara, CA 95052 **408-244-2181 (Fax)**

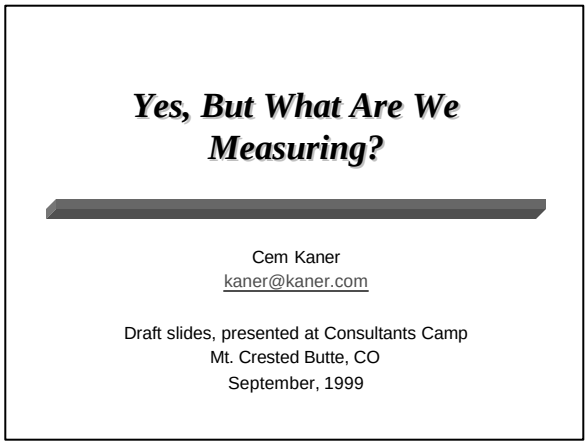
YES, BUT WHAT ARE WE MEASURING?

Notes to Accompany a Talk at PNSQC 1999

Copyright © Cem Kaner, 1999. All rights reserved.

Note: This is a preliminary draft[1] of a chapter on testing and software measurement that will appear in the third edition of Testing Computer Software (by Cem Kaner, Bob Johnson, Brian Lawrence, Jack Falk and Hung Quoc Nguyen). Reference lists, etc. will accompany the next draft. We expect this chapter to be controversial, but we very much want it to be fair and accurate. Please send comments (including flames, if necessary) to Cem Kaner at kaner@kaner.com.

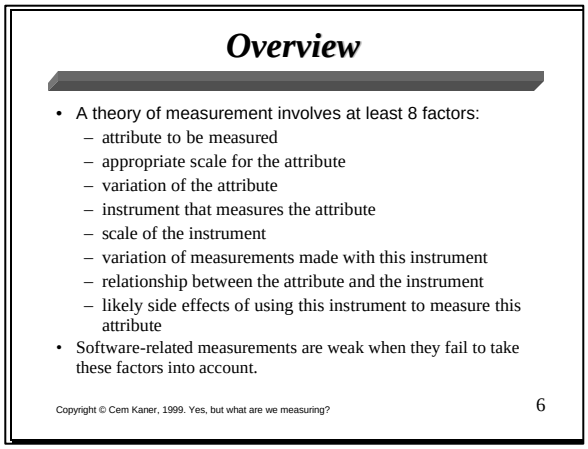
I've put this early version together to supplement the PNSQC slides. Practice runs that the material covers too many issues at once for some attendees. This paper tracks the flow of ideas in the talk. I hope that these notes will improve your experience at PNSQC.



Abstract

A theory of measurement must take into account at least 8 factors:

- **The attribute to be measured**
- **The appropriate scale for the attribute**
- **The variation of the attribute**
- **The instrument that measures the attribute**
- **The scale of the instrument**
- **The variation of measurements made with this instrument**
- **The relationship between the attribute and the instrument**
- **The likely side effects of using this instrument to measure this attribute**



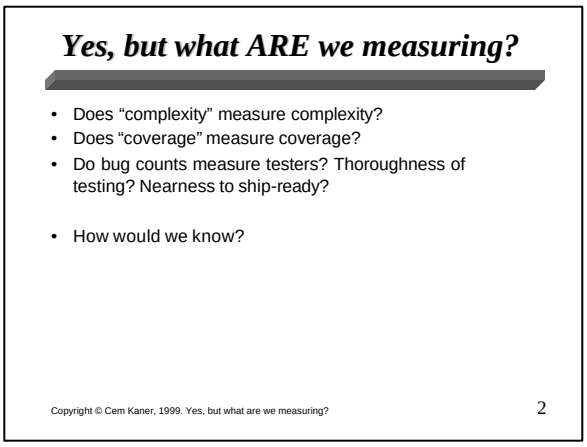
Software-related measurements often involve psychological or subjective components. They are questionable when they fail to take these factors into account. This article describes the factors and uses the history of perceptual measurement to illustrate the application of them.

[1] In early drafts, I use XXX to flag something that must be updated later. I hope that these flags are not too distracting.

Introduction

We would probably like to measure lots of things involved in software testing. Here are three simple examples that carry some publicity (or notoriety) in the field:

- ***Software Complexity:*** How complex (hard to understand, hard to maintain, etc.) is a given piece of software? How hard will it be to test? ***Is McCabe's "metric" a measure of software complexity?***
- ***Extent of Testing:*** After testing a program for a few months, how much testing has been accomplished? ***Is code coverage a measure of the extent of testing?***
- ***Tester Goodness:*** How "good" is a given tester? If "good" is too vague, substitute "productive" or "effective" or "valuable" (or some similar concept of your choice). ***Is bug count a measure of the goodness of testers? Of the overall effectiveness or thoroughness of the testing effort? Of the degree to which the software is ship ready?***



Claims have been made for each of these "measures" but how would we know if they were any good?

One of the core problems of measurement in computing is that many of the things that we would like to measure are subjective—that is, they are complex, qualitative, and involve human judgment or human performance.

- ***Software complexity is a psychological concept.*** If the term means anything, it deals with how complex the software is to a human. Indeed, the complexity metrics are sometimes explicitly referred to as measuring "psychological complexity."
- ***"Extent of testing" metrics are judgment-driven.*** A theory of testing is embedded (often hidden) in such measures. We are not using the population of all possible tests of a product as our baseline when we compute code coverage, and therefore we can have a 100% covered product that still has undiscovered defects. What is the underlying theory and what does it tell us about the relationship between complete coverage and adequate testing?
- ***Goodness (efficiency, productivity) of testers is an issue of human performance.***

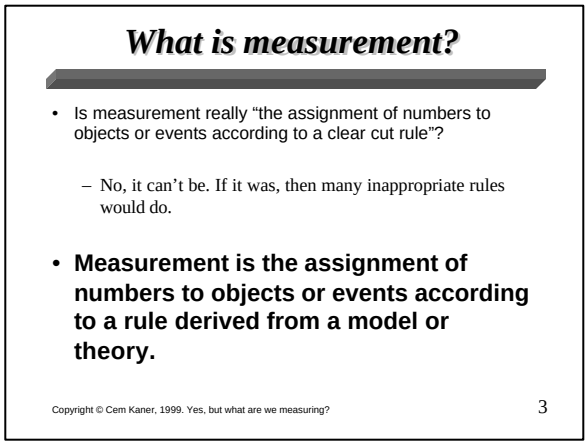
It's not easy to measure subjective attributes. Simplistic notions of measurement simply will not work. In practice, they not only don't work—they can also create harmful side effects.

This paper presents some concepts from measurement theory and illustrates them via an overview of the history of the field of psychophysics (perceptual measurement). That field is far along (after 150 years of work, it should be) despite the fact that the attributes being measured are subjective. I think that we have a lot to learn from that history.

The Definition of Measurement

Within the computing literature, measurement is often defined as the "assignment of numbers to objects or events according to a clear cut rule."

This can't be the right definition. If it was, then we *could* measure "goodness of testers" by counting their bug reports. This (as we'll see below) is ridiculous. The problem is that there are lots of available clear-cut rules. We have to select *the right clear-cut rule*.



Therefore, I prefer the following definition:

> ***Measurement is the assignment of numbers to objects or events according to a rule derived from a model or theory.***[2]

Building a Theory of Measurement

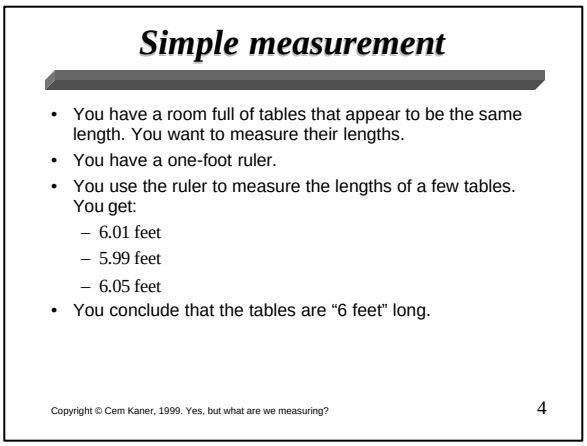
Any theory of measurement has several components. This section starts by laying out key terms, then illustrates them with examples.

Terminology

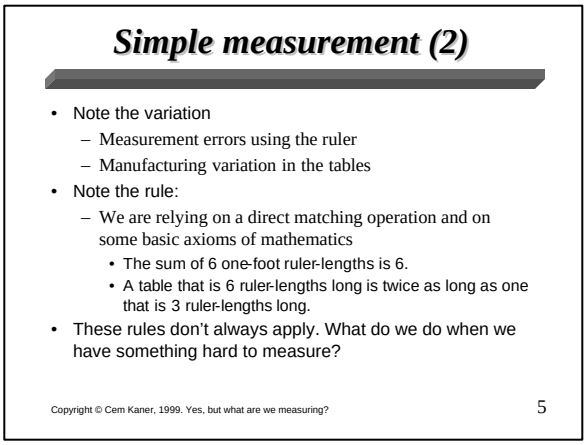
- The goal is to measure some *attribute* (such as length, complexity, or extent of testing).
- To do this, we often use an *instrument* (such as a ruler, when we measure length).

Here are some of the issues:

- The attribute itself is probably subject to random fluctuations. Therefore we need a theory of variation of the attribute.
- The attribute might best be represented on a specific type of scale.
- The act of taking measurements, using the instrument, carries random fluctuations. Thus, we need a theory of measurement error, or of variation associated with using and reading the instrument.



- The measurements are taken along a specific type of scale.
- The instrument will only rarely be a direct measure of the attribute. Normally, we'll need a theory of the relationship between the instrument and the attribute. Ideally, that theory will carry at least two types of information:
  1. What is the underlying cause or mechanism? When there is an increase in the underlying attribute, what causes an increase in the reading on the instrument?
  2. What side effects will result from using this instrument?



A simple example: Using a ruler

Let's start with an example of the simplest case, measuring the length of a table with a 1-foot ruler.

- ***Attribute.*** The attribute of interest is the length of the table.
- ***Attribute' s Scale.*** We'll have more to say about scaling soon. For now, note that the length of the table can be measured on a ratio scale. A table that is 6 feet is twice as long as one that is 3 feet. If we multiplied the lengths (by 12 for example), the (6x12)=72 inch table would still be twice as long

[2] There's no need to restrict measurement to the assignment of *numbers.* As the late S.S. Stevens pointed out, measurement can be looked at as a matching operation on two domains. For example, he would have people indicate the relative loudness of two tones by adjusting the relative brightness of two lights.

as the (3x12)=36 inch table. This preservation of the ratio relationship ("twice as long") when both items are multiplied is at the essence of the ratio scale.

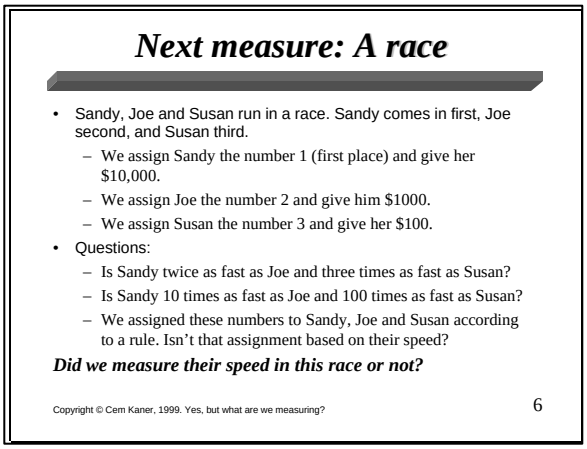
- ***Attribute' s Variation.*** If you measure the lengths of a few tables, you'll get a few slightly different measurements because there is some (not much, but a little) manufacturing variability in the lengths of the tables. Tables that are supposed to be 6 feet long might actually vary between 5.98 and 6.02 feet. The length of individual tables might not vary much, but if you went to an office supply store and asked for a 6-foot table, the table that you would get would only approximate (perhaps very closely approximate) 6 feet.
- ***Instrument.*** The ruler is the measuring instrument. It's 1 foot long, so we'll have to lay it down 6 times to mark off a 6-foot length.
- ***Instrument' s Scale.*** The instrument (ruler) measures length on a ratio scale.
- ***Instrument' s Variation.*** Try to measure a 6-foot table with a 1-foot ruler ten times. Record your result as precisely as you can. You'll probably get ten slightly different measurements, probably none of them exactly 6.000 feet. Even with ruler-based measurement, we have error and variability.
- ***Theory of Relationship.*** The relationship between the attribute and the instrument is *direct.* They're both on the same scale and (except for random error) a change in the attribute results in a directly comparable change in the measured value. (Example: Cut the table down to 4 feet and the next time you measure it, the ruler measurement will be 4 feet.)
- ***Probable Side Effects.*** I don't anticipate any side effects of using a ruler to measure the length of the table. I don't think, for example, that that using rulers will encourage table manufacturers to change how they make tables.

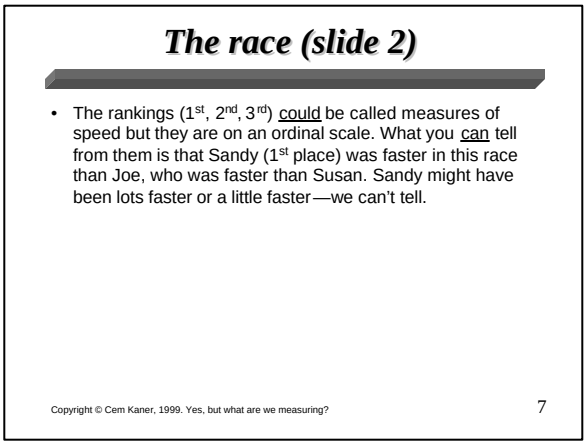
Second example: The scaling problem

Suppose that Sandy, Joe and Susan run in a race. Sandy comes in first, Joe comes in second, and Susan third. The race comes with prize money. Sandy gets $10,000, Joe gets $1000 and Susan gets $100.

We know from the results that Sandy ran a faster race than Joe and that Joe ran a faster race than Susan. Let's review the theory of measurement associated with these results:

- ***Attribute.*** The attribute of interest is the speed of the runners.
- ***Attribute' s Scale.*** The attribute's scale is probably best expressed in terms of miles (or meters) per hour. If so, this is a ratio scale (1 mile per 4 minutes = 15 miles per hour).
- ***Attribute' s Variation.*** The race only gave us one sample of the speed of these runners. If they ran the same track and distance again tomorrow, they'd probably have slightly different times.
- ***Instrument.*** The instrument is the simple observation of the order of the runners as they cross the finish line. There are two different sets of markers on this instrument (like the inches and centimeter markers on rulers). One set of markers says 1st, 2nd, and 3rd. The other markers say $10,000, $1000, and $100. You might prefer to measure speed with some *other* instrument but (oops) you





forgot your stopwatch and this is what you've got. (Please suspend judgment rather than quibbling with this detail of this example. I'm trying to use something that everyone understands in order to make the point easily. We run into plenty of dimensions that we have to measure with crude instruments because we have no idea what a better instrument, like a stopwatch, would be.)

- ***Instrument' s Scale.*** This instrument operates on an ordinal scale, not a ratio scale and not an interval scale. I'm not going to walk through extended definitions of the different scale types in this chapter. For detailed discussions, see Stevens (1976)(xxx his classic book on Psychophysics xxx) and Fenton (xxx) (book on software measurement). The following comparisons should give you a sense of the differences among the scales.
    - If we were operating on a ***ratio scale***, we would be able to say that Susan (measured as 3) was three times slower than Sandy (measured as 1). Or we could say that Susan (measured as $100) was 1/100th as fast as Sandy (measured as $10,000). Probably, neither of these statements is correct and we have no basis for deciding which one is closer to the truth.
    - If we were operating on an ***interval scale***, we could say that the difference between Susan and Joe (3-2=1) was the same as the difference between Joe and Sandy (2-1=1) and that the difference between Susan and Sandy (3-1=2) was twice the difference between Susan and Joe.
    - On an ***ordinal (or positional) scale***, all we know is that Sandy crossed the line first, Joe crossed it second, and Susan crossed it third. Suppose that Bill crossed the finish line too, but was disqualified because he took steroids. The ordinal scale tells us that if Bill crossed the line after Susan, he must also have been beaten by Sandy and Joe. But knowing that Bill crossed after Sandy and Joe doesn't tell us whether he crossed before Susan or not. Even if we knew that Bill crossed the finish line 14 minutes after Joe, we could not tell from this data (from the 14 minute fact and the fact that Susan was awarded 3rd place) whether Bill crossed the finish line before or after Susan.
    - The ordinal scale doesn't tell us much about speed but it tells us more than a ***nominal (or categorical) scale***. It would stretch the example too far to try to talk about a nominal scale for speed but since we've been comparing the different scale types here, this is the best place in this paper to describe a nominal scale. Suppose that you had hundreds of different pieces

Measurement scales

- ***Absolute scale:*** You have four computers, I have two.
- ***Ratio scale:*** You are twice as wealthy as me—you have $200, I have $100. The relationship is the same if we switch to Canadian dollars, $300CDN vs $150CDN. Multiplying doesn't affect the relationship.
- ***Interval scale:*** (Temperatures of 70, 75 and 80 degrees (Fahrenheit) differ from each other by 5 degrees. The difference (the interval) between 10 and 15, 70 and 75, and 75 and 80 is the same. But a 70 degree day is not 7 times as hot as a 10 degree day.
- ***Ordinal:*** (Position in the race. Bug severity: fatal, serious, moderate, minor)
- ***Categorical:*** (This is a design bug (category 1); that is a control flow bug (category 2).)

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 8

Measurement scales (2)

- It rarely makes sense to add or subtract numbers that are not at least interval-scaled
  - (Ordinal example) Is the difference between a Level 1 ("fatal") bug and a Level 2 ("serious") the same as the difference between a Level 2 and a Level 3 (moderate)?
  - (Categorical example) Is the sum of a user interface design error (category 1) and a looping error (category 2) the same as the difference between an initialization error (category 3) and a failure to block interrupts (category 6)? That is, in this case is 1 + 2 = 6 - 3? No.

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 9

Measurement scales (3)

- A tremendous amount of mathematical arm-waving can be (and has been) done about the scaling problem.
- The scaling problem is moderately interesting, but it is hardly the most difficult problem in a theory of measurement.
- ***The much more challenging problem lies in the relationship between the measure and the underlying attribute.***

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 10

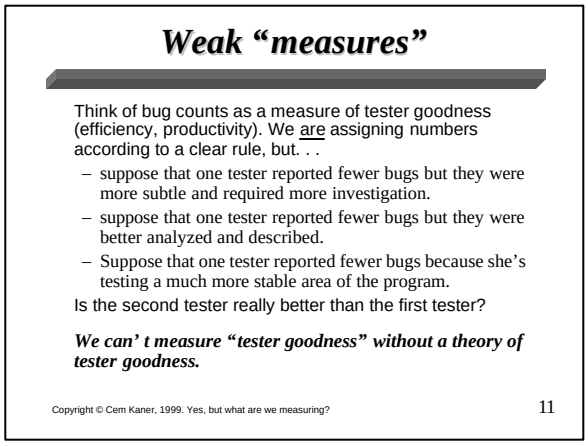
of cloth and you sorted them by color. Here is the brown pile. There is the pink pile. This is the green pile. By sorting, you have nominally (or categorically) scaled the cloths. For those who insist that measurement is the assignment of number to an attribute according to a clear cut rule, make brown color 1, pink color 2 and green color 3. Mix the three piles together and randomly pick one of the cloths. You'll be able to assign a number to it (1, 2, or 3) according to a clear cut rule (brown, pink or green), so categorical scaling satisfies that definition of measurement.

- The final scale to mention is the ***absolute scale.*** If you have 1 child, you have 1 (one) child. This doesn't scale. One is 1 is one. Unlike inches and centimeters for length, the units of measurement (for number of children) are fixed. Two children are twice as many as one, and four are twice as many as two, but these ratios reflect more children, not a change in scale. You don't have two half-children, you have one child. If you cut the child in half, you have a mess, not 2 half-children and not (any longer) 1 child. (Too gruesome an example? Substitute "laptop computer" for child.)

- ***Instrument' s Variation.*** There is probably not much variation in this example, unless the race was very close. We would see variation if different judges, sitting in different places, disagreed as to whether Joe or Sandy came in first.
- ***Theory of Relationship.*** The mechanism underlying the relationship between speed and position in the race is straightforward. Faster speed results in a better position (first, second, third).
- ***Probable Side Effects.*** I don't see obvious side-effects (changing how people run races) in this particular case.

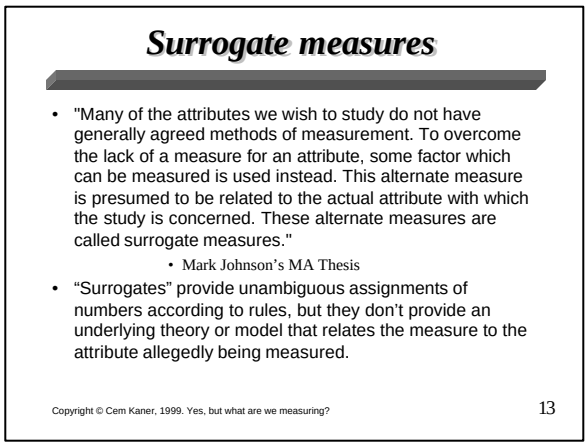
Example 3: Bug counts and the theory of relationship

Should we measure the quality (productivity, efficiency, skill) of testers by counting how many bugs they find? Leading books on software measurement suggest that we compute "average reported defects / working day" (Grady & Caswell, 1987, p. 227) and "tester efficiency" as "number of faults found per KLOC" (Fenton & Pfleeger, 1997, p. 36) or "defects found per hour" (Fenton, Pfleeger, & Glass, 1994). These authors are referring to averages, not measures of individual performance, and they sometimes warn against individual results (because they might be unfair.) However, I repeatedly run into pointy-haired bosses (or, at least, the test managers who work for them) who compute these numbers and take them into account for decisions about raises, promotions, and layoffs.



Let's do the analysis and see the problems with this measure:

- ***Attribute.*** The attribute of interest is the goodness (skill, quality, effectiveness, efficiency, productivity) of the tester.
- ***Attribute' s Scale*** I don't know. Neither do you.
- ***Attribute' s Variation.*** I don't know. But there is variation. Joe probably does better work on some days than others.
- ***Instrument.*** We don't have an obvious, easy to use, unambiguous direct measure of tester effectiveness / skill / etc. So instead, we use a



surrogate measure, something that is easy to count and that seems self-evidently related to the attribute of interest. In this case, the instrument is a counter of bug reports.

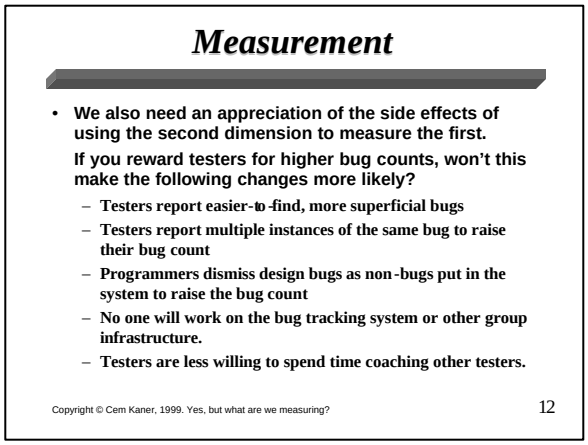
- ***Instrument' s Scale.*** Bug counts are an absolute scale (two half-reports do not equal one whole-report).
- ***Instrument' s Variation.*** There's not much random variation in counting of bug reports, though there is some, such as bugs classified as duplicates, changing the count. There is systematic measurement-related variation, as we'll see in the discussion of side effects.
- ***Theory of Relationship.*** There is a rough intuition that great testers report more bugs than crummy testers. But there is no theory of mechanism that says, an increase of so much in tester effectiveness will result in an increase of so many in the bug count. Here are two examples of disconnection between bug counts and value obtained from a tester. These are both from real situations.

  In the first case, corporate management was overeager to ship a product by a publicized release date. I assigned one tester to use his considerable exploratory testing skills to hunt for showstoppers. He reported as few as four bugs per month, compared to rates of perhaps 25-150 from the other eleven testers. We held that product for six months past the scheduled ship date by finding at least one showstopper per week, every week. Most weeks, one of those showstoppers (often the only one) came from our hunter. Was he the most productive member of our group or the least productive?

  In the second case, a project team identified one area of the product (graphic filters) as very high risk. This was a new version of a hot selling mass market product and there were important business reasons for shipping the product on time. However, if we shipped with the serious problems in this area that we expected, we would be drowned in phone calls. The two testers assigned to this area found surprisingly few bugs and the project team (including me) worried that they might not have gotten a complete handle on the risks. I pulled in a very senior third tester who knew a lot about graphic filters to assess the testing and the area. Over about five weeks, he found about five bugs. He also reported that the test plans and strategies of the first two testers were pretty good. Partially based on this reassurance, we shipped the product a day ahead of schedule, sold a huge number of copies on the first day, and got few complaints.

  In both of these cases, testers with very low bug counts did fine work and played a more significant role in the project than some other testers who found many more bugs in other areas. Bug counts would have seriously mis-measured these testers.

- ***Probable Side Effects.*** Here are just a few of the things that can and probably will go wrong if we measure individuals by counting their bug reports.
  - People are good at tailoring their behavior to things that they are measured against. If you ask a tester for more bugs, you'll probably get more bugs. Probably you'll get more bugs that are minor, or similar to already reported bugs, or design quibbles -- more chaff. But the bug count will go up. (Weinberg & Schulman, 1974)
  - People know that other people tailor their behavior. Put a tester under incentive to report more bugs and every programmer will become more skeptical of the value of the bug reports they receive. Does this tester believe in this bug, they ask, or is she just inflating her bug count? Bug counting creates political problems (especially if you also count bugs per programmer).
  - You can make a tester look good or bad just by choosing what type of testing she should do (regression testing often yields fewer bugs than exploratory testing of the same area of the



program) or what area of the program she should test (fewer bugs to find in less buggy code). If raises and promotions are influenced by bug counts, project assignments will often be seen as unfair. More political problems.

- The measurement system creates incentives for superficial testing (test cases are quick and easy to create) and against deep tests for serious underlying errors. Bug counts punish testers who take the time to look for the harder-to-find but more important bugs.
- The system also penalizes testers who support other testers. It takes time to coach another tester, to audit his work, or to help him build a tool that will make him more effective. The tester who does this has less time to find bugs.
- Time spent on any process that doesn't lead to more bugs faster is time that counts against the tester. For example, bug counting rewards testers who minimize the time they spend documenting their test cases.

Problems like these have caused several measurement advocates to warn against measurement of attributes of individuals (e.g., Grady & Caswell, 1987) unless the measurement is being done for the benefit of the individual (for genuine coaching or for discovery of trends) and otherwise kept private (e.g. DeMarco, 1995).

With only a weak theory of relationship between bug counts and tester goodness, and serious probable side effects, we should not use this measure (instrument).

Measurement in the Real World

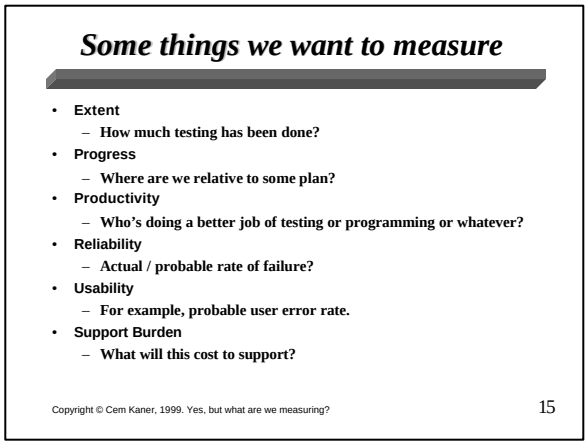
The examples of length and position in the race are toys. They are easy to figure out. The theories of relationship are clear cut and the side effects are minimal.

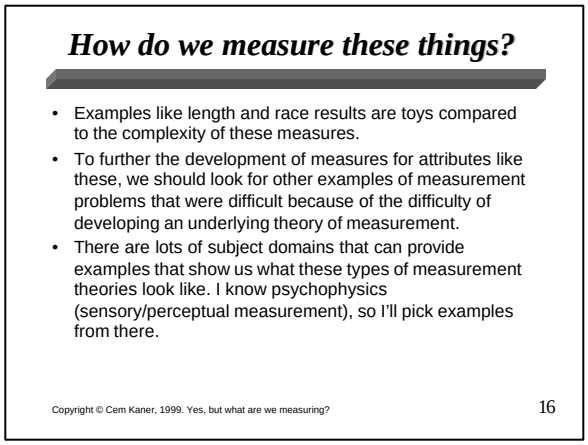
When it comes to things that we would really like to measure, life is more difficult. Examples of the kinds of things that testers are routinely asked about are:

- ***Extent***: How much testing has been done?
- ***Progress***: Where are we relative to some plan?
- ***Productivity***: Who's doing a better job of testing or programming or whatever?
- ***Reliability***: What is the actual and probable future rate of failure?
- ***Usability***: For example, what is the probable user error rate?
- ***Support Burden***: What will this cost to support?

How do we measure these? Each one involves complex issues. Typically they involve a lot of judgment (which is subjective). Additionally, several of the most interesting dimensions involve human behavior. For example, reliability in the field depends on use patterns (use of the product is done by humans, who might not use it the same way as the product developers predict.)

Some of the people who talk about measurement and the development of "science" in our field distinguish between "hard" measurements and "soft" ones, or between "objective" and "subjective." I don't think that these



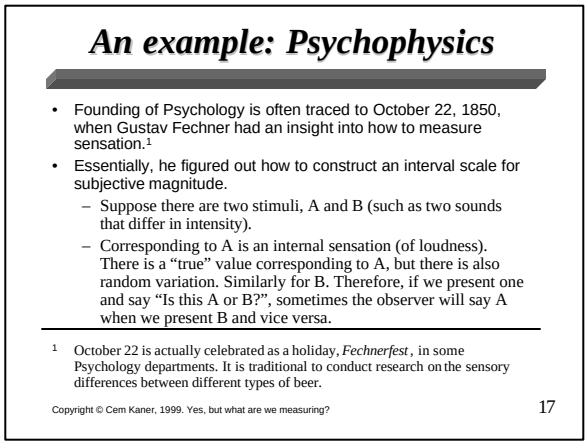


distinctions are useful. They carry too much emotional and judgmental baggage. They carry a notion that subjective issues are immeasurable and unscientific. That's not very helpful if we work in a field of human endeavor, called computing, whose essential work product is the stuff of mental creation.

Rather than shrinking from the subjective issues that dominate our field, I think it would be useful to look at the development of measurement in a field that deals directly with attributes that are fundamentally subjective.

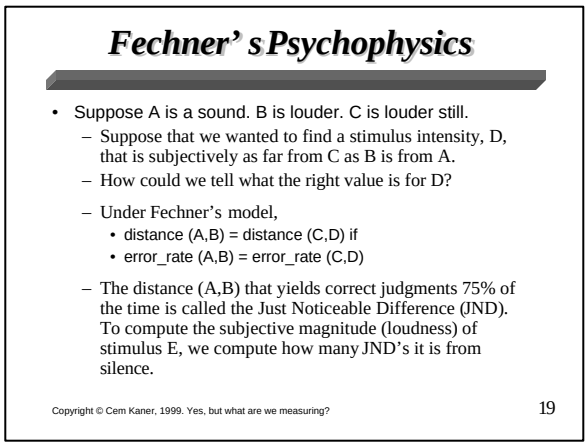
Welcome to Psychophysics

On October 22, 1850, Gustav Fechner had The Great Thought (an insight into how to measure sensation) that initiated the field of Experimental Psychology.[3] In essence, Fechner thought of a way to develop an interval scale for sensations, such as loudness, brightness, or heaviness. Fechner's work was extended by several important researchers and my summary merges their extensions with Fechner's original idea. For important examples of the evolution, see Boring (1917), Urban (1930), and the collection of Thurstone's papers (1959). For thoughtful histories, see Boring (1943) and Link (1992).



Sensations are subjective experiences. When a car honks its horn, I might hear that sound and you might hear that sound, but does the horn sound to me the same way that it sounds to you? Who knows? So how could we measure your personal, subjective experience of the loudness of that horn? That question is the stuff of psychophysics, which relates the psychological experience to the physical stimulus.

Suppose that *A* and *B* are two sounds that differ only in loudness. They don't differ by much, so if you repeatedly present *A* to an observer (a participant in your experiment), and say, "Is this *A* or *B*?", the observer will sometimes mistakenly say *B*. Similarly, the observer will sometimes mistakenly identify *B* as *A*.



If the observer gets *A* and *B* confused often (is only right 50% of the time) then *A* and *B* must be very similar, at least to this observer. On the other hand, if *A* and *B* are very different, then the observer will respond correctly most of the time. So, at this point, we don't know how to quantify how different *A* and *B* are to the observer, but we can say that the more different *A* and *B* are, the more often the observer will correctly distinguish between them.

Now suppose that we have two more auditory stimuli, *C* and *D*, which also differ (from each other and from *A* and *B*) only in loudness, and they don't differ by much. Suppose that you ran the same experiment with *C* and *D* as you did with *A* and *B*, and the results were the same. We wouldn't be able to quantify *C* and *D*, but if the observer finds it as hard to tell *C* from *D* as *A* from *B* (the error rate in identifying *C* vs *D* is the same as the rate in identifying *A* from *B*), then the difference (to this observer) between *A* and *B* must be the same as the difference between *C* and *D*.

- ***Let* error_rate(*A*,*B*) *reflect the percentage of mistakes the observer makes when distinguishing between stimulus A and stimulus B. Thus an* error_rate(*A*,*B*) *of 25% means that 75% of the time,***

[3] This is also often identified as the start of modern psychology as a standalone field. *Fechnerfest* is still celebrated (with small parties or gatherings) by psychology departments at several universities.

the observer correctly identifies the stimulus as A or B and 25% of the time, the observer is mistaken.

- ***Under Fechner' s theory, two pairs of stimuli, (A,B) and (C, D) are equally different to an observer if that observer' s error_rate (A,B) = error_rate (C,D).***

In other words, two pairs of stimuli are equally similar (equally different) if we get them confused with each other equally often.

So, we don't know how loud *A, B, C,* or *D* are, but we now know how to meaningfully say that to a given observer, the difference in loudness between *A* and *B* is the same as the difference in loudness between *C* and *D*.

Let's establish one more convention.

- ***If error_rate (A,B) = 25%, we say that A and B are separated by one Just Noticeable Difference (JND).***

There's nothing magical about 75% correct. We could use some other number without affecting the theory, but 75% is the number Fechner chose. JND is the unit of measurement on Fechner's interval scale.

Suppose that you present two stimuli, *A* and *B*. But *A* is actually a silent stimulus, whereas *B* is a very quiet one. Adjust the intensity of *B* until finally, the observer distinguishes correctly between *A* and *B* 75% of the time. *B* is therefore one JND louder than silence (*A*).

Now compare *B* with a lot of slightly louder stimuli until you find one, *C*, that is one JND louder than *B* (correctly distinguished from *B* 75% of the time). *C* is thus two JNDs louder than *A* (silence).

Keep going, finding a *D* that is a JND louder than *C*, and an *E* that is a JND louder than *D*, etc., until finally you reach a sound that is too loud to listen to. The result is a scale that goes from silence to painful loudness.

We know the sound energy (measured, say, in decibels) of stimuli *A, B, C, D, E, F*, etc. Therefore we can plot a graph, called the psychophysical function, that shows sound energy on one axis and subjective loudness on the other. The subjective loudness of *A* is 0, *B* is 1 (one JND from silence), *C* is 2, *D* is 3, etc.

Now let's look at this in terms of the 8-factor approach to describing a theory of measurement.

- ***Attribute.*** So far, we have considered only one attribute, loudness. We could just as well consider heaviness, brightness, pitch, perceived severity of a crime, or many other subjective variables. For simplicity, I'll stick to loudness throughout this paper.
- ***Attribute' s Scale.*** We should probably represent the attribute on a ratio scale. However, the theory doesn't take us that far. The loudness of the sounds is expressed on an interval scale, whose units, by convention of the field, are called JND's.
- ***Attribute' s Variation.*** The same physical stimulus has varying subjective results. Sometimes a honking car sounds louder, sometimes softer, even though the honk is the same and even when you are as close to the car as you were the last time.

  If we think in terms of the physiology of the sensory system, this makes some sense. Many cells in the brain are activated by sound (by signals received from cells in the ear that are activated by changes in sound-related pressure) and more of these cells "fire" in response to louder stimuli. If the loudness of the sound is determined by the number of sound-related neurons that are firing, then the loudness will vary across presentations of the same stimulus. This is because the firing probability of a given cell is determined partially by the time since it last fired and the activity of neighboring cells, as well as by the amount of direct stimulation of this cell.

  If the loudness of a sound is determined by the total amount of neural activity associated with the sound, and if the individual neural cells might fire sometimes and might not fire other times in response to the same stimulus, then the loudness is a function of the sum of many random variables. If we assume that the variables are independent (a hazardous assumption, but that's the standard assumption of this class of theory at this point in this classic discussion) then the value of the sum is a random variable that is almost certainly normally distributed (distributed according to the Gaussian, bell-shaped curve.)

The theory of variability and error is illustrated in Figure 1, titled "Fechnerian Theory of Discrimination."

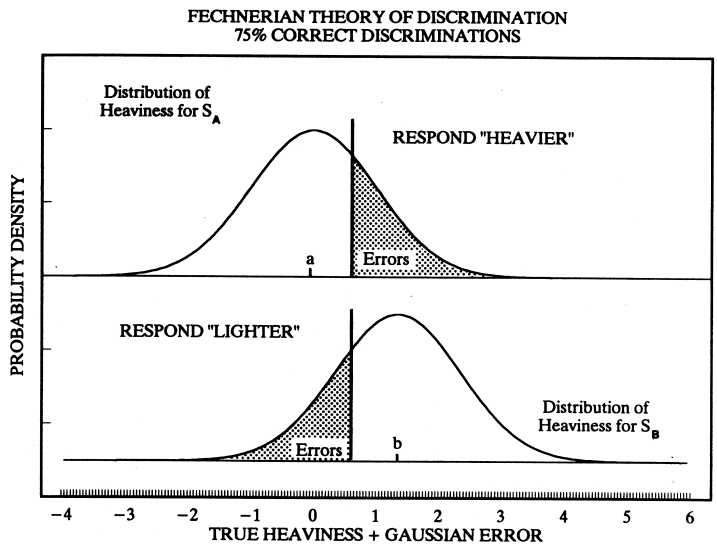
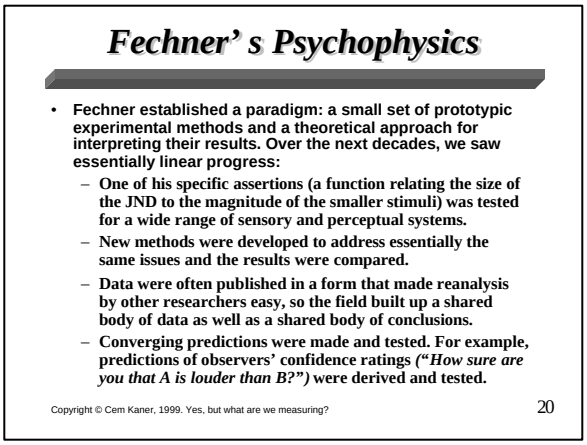


FIG. 1.1. The two Gaussian distributions characterize the distribution of sensory error about two *true* heavinesses, *a* and *b*, evoked by stimuli S_A and S_B. The area to the right of the midpoint, $(a + b)/2$, represents the probability of an error that applies equally to the judgment of either stimulus. The abscissa is scaled in standard deviation units.

Figure 1--These figures have been scanned in from Boring (1943) and Link (1992)

- ***Instrument.*** There were many different experimental techniques, and they yielded similar psychophysical functions. You might present single stimuli and ask which one (of two possibilities) had been presented ("Is this the loud one or the soft one?") or you might present them together (*A* then *B* or *B* then *A*) and ask whether the louder one came first or second, etc., etc., etc. At the core, though, the physical thing that we are varying is the sound energy (decibels) of the stimuli. A point of interest in this type of work is that we don't just do this test a few times. We might present *A* and *B* and *C* and *D* (etc.) hundreds or thousands of times to the observer, in order to obtain stable behavior (the person is not distracted by the mechanics of the experiment) and a more precise estimate of the difference between stimuli.



- ***Instrument's Scale.*** Percent correct, in an experiment involving stimuli that vary in sound energy (most often measured in decibels).
- ***Instrument's Variation.*** There is some variation in the physical stimulus, but we can keep it small by using precise equipment. Different experimental methods yield different patterns of response error, part of the reason that several different methods were used.
- ***Theory of Relationship.*** Associated with a stimulus, we have a subjective experience. There is variation in the experience, but there is an average value (at points *"a"* and *"b"* for stimuli *A* (S_A on the diagram) and *B* (S_B on the diagram) in the diagram above). We don't have a direct measure of the subjective loudness associated with *"a"* or *"b"* but we can measure how close *"a"* is to *"b"* by

considering how often the observer says *B* when the stimulus is *A* and how often the observer says *A* when the stimulus is *B*. The lower the error rate, the bigger the subjective distance between *A* and *B*. As *B* increases (in decibels), *"b"* (the average subjective experience associated with *B* (the physical stimulus), loudness increases too.

The psychophysical function relating sound energy and subjective loudness is approximately logarithmic, but is probably better shown as a power function *"b"* = B^x, where *x*, the exponent, is about 0.67. Different modalities (brightness, etc.) are characterized by different exponents.

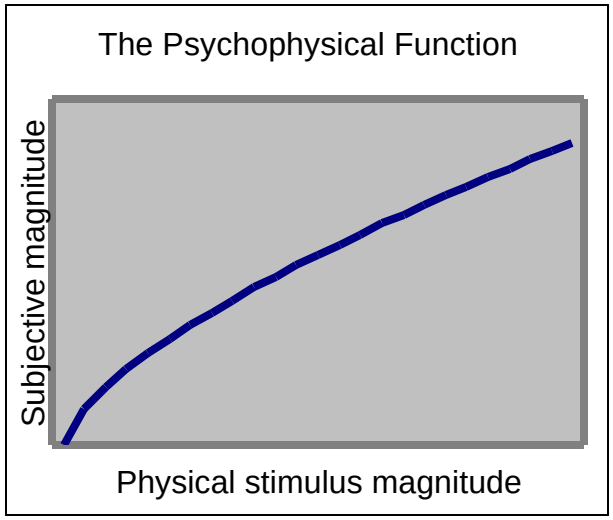


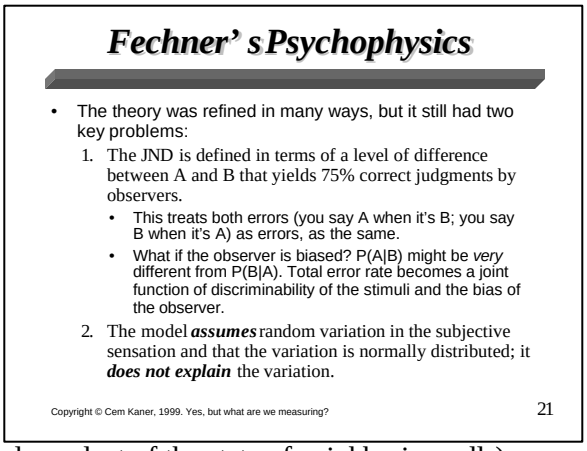
Figure 2 -- The Psychophysical Function

- ***Probable Side Effects.*** Fechner didn't anticipate side effects of the measurement. In particular, his approach does not anticipate that people would change how they respond and how they experience sounds (weights, etc) based on their participation in his experiments.

Despite the brilliance of the insight, Fechner's approach left some puzzles behind.

- The first puzzle is that people respond more quickly when the stimuli are very different from each other. A related finding is that, in general, the average response time is shorter for correct responses than for erroneous responses. How are these time-related results predicted by this theory? There have been several efforts made to extend Fechner's approach to response times, but I haven't found them compelling.

  If an observer's decision is based on comparison of the neural activity associated with stimuli *A* and *B*, then the amount of time taken to get a reading on *A* and *B* should be constant across different *A's* and *B's*. (If you count how many pulses occur in a certain time, the amount of time is a constant.) The decision time should not depend on the relative magnitudes (how much louder *A* is than *B*). We'll see a compelling alternative approach in Link's Wave Difference theory, below.



- Second, can we really represent the underlying probability distribution for sensory magnitude as a normal distribution? The assumption of independence (the probability that one cell will fire is independent of the state of neighboring cells) is wrong. The firing of one cell has a well-known effect on the probability that a neighboring cell will fire. (See, for example, the many studies of lateral inhibition, discussed in almost every introductory psychology text.) Without independent probabilities, the normal distribution is not mathematically implied.

  It is central to Fechner's theory that the underlying subjective experience associated with a physical stimulus be randomly variable, and the results obtained across several experiments were compatible with the hypothesis that the distribution of that variable is symmetric and bell-shaped, but there were no strong predictions from the specifics of the distributional shape about the data and therefore (Popper, XXX) this was not an essential part of the theory.

- The third puzzle is that Fechner treats all errors in the same way, but they are not the same. Suppose that A and B were correctly distinguished 75% of the time and that of the 25% of the judgments in which errors were made, 12.5% involved saying *A* instead of *B* (which was the stimulus presented) and 12.5% involved saying *B* instead of *A*. This is a very different pattern from 75% correct with 24% of the errors being *A* when *B* was presented and 1% being *B* when the observer should have said *A*. Are these two 75%'s the same? They are treated the same in the theory, which does not allow for response bias.

  In general, there is no theory of measurement error. Variation is attributed to variation in the sensory system, not to variation induced by the experiment.
- Finally, consider the core conjecture of the theory, that psychological difference between *A* and *B* is the same as the difference between *C* and *D* if error_rate (*A*,*B*) = error_rate (*C*,*D*). This is a conjecture, not a fact. A very plausible one, but it has been subject to direct challenges. (xxx Stevens, 1975, and other reports of difference-matching experiments. Check Cross's publications.)

An Aside

Here are a few things to notice before we go on.

- A measurement system doesn't start out fully formed. There are significant holes in the Fechnerian approach.
- At the core of the success of Fechner's approach is a strong statement of relationship between physical stimuli and the subjective experience being measured.
- Confronting the holes in Fechner's approach will lead us to substantially more powerful models.

Bias: Measurement Errors

When we ask an observer whether a stimulus was *A* or *B*, there are four results:

- The observer says *A* and it was *A*
- The observer says *A* and it was *B*
- The observer says *B* and it was *A*
- The observer says *B* and it was *B*

Fechner's approach assumes that there is no bias with respect to *A* or *B*. Figure 3 (Psychological Magnitudes) illustrates this assumption.

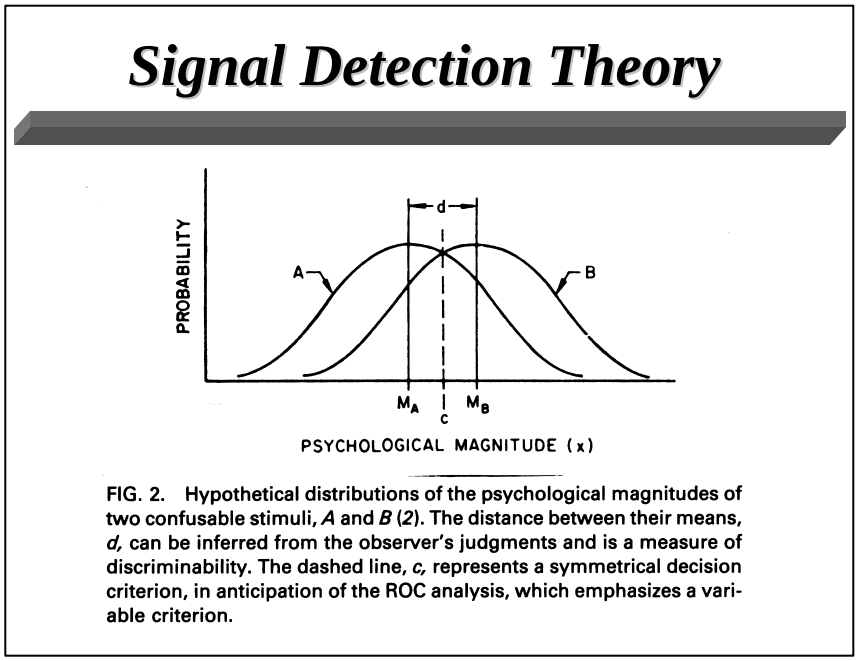


Figure 3 Psychological Magnitudes

The figure shows two probability distributions, one associated with stimulus *A*, another associated with stimulus *B*. The value M_A is the mean of the distribution associated with *A*. In other words, M_A is the average loudness experienced by the observer when presented with stimulus *A*. Similarly, M_B is the mean of the distribution associated with *B*. The average psychological distance between *A* and *B* is the difference between M_A and M_B, which is shown on the figure as *d*.

The point *c* is the midpoint between M_A and M_B. This point, *c*, is called the *criterion*.

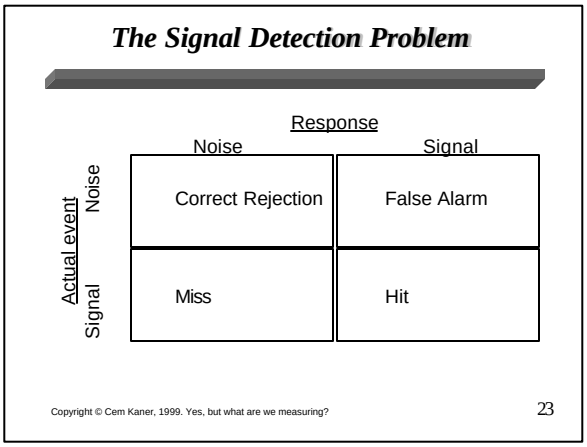
Here's how this works. When stimulus *A* (a tone) is presented, the observer hears something. If the perceived loudness of A is less than *c*, the observer says "That was an *A*." If the perceived loudness is greater than *c*, the observer says "That was a *B*." Because the criterion is halfway between M_A and M_B, the observer is just as likely to say *B* when the stimulus is an *A* as she is to say *A* when the stimulus is a *B*.

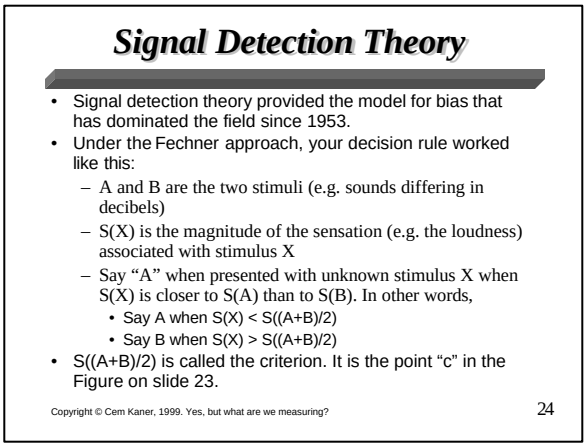
Now suppose that we create some payoffs. We'll pay $10 every time the observer says *A* when it is *A*, and we'll charge the observer $10 every time she says *B* when the stimulus was *A*. Additionally, we'll pay $5 when the observer gets *B* right and when the observer says *A* in response to *B*.

This payoff creates an incentive to guess *A*. If the observer guesses *A*, she makes $10 when she's right and loses only $5 when she's wrong. If she guesses *B*, she makes $5 when she's right, but loses the full $10 when she's wrong. Still, if the stimulus sounds very loud, so that it is almost certainly *B*, it would be foolish to say *A*.

The change in guessing strategy is naturally represented as a change in the location of the criterion. Move it to the right to a new point, *c'*, and the observer won't say *B* unless the stimulus sounds louder than *c'*. Now, if we present *A* 50 times and *B* 50 times, the observer might guess *A* 90 times and *B* 10.

By inducing the observer to change her response criterion, we can change her percentage and pattern of errors in response to the same pair of stimuli. The next fundamental advance in psychophysical theory addressed this issue head-on.



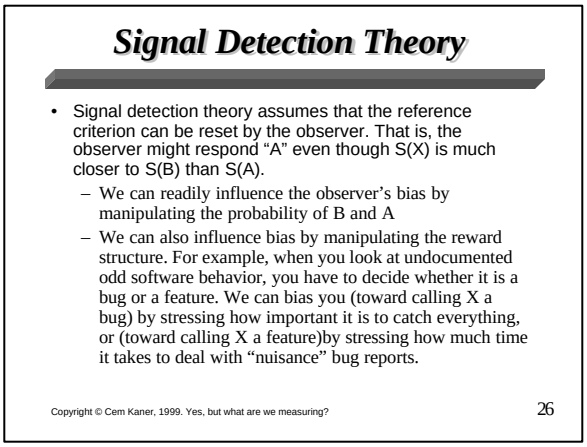


Signal Detection Theory

The classical discussions of signal detection theory start with discriminating between (telling the difference between) signals and noise. When you present a weak signal (tone) to an observer, the comparison isn't to true silence. Put yourself in a perfectly quiet room sometime and you'll hear your own breathing, your heart beating, and other small noises. A weak signal is noticed (or not) against a background of noise.

If you present an observer with a stimulus (either silence, *A* or a tone, *B*), the results are:

- The observer says *A* and it was *A* (correct rejection)
- The observer says *A* and it was *B* (missed the signal)
- The observer says *B* and it was *A* (false alarm)
- The observer says *B* and it was *B* (hit)



Using the analysis drawn in Figure 3 (Psychological Magnitudes), signal detection theorists use bayesian statistical methods to separately estimate the four conditional probabilities associated with hits, misses, correct rejections and false alarms.

Having separated out the four types of responses, the experimenter pushes around the observer's criterion, either by payoffs or by changing the relative probabilities of *A* and *B* (thus changing the expectation of the

observer.) This lets us separate out the effect of the criterion, in order to estimate the value of , *d'*, the distance between M_A and M_B.

Signal detection theorists plot Receiver Operating Characteristic curves to show this separation. This curve plots:

- **Probability of a False Alarm** (probability that the observer says *B* (signal) when *A* (noise) was in fact presented)
- **Probability of a Hit** (probability that the observer says *B* (signal) when *B* was in fact presented)

In Figure 4 (Receiver Operating Characteristics), start by looking at the line marked *d'=0.* The probability of a hit must be the same as the probability of a false alarm because there is no difference between stimulus *A* and stimulus *B.* (That's why *d*' is zero.)

Now look at the *d'=0.5* curve. *A* and *B* aren't very different in this case. (The distance between M_A and M_B is only 0.5 standard deviations of the distribution associated with *B*.) On this curve, when the probability of a false alarm is 0.10, the probability of a hit is about 0.2. So, if you find a pair of stimuli, *A* and *B*, that simultaneously yield a 20% hit rate and a 10% false alarm rate, here is what you know—if you can manipulate the observer's bias to yield a false alarm rate of 50%, the hit rate will be about 65%. (xxx get the exact numbers later). Once you know one point on the curve, you can predict the rest.

So, if you work with a pair of stimuli and obtain a hit rate of 53% (xxx) and a false alarm rate of 20%, you know that the *d*' value for these stimuli is 1.0 and that the hit rate will be 85% (xxx) when the false alarm rate is 60%.

These predictions were extensively studied and the results were largely in conformance.

By developing a theory to explain measurement errors associated with Fechner's psychophysics, we gained a much stronger predictor of a wider range of behavior.

Distributional Approaches

Signal detection theory assumes that the distribution of subjective magnitudes is normal but as leading theorists in the field note (xxx probably Green & Swets, 1966), this assumption is not essential. As long as the distributions are roughly symmetrical, the theoretical ROC curves will look much the same.

Signal Detection Theory

- Now the trick becomes to determine the distance between S(A) and S(B), given that biased observations make percent correct a misleading number.
- Notation:
  - **d'** is shown on slide 23 as the distance between S(A) (there called $\mathbf{M_A}$) and S(B) (there called $\mathbf{M_B}$). In other words, it is the distance between the average loudness of A and the average loudness of B.
  - If **d'**(A,B) is 2 and **d'**(A,C) is 4, then C is twice as much louder than A as B is.
- The goal is to measure **d'** accurately, without pollution by bias. We can do that by running several experimental sessions with the same stimuli, but with different biases. The pattern of change in P(A|A) and P(A|B) will reveal d'.

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 27

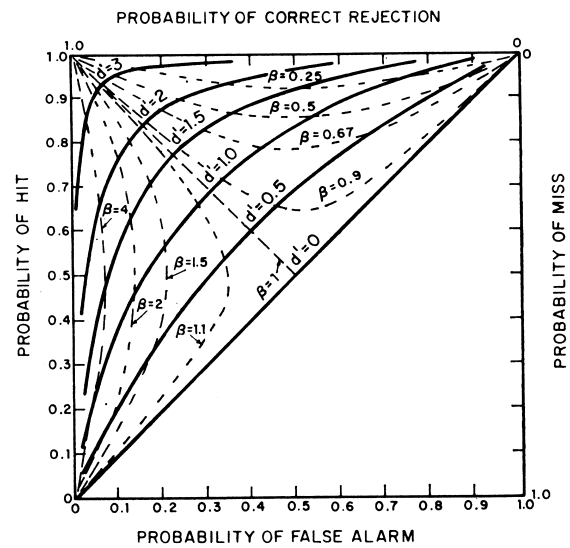


FIG. 7. A family of theoretical ROC curves based on normal, equal-variance distributions, with the parameter *d'* (solid lines). Also shown is a family of theoretical isobias curves, with the parameter *β* (dashed lines). The quantities shown on the left and lower coordinates are the two quantities ordinarily used in ROC analysis; the quantities shown on the right and upper coordinates are added here to point out that they are complements of the other two, respectively.

Figure 4 Receiver Operating Characteristic

Dealing with Variability

- When you are presented with a stimulus like a sound, the stimulus energy triggers neural discharges. The more neural activity, the louder the sound will be perceived as being.
- At any given time, several neurons are incapable of firing (typically because they fired recently and are in a refractory period). Therefore the number of neurons that can / will fire in response to a constant stimulus will vary over time.
- The number of discharges per unit time is often modeled as a Poisson process. Because of the nature of that probability distribution, we can approximate the variability of S(X) with a Normal distribution, but if we are really dealing with a perception that develops over time, we can do better by thinking of the process as a random walk.

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 29

Other approaches have focused on modeling the underlying mechanism. The wave theory (Link, 1992) is a strong example of this approach. The slides explain the theory well enough for this first draft paper. (In other words, I'm running out of steam.) The primary illustration here is that when by tackling yet another hole in the theory (the underlying variation of the attribute and the mechanism underlying the relationship between the attribute and the response), psychophysicists developed an approach that was stronger still.

Wave-Difference Theory

- In a given unit of time, there's a certain level of neural activity. If the experiment involves stimuli A and B (B is louder than A), then the level is either greater or less than our expectation from a stimulus of intensity (A+B)/2.
- Compute the difference between the moment's level and the level expected from (A+B)/2.
- Add up the differences over time. Eventually, the sum will become large positive or large negative. When the sum is large enough, say "A" if the sum is negative; say "B" if the sum is positive.
- This model predicts (quite well) a relationship between responses, response times, and confidence.

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 31

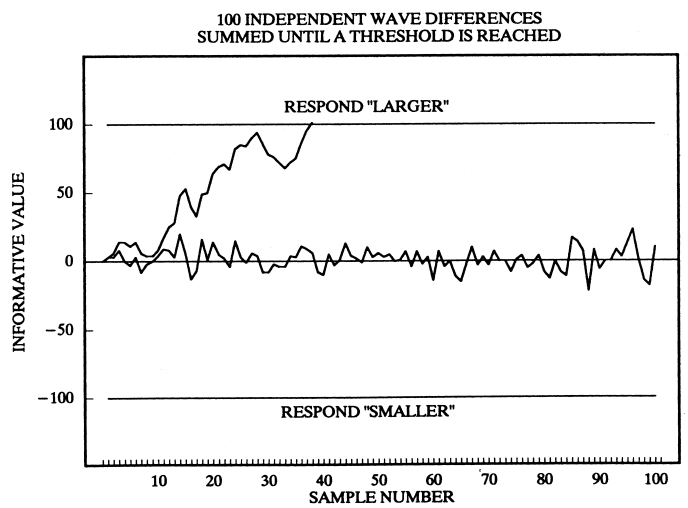


FIG. 9.4. Comparative differences between two random variables are Informative Values that are added sequentially to form a stochastic path to the response threshold at *A* from the starting position at 0.

Other distributional approaches

- Many other models have developed that take into account the sequential nature of a judgment in an experiment. For example,
  - It takes time for a stimulus to trigger neural activity
  - It takes time to make a decision
  - It takes time to actually make the response.
- These three times are variable, and their probability distributions are probably different. When we record response latency (time between the stimulus and the response), we are recording a sum of (at least) three random variables. If we have a theory of temporal variation for each part of the sum, we can compare the predicted distribution of response latencies to the distribution actually obtained.
- This is another example of using converging measures to check a theory of relationship.

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 32

Back To Testing

It takes a lot more than an easy-to-work-with surrogate and a plausible story to relate a proposed measure to an underlying attribute.

- We need a theory of the relationship between the measure and the attribute. Otherwise, what are we measuring?
- We need a clear idea of the probable side effects of a given measure, or we will probably do more harm than good.
- We need strong tests of the theory

Relate this back to Testing

" It takes a lot more than a plausible story to relate a proposed measure to an underlying attribute.

We need a theory of measurement (including a theory of the relationship between the measure and the attribute);

We need strong tests of that theory of relationship.

" Let s look back at code coverage as a measure of completeness of testing to see how much is missing

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 34

So what should we do in the absence of a strong theory of mechanism? From the point of view of limiting side effect risks, I think that we should use balanced scorecards.

For example, consider the problem of code coverage. 100% line coverage (or 100% line and branch coverage) have often been equated with 100% testing, or with good enough testing, at least.

Code coverage

Coverage measures of the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors:

» ***100% line coverage*** means that you tested for every bug that can be revealed by simple execution of a line of code.

» ***100% branch coverage*** means you will find every error that can be revealed by testing each branch.

» ***100% coverage*** *should* mean that you tested for every possible error. This is obviously impossible.

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 35

The Problem of Coverage

- Several people seem to believe that complete statement and branch coverage means complete testing. (Or, at least, sufficient testing.)
- Part of the rationale comes from IEEE Std. 982.1-1988, § 4.17, "Minimal Unit Test Case Determination"
- IEEE Unit Testing Standard is
- 100% Statement Coverage
- and 100% Branch Execution
- Most companies don't achieve this (though they might achieve 100% of the code they actually write.)

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 36

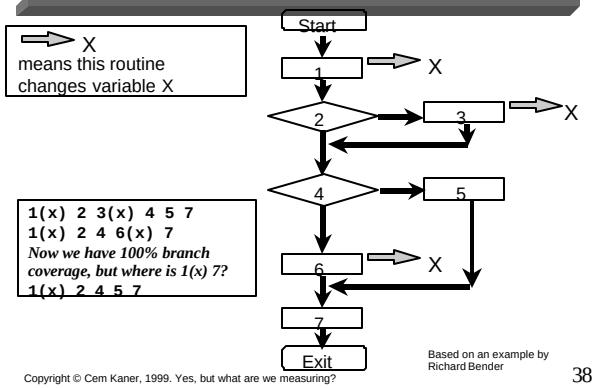
But line / branch coverage just test the flowchart

You're not testing:

» data flow
» tables that determine control flow in table-driven code
» side effects of interrupts, or interaction with background tasks
» special values, such as boundary cases. These might or might not be tested.
» unexpected values (e.g. divide by zero)
» user interface errors
» timing-related bugs
» compliance with contracts, regulations, or other requirements
» configuration/compatibility failures
» volume, load, hardware faults

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 37

Line coverage and data flows



Based on an example by Richard Bender

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 38

Lacking a theory, consider a balanced scorecard

"Coverage" measures are popular because they provide management with essential (if incorrect) feedback about the progress of testing.

Rather than reporting a single not-very-representative measure of testing progress, consider adopting a "balanced scorecard" approach. Report:

– a small number (maybe 5) of different measures,
– none of them perfect,
– all of them different from each other, and
– all of them reporting progress that is meaningful to you.

Together, these show a pattern that can more accurately reflect your progress.

For 101 examples of possible coverage measures, see "Software Negligence and Testing Coverage" at www.kaner.com

Copyright © Cem Kaner, 1999. Yes, but what are we measuring? 39

Side effects arise from using coverage metrics as measures of the extent to which some minimum amount of testing is complete. People focus on the count (how many lines tested) and are likely to miss many types of bugs.

As an exercise, try to analyze code coverage as a measure. Here are the 8 factors:

- ***Attribute.***
- ***Attribute' s Scale.***
- ***Attribute' s Variation.***
- ***Instrument.***
- ***Instrument' s Scale.***
- ***Instrument' s Variation.***
- ***Theory of Relationship.***
- ***Probable Side Effects.***

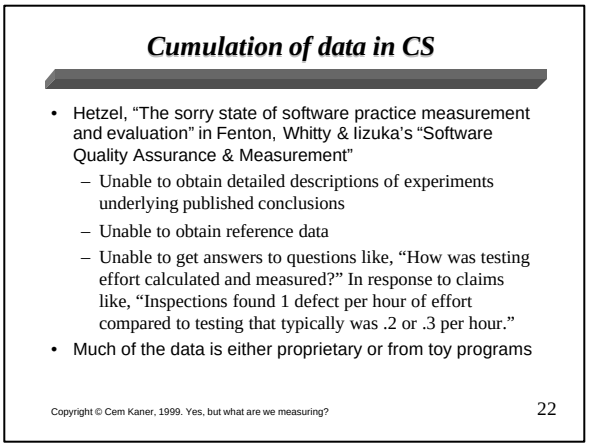
The notion of the balanced scorecard is that it reports several different types of indicators. The original scorecard had to do with overall health of a business. Rather than reporting (and optimizing for) a single variable such as price/earnings ratio, quarterly profits, personnel turnover, investment in R&D, etc., the scorecard reports on several (perhaps a dozen) different variables that address different dimensions of the health of the business.

A key advantage of the balanced scorecard is that it minimizes side effects. In the absence of a solid theory relating a measure (such as code coverage) to a key underlying attribute, the balanced scorecard provides a general way to get at the attribute being measured while limiting the distortion caused by a half-measure.

We Need Data

Bill Hetzel tried to find out the basis of published software measurement results. He asked direct questions of several leading people in the field.

- He was unable to obtain detailed descriptions of experiments underlying the results. This means that he cannot independently replicate the experiments.
- He was unable to obtain the reference data themselves. This means that he cannot independently reanalyze the results.
- He was not even able to get answers to questions like, "How was testing effort calculated and measured?" in response to claims like, "Inspections found 1 defect per hour of effort compared to testing that typically was .2 or .3 per hour."



Hetzel reported that much of the data that are available are from proprietary software or from toy programs. This certainly accords with my experience. And as a recent public example, Watts Humphrey presented some striking data at the QAI conference in Orlando, 1999. James Bach recently asked whether he could have access to the data and Watts said no, it is proprietary.

These problems are typical of applied research, whether we're dealing with computing or psychophysics.

Research labs in psychophysics operated differently. We ran experiments that collected huge amounts of data. My own dissertation was based on over 1,000,0000 judgments by observers. We ran these experiments to extend or clarify theory. It was common to reanalyze other researchers' data and to reinterpret their results.

I am concerned about the extent to which significant projects in universities have corporate partners and generate private data. It seems to me that we might be eating our seed corn.

(Some of the) References

Boring, E.G. (1917) “A chart of the psychometric function” *American Journal of Psychology,28*, 465-470.

Boring, E.G. (1943) *Sensation and Perception in the History of Experimental Psychology*. Appleton Century Crofts.

DeMarco, T. (1995), "Mad about measurement" in *Why Does Software Cost So Much?*, Dorset House.

Fenton, N.E. & S.L. Pfleeger (1997, 2nd Ed.) *Software Metrics: A Rigorous & Practical Approach*, PWS Publishing.

Fenton, N.E., S.L. Pfleeger, & R.L. Glass (1994) "Science and substance: A challenge to software engineers," *IEEE Software*, 11(4), 86-95.

Gescheider, G. (1997) *Psychophysics: The Fundamentals*, 3rd Ed., Lawrence Erlbaum.

Grady, R.B. & D.L. Caswell (1987) *Software Metrics: Establishing a Company-Wide Program*, PTR Prentice-Hall.

Green, D.M. & J.A. Swets (1966) *Signal Detection Theory and Psychophysics*, Wiley.

Johnson, M.A. (1996) *Effective and Appropriate Use of Controlled Experimentation in Software Development Research*, Master's Thesis (Computer Science), Portland State University.

Kaner. C. (1983) *Auditory and Visual Synchronization Performance Over Long and Short Intervals*, Doctoral Dissertation, McMaster University, Department of Psychology.

Kaner, C., J. Falk, & H.Q. Nguyen (1993, 2nd Ed.) *Testing Computer Software*, Wiley.

Link, S.W. (1992) *The Wave Theory of Difference and Similarity*, Lawrence Erlbaum.

Popper, K. (xxx) “Conjectures and refutations”

Stevens, S.S. (1975) *Psychophysics*, Wiley.

Thurstone, L.L. (1959) *The Measurement of Values*, University of Chicago Press.

Urban, F.M. (1930) “The future of psychophysics”, *Psychological Review,37*, 93-106.

Weinberg, G.M. & E.L. Schulman (1974) "Goals and performance in computer programming," *Human Factors*, 16(1), 70-77.

QW2000 Vendor Technical Paper 4V1

Steve Nemzer
(Veritest)

Zero Defect of Simship? The Global Testing Challenge

BACK TO QW2000 PROGRAM

Key Points

- Testing and localizing for global product rollouts
- Workflow
- International Quality Assurance

Presentation Abstract

The global e-commerce economy is here. Are your products localized, tested, and ready? Increasingly, sophisticated customers expect software applications to be available and fully functional in localized versions simultaneously with the English release.

Yet, few large IT companies have a clear strategy for testing their geo-specific SKUs simultaneously and consistently. Globalization tasks are often left to local subsidiaries who find an English language version "thrown over the ocean" to them for localization and testing. The dynamic nature of web-based products further complicates matters. Successful global product roll-outs require that localization testing methods incorporate workflow attributes.

In the seminar you will learn:

- How leading technology firms approach localization quality assurance
- Proven methods for releasing globally tested products
- When to apply the "Good Enough" test model, and when to apply "Four 9s" models
- When to use exploratory testing vs. when to use acceptance testing
- How to prevent defect propagation through results analysis and defect tracking
- How to utilize workflow systems for localizing and testing dynamic content
- How to apply metrics for evaluating both test teams and test projects

This seminar targets the test managers who must develop a strategy to balance rigorous international quality assurance with cost and time-to-market factors. It is appropriate for product managers, QA managers, and worldwide localization managers in the commercial software and hardware business, responsible for rolling out products to global markets. Presentation abstract to be supplied.

About the Author

One of the pioneers of outsourced testing, Steve Nemzer co-founded VeriTest in 1987 to provide services to market-leading hardware and software developers. Over the last 12 years, he has led VeriTest to prominence as the premier test lab to the IT community. VeriTest now operates internationally as a service of Lionbridge Technologies, a leading provider of test and localization services.

BACK TO QW2000 PROGRAM

Zero Defects or Simship? The Global Testing Challenge

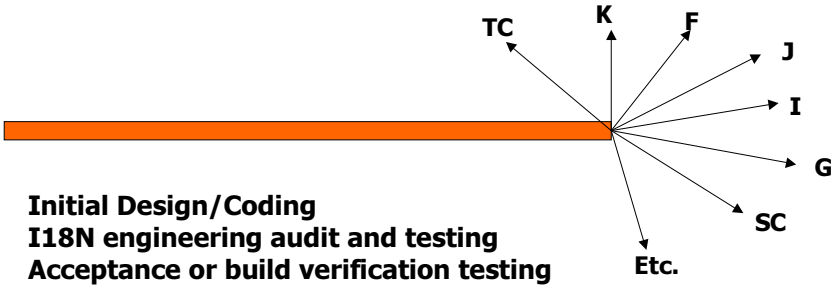
Presenter: Steve Nemzer, Vice President
VeriTest Business Unit
Lionbridge Technologies

VeriTest.

Presentation Topics

- The 'last mile' in L10n Software Quality
- Two Models for Software Quality Testing
- Globalization Test Planning Tactics
- Future Directions and Recap

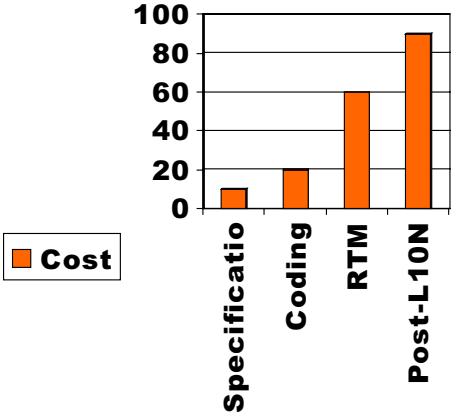
The 'last mile' of quality



Initial Design/Coding
I18N engineering audit and testing
Acceptance or build verification testing
Functional testing
Base language UI testing
Compatibility testing

DEFECT PROPAGATION!

Cost to fix defects vs. Dev Cycle phase



The last mile….

- Lack of economies of scale in "The Last Mile" has, in the past, made globalization solutions too costly for all but the largest vendors, and remains a primary constraint to global deployment.
- Globalization vendors, with combined L10n and comprehensive test services, have changed the game somewhat and do offer cost-effective means for shipping quality products.

Two Models of Software Quality

- 4 '9s' software
  - 99.99% uptime
- "Good-enough" software
  - Primary functions perform to user expectations
  - Defects in contributing functions don't add up to catastrophic failure

‘Good Enough’ software….not so good anymore

- Increasing pressure from consumers, and HQ-driven quality initiatives
- Transition from COTS model to data centers and ASPs
- High-tech ‘appliances’ penetrating into low-tech populace -- product complexity growing, user sophistication decreasing
- Open source movement
- Globalization of products

Four 9s methods have increasing relevance

- Thin client computing, ASP models, and migration of “micro” computers into the data center mean that when an application crashes, many users are affected
- Globalization processes create many code streams that need highly uniform test methods

Testing may be even more important for products deployed internationally…

- Tech support resources may be stretched thin across borders
- Cultural expectations for product quality may be greater internationally -- don't count on forgiving 'early adopters' to accept buggy products
- Platform and operating system variables add more potential for problems to the mix

The global environment of '00 requires a blended approach

- Initial Design/Coding
- I18N engineering audit and testing
- Acceptance or build verification testing
- Functional testing
- Base language UI testing
- Compatibility testing
- L10N process
- L10N UI testing
- L10N compatibility testing
- Documentation/Help testing

The global environment of '00 requires a blended approach

- Initial Design/Coding
  - real-time white box testing with unit test methods. Coders catch 97% of their own bugs
- I18N engineering audit
  - well-scripted in advance, bottom-up approach with robust methodologies
- Acceptance Test
  - well-scripted in advance

The global environment of '00 requires a blended approach

- Functional Testing
  - 'good enough' risk-based methods are only cost-feasible approach
- Base Language UI Testing
  - can be well-scripted in advance, using 4 9s approaches
- Compatibility Testing
  - 'good enough' risk-based methods are only cost-feasible approach

The global environment of '00 requires a blended approach

- L10n UI Testing
  - requires 4 9s approach, as same methodologies need to b applied across many variants
- L10n Compatibility Testing
  - requires 4 9s approach, as same methodologies need to be applied across many variants -- test plan metric is higher

"Good enough" Test Planning Tactics

- create a top-level function list
  - review of user documentation, menu choices, lists of reports, panels, dialog boxes, error messages, etc.
- Analyze inputs, boundary conditions, 3rd-party dependencies
- Spot good test cases quickly
- Reduce redundancies
  - sprinkled matrices

Special considerations for L10n Test Planning

- What is the level of I18n?
  - Was the product multi-byte enabled from scratch?
  - Is the program environment Unicode?
  - Has Resource / Code separation been implemented?

Special considerations for L10n Test Planning

- Has an I18N code review been completed? Review the assessment of:
  - String usage (buffer sizes, scanning, manipulation, sorting, comparing)
  - Code Page assumptions
  - Input/Output routines (Disk, Keyboard, Screen)
  - Font usage (direction assumptions, word wrapping, coordinate calculation)
  - Date/time/currency/code page assumptions

Special considerations for L10n Test Planning

- Is data available from beta programs? (technical as well as cultural feedback)
- Does legacy data need to be supported?
- If automated test scripts have been developed for E versions, are the scripts multi-byte enabled? Is valid localized test data available?

Special considerations for L10n Test Planning

- Compatibility Issues
  - Are local hardware variations (platform, peripherals, network, telecom) likely to create compatibility problems?
  - Will the product be used on both English and MBCS operating systems platforms? Don't forget all the Service Packs and SHWPs...
  - What about third-party software applications, such as 'plug-ins', management utilities, data import/export, and office suites? Will there be a mix of of E and localized products to interface with?

Schedule factors in L10n Testing

- Native speakers as testers need time to learn the technical side of the product
- Existing QA team has difficulty dealing with localized UI
- Language issues may necessitate modifications to UI

Measuring Testing Efficiency, Maximizing L10N Quality

- Pilot projects
- Post-mortems
- Involve regional staff
- Train development and QA staff in I18N and L10N issues
- Have linguistic expertise available for on-the-spot assistance with OOBE issues

Future Directions

- The ROI relative to globalization testing is great, and trends indicate it will increase over the next few years. Bug tolerant early adopters are history.
- The two quality and test models, 'Four 9s' and 'Good Enough', each have their place at distinct phases in the development cycle.
- As software deployment models change, and move to the data center and the ASP, reliability testing becomes more and more important - "Good enough" is not so good anymore.

Future Directions(b)

- The Internet, more than any other single exchange media, has forced us to rethink on testing issues, particularly for multi-byte locales. Global support requirements are evident in the current design philosophies embodied by HTML 4.0, Win2000 app specs. The Internet raises the bar for regional testing and support --an object can be downloaded from anywhere on the Internet and be expected to work in a menagerie of objects on your local machine.
- Interacting component-based applications (Active X, Java Applets, etc) add levels of complexity.

Resources

- Globalization & Localization: Best Practices for Windows 2000 Compliant Software, Microsoft Corp.
- Developing International Software, Nadine Kano
- Windows 2000 Application Specification, Microsoft Corp.

Resources

- www.unicode.org
- www.w3.org/International
- www.microsoft.com/globaldev
- www.digital.com/info/DTJB00
- www.java.sun.com/products/jdk/1.1/docs/guide/intl/index.html
- LISA web site

Tools

- I18n test tools
  - OneRealm Tools (www.onerealm.com)
  - Corel Catalyst 2.5
- L10n test tools
  - Rational tools (Robot)
  - Translation Craft HelpQA, HTMLQA
  - Translation Craft ToolProof
- Defect Management
  - TechExcel DevTrack

QW2000 Paper 4T1

Dr. Rainer Stetter
(Software Factory & ITQ GmbH)

Software Quality for Embedded Systems

BACK TO QW2000 PROGRAM

Key Points

- Problems and special quality demands on embedded systems
- Measurements, hints and practical solutions to improve quality in embedded systems
- Future quality approaches and techniques

Presentation Abstract

Embedded Systems is a prevalent term. Because of this I'll give an short overview of what do Embedded Systems means for us. In our context Embedded Systems mostly are parts of machines (e.g. tool machines or large manufacturing plants).

Therefore Embedded Systems are a conjunction of software and hardware and have to interact with a huge number of other components (mechanical, electrical, electronic, real-time software, non real-time software). Our experience is based on our university background and our work together with major corporations during the last ten years. The quality of the software in this kind of systems has a major influence on the quality of the whole system.

The system (this means the machine/plant) typically costs about 500.000 $ up to several million of $'s. Additionally there are safety critical elements, especially in any conjunction with medical devices.

About the Author

At the Technical University, Munich I studied mechanical engineering and as I was interested in software engineering I took some classes in computer sciences. While I was doing my PhD in developing a robot simulation system, which I got in 1993, I improved my knowledge in software engineering. From 1993 until 1997 I was working as a Research & Development Manager at Zwick Company, Ulm - Germany.

Since 1997 I have been one of the General Managers of Software Factory GmbH, Munich. In addition, since 1998 I have been working with the Munich based firm itq GmbH as a General Manager. Together, Software Factory GmbH and itq GmbH form the Software Quality Center. With some partners of the Technical University of Munich and VDMA (German Machinery and Plant Manufacturers' Association), we work on approaches to improve the quality especially in the field of embedded systems.

Software Quality for Embedded Systems

Dr.-Ing. Rainer Stetter

Software Quality Center

ITQ

Information: Technology: Quality

- Project management
- Quality planning
- TestCenter
- IT-Consulting
- Quality assurance
- Process improvement

SOFTWARE SF FACTORY

Software Quality Center

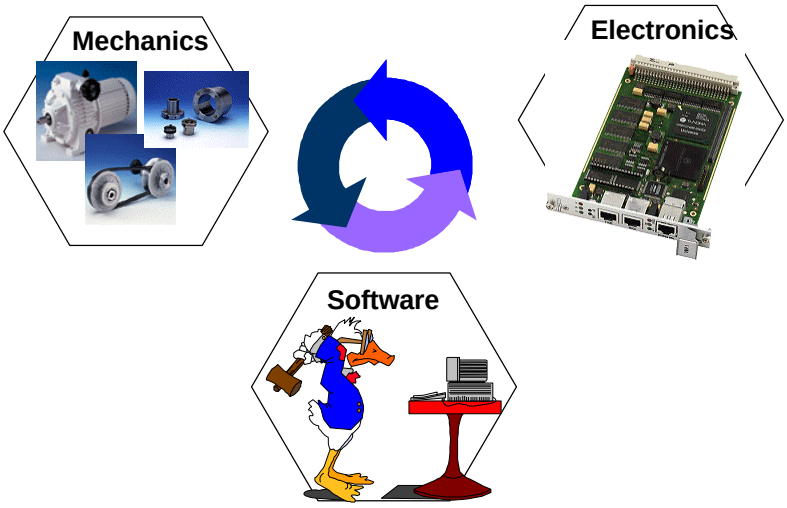
- Information systems
- Factory automation
- Datamanagement
- IT-Systems
- CAD/CAM

Example: Personalization of Smart Cards

Example: Material Testing

Example: Airbag production

Embedded Systems



Quality Requirements

- Mission critical applications
  - Production of the plant
  - Medical devices
- Safety critical
  - Dates and informations
  - Life and health
- Quality Requirements according to the FDA
  - stable and detailed process
  - Validation plan

Challenges

- Huge machine <-> untouchable software
  - Underestimation of the CEO
- Different worlds (mechanics, electronics, software)
  - Different traditions and way of thinking
  - Different levels of standardisation
  - Different organisations/ departments
  - Esteem of each other
  - Different „languages“
  - Different possibilities to visualize <-> level of abstraction
  - Different life cycles
  - Fear of new techniques

Levels of maturity

Mechanical engineering

Electronical engineering

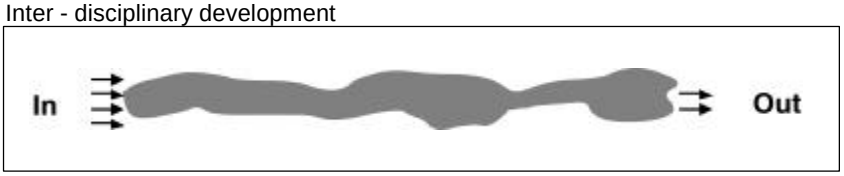
Software engineering

=

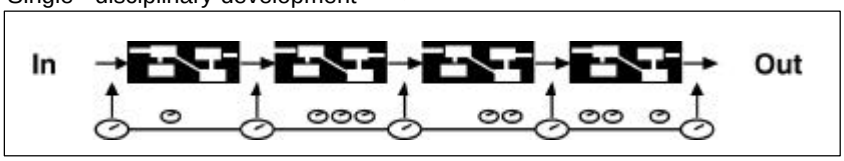
Software Quality for
Embedded Systems

Comparison

Inter - disciplinary development

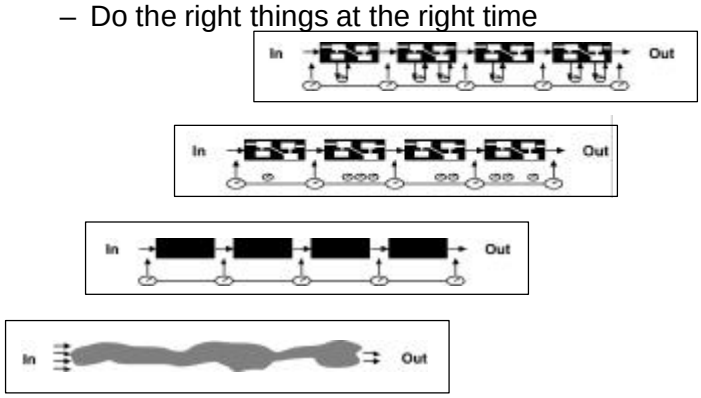


Single - disciplinary development



Precondition for success

- Change process from chaos to control
- Measure maturity of organization
  - Do the right things at the right time



What are we working on ?

- Form a community
  - VDMA – German association for machinery and plant manufacturing
    - Software
    - Hardware
    - Research
- Transfer and adept methods and approaches of computer science
  - CMM
  - UML
- Implement stable processes
  - Maintain test center for clients
  - Coach and guide

Measurements, techniques and tools

- FDA demands
  - Validation plan
    - Extensive reviews at all phases
      - Requirements, specifications, design, code
- Separate test team
  - Planning tests
  - Testing of hard - and software
  - Code coverage with CodeView
- Modeling system with UML (C++)
  - Rose98 with Reverse Engineering
  - Documentation with SoDa

What will we do next?

- Analyse usefullness of formal methods in our field
  - E.g. requirements engineering
- Case study – „Ideal process and techniques“
  - Toy race track
- Maturity model for embedded systems
  - Special needs for interdisciplinary development
- Requirements for tools
  - Configuration management
    - CAD, Source, Documentation

Conclusions

- Importance of embedded systems
  - Is enormous already
    - But is not yet seen of everyone
  - Will increase dramatically in the next ten years
- Knowledge and practices of computer science
  - Has to be transferred to traditional fields of engineering
  - Has to be adapted to special needs
- Main problem
  - Bring together different worlds
    - Social aspect is very important for success

QW2K Paper 4A1

Mr. William Lorensen & Mr. James Miller (GE Corporate Research & Development)

Visualization Toolkit Extreme Testing: A Production Release Every Day

BACK TO QW2000 PROGRAM

Key Points

- Automated software testing for medium-sized software with concise summarization of testing results
- The process provides a daily assessment of the readiness of the system for release
- The process empowers developers to improve the software without fear of breaking it

Presentation Abstract

Every large software development requires a process to move from requirements through design, implementation, testing and maintenance. Although tools and methodologies exist to carry out each of these steps, the degree to which these tools are applied is usually dictated by the size of the software product. The Capability Maturity Model developed by the Carnegie Melon Software Engineering Institute defines five levels to the software process. For the most part, this discipline is applicable to large software development projects that have a separate software quality assurance and testing organization. There are smaller projects that do not have a separate organization, but still need to apply sound software testing to achieve low defect software.

In this paper we describe a practical, automatic software testing process for a medium sized software package. The software is the Visualization Toolkit http://www.kitware.com/vtk.html, an open source, C++ class library consisting of about 600 classes. Vtk's class library supports visualization, graphics and imaging. It runs on multiple vendor Unix platforms and Windows 95/98/NT.

Author Bio...

Bill Lorensen is a Graphics Engineer in the Electronic Systems Laboratory at GE's Corporate Research and Development Center in Schenectady, NY. He has over 30 years of experience in computer graphics and software engineering. Bill is currently working on algorithms for 3D medical graphics and scientific visualization. He co-developed (with Harvey Cline) the marching cubes and dividing cubes surface extraction algorithms, two popular isosurface extraction algorithms. His other interests include computer animation, information display, and object-oriented software tools.

Bill is the author or co-author of over 60 technical articles on topics ranging from finite element pre/postprocessing, 3D medical imaging, computer animation and object-oriented design. He is a

co-author of "Object-Oriented Modeling and Design" published by Prentice Hall, 1991. He is also co-author with Will Schroeder and Ken Martin of the book "The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics" published by Prentice Hall in November 1997. The text describes vtk, an open source visualization C++ class library. He gives frequent tutorials at the annual SIGGRAPH and IEEE Visualization conferences.

Bill holds twenty six US Patents on medical and visualization algorithms. In 1991, he was named a Coolidge Fellow, the highest scientific honor at GE's Corporate R&D. Prior to joining GE in 1978, he was a Mathematician at the US Army Benet Weapons Laboratory where he worked on computer graphics software for structural analysis. He has a BS in Mathematics and an MS in Computer Science from Rensselaer Polytechnic Institute.

BACK TO QW2000 PROGRAM

Visualization Toolkit Extreme Testing

A production release every day

http://vtk.scorec.rpi.edu/Nightly/MostRecentResults

Bill Lorensen
Graphics Engineer

Jim Miller
Computer Scientist

n

Software Development Team

GE CRD

- *Majeid Alyassin*
- *Rick Avila*
- *Rusty Blue*
- *Jon Davis*
- *Tim Kelliher, MBB*
- *Annie Kelly*
- *Steve Linthicum, MBB*
- *Bill Lorensen*
- *Jim Miller*
- *Matt Turek*
- *Boris Yamromn*

GE Medical Systems

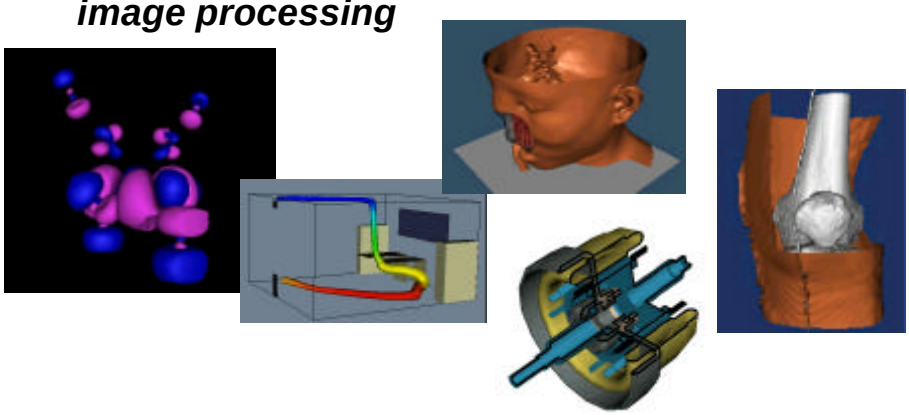
- *Dave Deaven*
- *Darin Okerlund*
- *Chris Slack*
- *Bill Stovel*

Kitware, Inc

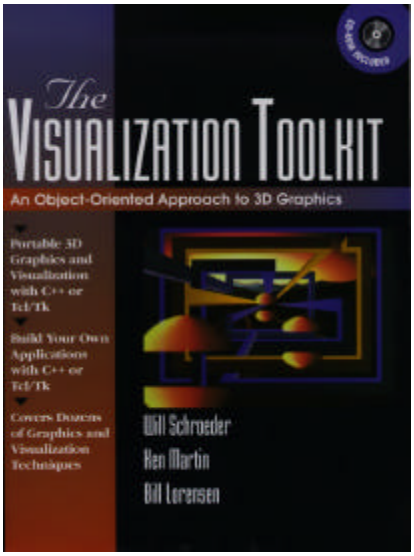
- *Lisa Avila*
- *Bill Hoffman*
- *Charles Law*
- *Ken Martin*
- *Will Schroeder*

Visualization Toolkit

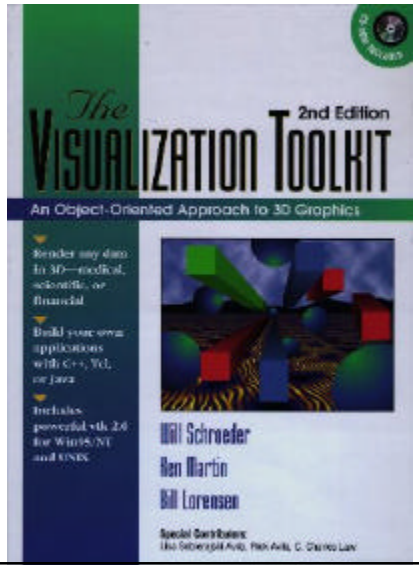
- *Open source toolkit for scientific visualization, computer graphics, and image processing*



vtk1.0 1995



vtk2.0 1997



Visualization Toolkit

VTK is used by

- GE Medical Systems
  - Platform software and applications, 1996
- GE Medical Systems CT
  - Volume visualization on the scanner, live 1999
- GE Aircraft Engine
  - Engineering productivity tools, 1996
- Lockheed Martin
  - Composite inspection system for JSF, live 1999
- DARPA
  - Virtual Endoscopy, 1996
- FBI
  - Facial Reconstruction, 1999
- Engineering Animation
  - Swept Surface, 1998

Visualization Toolkit

External "customers"

- Visual Interfaces Inc.
- Principia Mathematica Inc.
- Numerical Objects Inc.
- Brigham and Womens Hospital
- Rensselaer Polytechnic Institute
- NCSA
- Los Alamos National Lab, Argonne

- 900 people on mailing list
- 12 Universities using VTK textbook

Visualization Toolkit

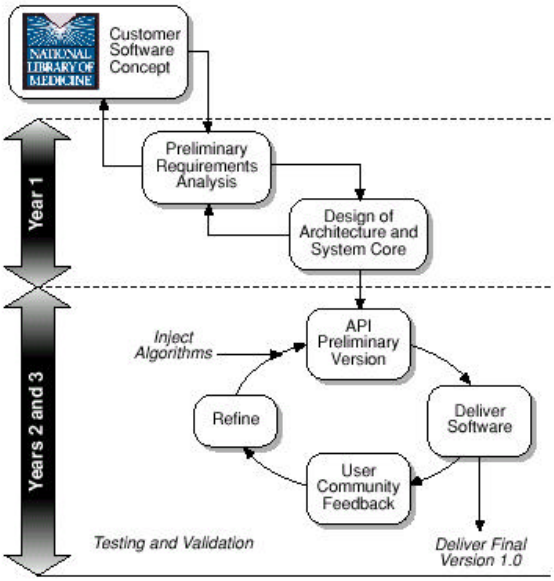
- Open Source
- 600 C++ classes
- 250,000 Lines of Code (100,000 executable)
- 20+ developers
- 5 years of development

"We dont sell VTK, we sell what we do with VTK."

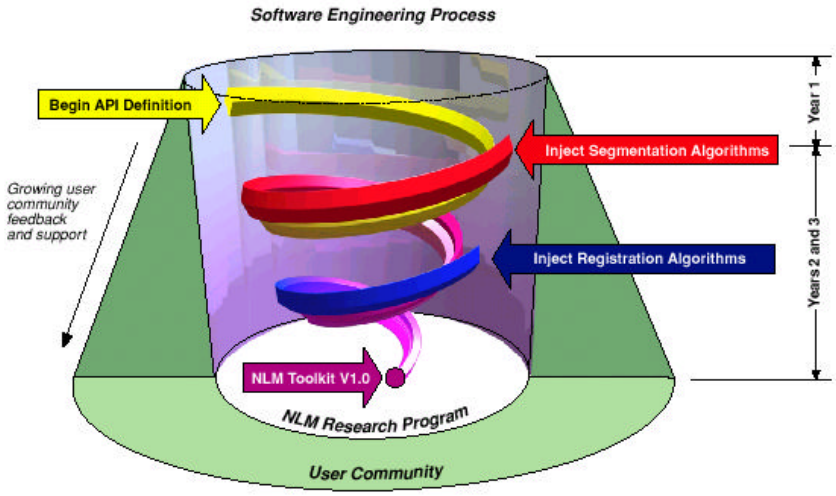
Testing Motivation

- Nature of our business
  - People working on projects for many different customers
  - Leading edge algorithm and software development
  - Development cycle does not fit "nicely" into a standard release schedule
    - Partly because VTK is not our "product"
- Software is dynamic.
  - Code worked when written, may not work now
- Need "release on demand"
  - Requires continuous software quality assurance

In Search of New Software Engineering Processes



GE CRD


Software Engineering Process
Begin API Definition
Inject Segmentation Algorithms
Inject Registration Algorithms
Growing user community feedback and support
NLM Toolkit V1.0
NLM Research Program
User Community
Year 1
Years 2 and 3

GE CRD

COMPUTER
Stumbling toward a Privatized DNS
Software Engineering in the Small
MEMS Move the World
The Wireless Web
Code of Ethics for Software Engineers, p. 84

IEEE Computer October, 1999

GE CRD

Cover Feature

Embracing Change with Extreme Programming

Extreme Programming turns the conventional software process sideways. Rather than planning, analyzing, and designing for the far-flung future, XP programmers do all of these activities—a little at a time—throughout development.

Kent Beck
First Class Software

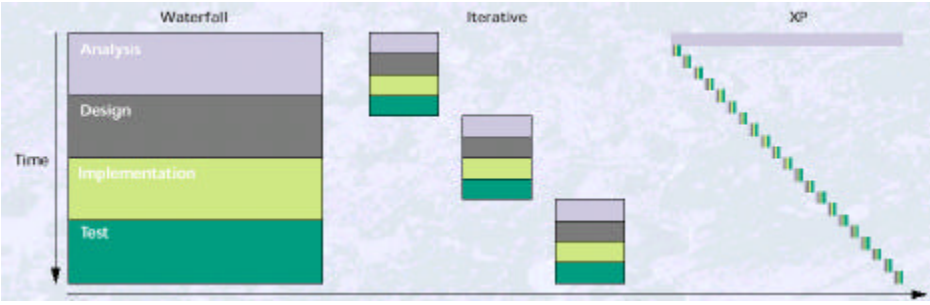
In the beginning was the waterfall (Figure 1a). We would get the users to tell us once and for all exactly what they wanted. We would design the system that would deliver those features. We would code it. We would test to make sure the features were delivered. All would be well.

shows, the waterfall began iterations.

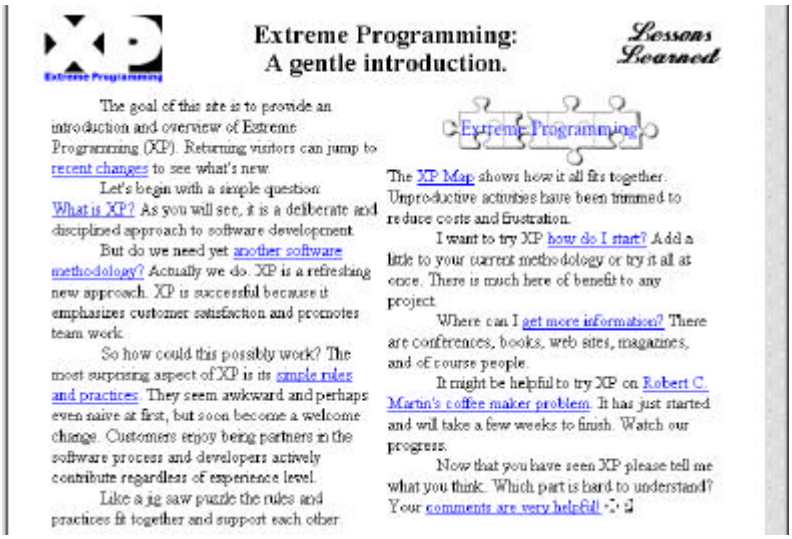
The waterfall model didn't just appear. It was a rational reaction to the shocking measurement that the cost of changing a piece of software rose dramatically over time. If that's true, then you want to make the biggest, most far-reaching decisions as early in the

GE CRD

Extreme Programming



www.extremeprogramming.org



Extreme Programming: A gentle introduction.

Lessons Learned

The goal of this site is to provide an introduction and overview of Extreme Programming (XP). Returning visitors can jump to recent changes to see what's new.

Let's begin with a simple question: What is XP? As you will see, it is a deliberate and disciplined approach to software development.

But do we need yet another software methodology? Actually we do. XP is a refreshing new approach. XP is successful because it emphasizes customer satisfaction and promotes team work.

So how could this possibly work? The most surprising aspect of XP is its simple rules and practices. They seem awkward and perhaps even naive at first, but soon become a welcome change. Customers enjoy being partners in the software process and developers actively contribute regardless of experience level.

Like a jig saw puzzle the rules and practices fit together and support each other.

Extreme Programming

The XP Map shows how it all fits together. Unproductive activities have been trimmed to reduce costs and frustration.

I want to try XP how do I start? Add a little to your current methodology or try it all at once. There is much here of benefit to any project.

Where can I get more information? There are conferences, books, web sites, magazines, and of course people.

It might be helpful to try XP on Robert C. Martin's coffee maker problem. It has just started and will take a few weeks to finish. Watch our progress.

Now that you have seen XP please tell me what you think. Which part is hard to understand? Your comments are very helpful!

Testing Constraints

- ***We only have 10 active developers, spread across many projects***
  - ***Cant afford separate SQA division***
- ***We dont have dedicated testing hardware***
  - ***Testing cannot hinder project work***
- ***We are "algorithm developers" not "software engineers"***
  - ***Testing must be automated and concise***

Testing Design Goals

- Frequent testing
  - Identify defects as soon as they are introduced
  - Too hard to find cause if not done frequently
- Automated testing
- Automated report generation/summaries
  - Must be concise yet informative
- Minimally invasive to daily activities
- Track testing results over time

Testing Terminology

- Regression Test
- Smoke Test
- White Box vs Black Box Test

Regression Test

What

- test code and data that reproduces an expected result
- usually hand crafted (white box testing)

Why

- assure that changes made to software do not introduce new defects

Who

- developer
- team

When

- throughout the life cycle

Smoke Test

What

- a quick sanity check
- a single compile, link and run

Why

- show minimum capability

Who

- developer
- test process

When

- anytime a change is made

White Box vs Black Box

White Box

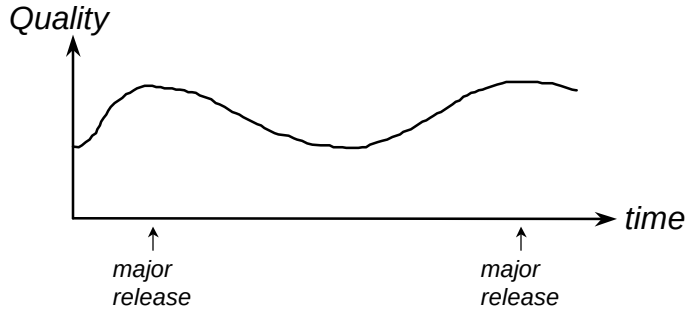
- exercise procedural control points
- requires knowledge of individual classes/methods
- hand crafted
- logic driven

Black Box

- automatically created
- limited knowledge of class details
- test cases derived from specs
- data driven

Prior to 1998

- 100 regression tests
- Performed manually and infrequently
  - just before a major release (6 months)



1998 Brought 14 Green Belt Projects

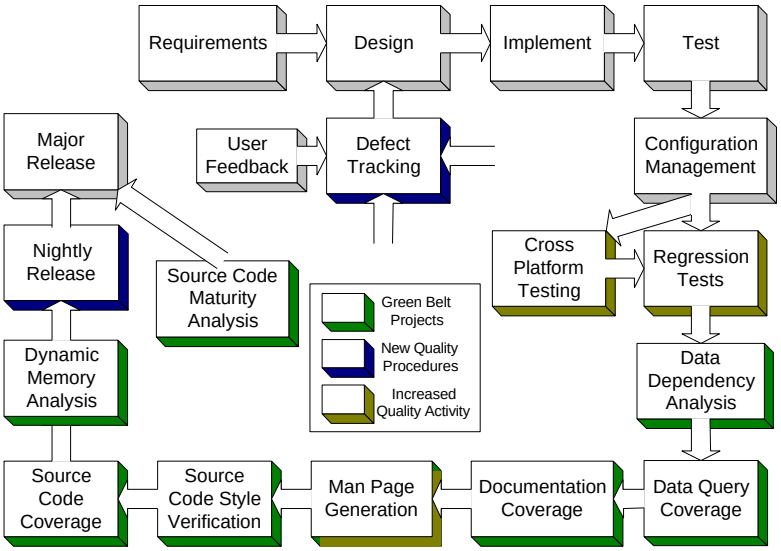
MAIC applied to various components of the software, i.e.

- ***Dynamic memory analysis***
- ***Static code analysis***
- ***Coverage analysis***

Each of these projects yielded measurable (albeit temporary) improvement.

Next step was to apply these tests as frequently as possible

Visualization Toolkit (VTK) Software Process



A day in the life of VTK Quality

The day starts at 8pm (EST)

- Determine what has changed in the system
- Update the testing system's version of the software
- On 12 different system configurations, we
  - Build the software
  - Run 500 regression tests
- Dynamic memory analysis
- Coverage analysis
- Check on coding style and documentation

8:01pm - Change Log

- Every day we summarize what files changed
- This tightens the "cause and effect" cycle

There were 45 files changed since the last dashboard.

```
Changes for Wed Jan 6 22:02:59 EST 1999

U common/vtkVersion.h
  revision 1.55
date: 1999/01/07 03:05:44;  author: millerjv;  state: Exp;  lines: +3 -3
Automated commit to force versioning.
=============================================================================

U common/vtkHexahedron.cxx
  revision 1.57
date: 1999/01/06 15:07:45;  author: millerjv;  state: Exp;  lines: +3 -4
ENH: speed enhancements to Locators
=============================================================================
```

8:05pm - Build Log

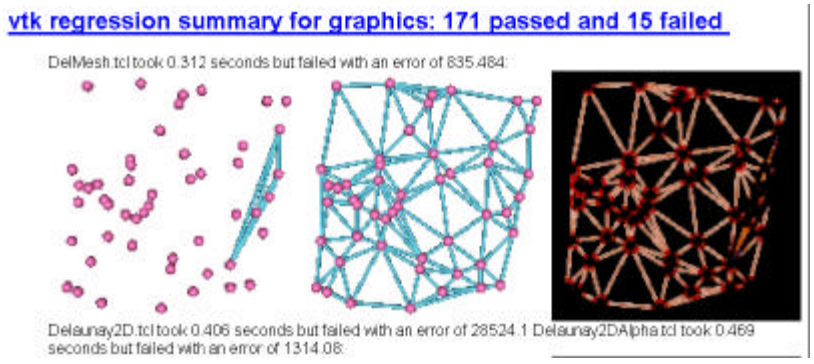
- *Summarize errors and warnings*

Platform	Build VTK	
	Errors	Warnings
irix6	0	286
irix6n32	2	2419
solaris	0	16273

- *Keep complete log of errors and warnings*

9:00pm - Regression Tests

- *Each regression test produces an image which is compared against a "gold standard image"*



Regression Tests

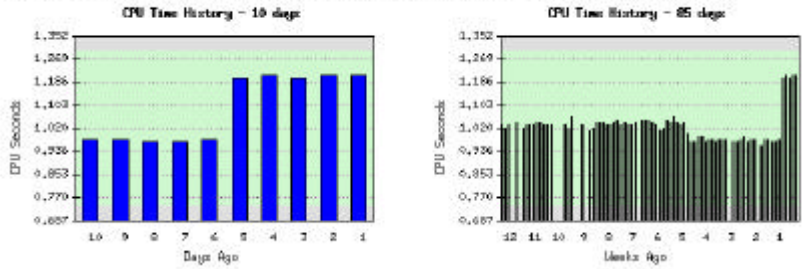
- *Image comparison done automatically*
- *Defects quantified*
- *Summary produced*

Platform	Build VTK		Test Tcl				Test Cxx	
	Errors	Warnings	Passed	Failed	Faster	Slower	Passed	Failed
irix6	0	286	355	0	7	4	44	2
irix6n32	2	2419	355	0	1	0	46	0
solaris	0	16273	341	2	3	1	41	0
solarisCoverage	0	10	353	2	0	8	45	0
WinNT	Catastrophic failure.		Catastrophic failure.					
hp	0	4955	0	396	0	0	0	630
linux	0	10	Catastrophic failure.					
solaris26	0	0	310	9	0	0		

Regression Tests

- *CPU time is recorded and compared to history*
- *"Defect" is test that takes "statistically" more time than usual*

VisQuad.tcl - 1.3794 wall, 1.2100 cpu, 0.5398 imgdiff, Passed Warning: Recent Time Increase.



1:00am - Coverage Analysis

- ***Identify what lines of code are tested***
- ***"If it isnt tested, it doesnt work."***
- ***Summarize coverage across VTK, across each kit, and across individual files.***

Coverage results: 72.70%, tested: 52088, untested: 19562, total: 71650

vtkActor2D.cxx	: 78.12%
vtkActor2DCollection.cxx	: 84.21%
vtkAttributeData.cxx	: 88.51%
vtkBitArray.cxx	: 69.23%
vtkByteSwap.cxx	: 57.14%
vtkCell.cxx	: 80.87%
vtkCellArray.cxx	: 81.25%
vtkCellData.cxx	: 5.26%
vtkCellLinks.cxx	: 64.10%
vtkCellTypes.cxx	: 90.48%
vtkCharArray.cxx	: 58.41%
vtkCollection.cxx	: 80.00%
vtkContourValues.cxx	: 91.49%
vtkCoordinate.cxx	: 36.27%

Annotated Coverage

Executable & Covered

```
    28    numNewLines = 0;
    28    numNewPolys = 0;
    28    polyAllocSize = 0;
    28    for (inStrips->InitTraversal(); inStrips->GetNextCell(npts,pts); )
            {
```

Executable & Not Covered

```
######      numNewPts += (npts-2) * 3;
######      polyAllocSize += (npts - 2) * 4;
######      }
    28    for (inStrips->InitTraversal(); inStrips->GetNextCell(npts,pts); )
            {
######      for (j=0; j<(npts-2); j++)
              {
######        p1 = inPts->GetPoint(pts[j]);
######        p2 = inPts->GetPoint(pts[j+1]);
```

Not Executable

```
            //
            // Update self and release memory
            //
    28    output->SetPoints(newPoints);
    28    newPoints->Delete();
            void vtkShrinkPolyData::PrintSelf(ostream& os, vtkIndent
indent)
     1  {
     1    vtkPolyDataToPolyDataFilter::PrintSelf(os,indent);
     1    os << indent << "Shrink Factor: " << this->ShrinkFactor << "\n";
            }
```

1:00am - Style

- Check coding conventions
- Detect code which has the potential for a future defect
  - Someone other than the original author may be modifying the code
- Summarize overall style and style per file

Style test results: 53 defective files out of 972 files, 757 defects out of 104876 opportunities.

Style Rule	Opportunities	Defects
Class name prefix "vtk"	507	0
Class name alphanumerics	507	0
Instance variable begins with uppercase	2222	15
Protected member data	2222	153
Member function begins with uppercase	8590	0
Dereferencing member data	33641	249
Dereferencing member function	24656	85
Braces around "if", "else", "for", ...	20721	188

1:00am - Memory Analysis

- Check for memory leaks, array bound writes, uninitialized memory reads, etc.

Purify Total Critical Defects: 0. (Purify last run on Jan 6 06:11)

```
VTK Dynamic Code Analysis for Thursday January 07, 1999 at 07:35 AM
================================
Total Memory Leaked: 0 bytes
Number of Leaks    : 0
Number of ABRs     : 0
Number of ABWs     : 0
Number of CORs     : 0
Number of FMMs     : 0
Number of FUMs     : 0
Number of FMRs     : 0
Number of FMWs     : 0
Number of MAFs     : 0
Number of UMCs     : 213
Number of UMRs     : 144

Purify Total Critical Defects: 0
```

7:00 am - Tests complete

- *Top level Web-dashboard constructed*
- *Testing results packaged and exported*
- *Software packaged and exported for world wide distribution*

7:00am - A Good Day

There were 32 files changed since the last dashboard. Changes for the month.

Platform	Build VTK		Test Tcl				Test Cxx	
	Errors	Warnings	Passed	Failed	Faster	Slower	Passed	Failed
irix65	0	22	416	0	3	5	49	0
irix6n32	0	5210	416	0	0	0	49	0
irix6n64	0	445	416	0	0	0	49	0
solaris	6	19815	394	2	0	13	44	0
solaris27	0	4064	416	0	8	3	49	0
solarisCoverage	0	679	412	3	0	5	49	0
Win47	0	2	414	2	0	0		
hp	0	11729	413	3	1	2	48	1
linux8432	0	298	415	1	0	4	49	0
solaris26ed	0	4070	408	8	0	3	47	2
solaris26	0	0	402	1	0	0		

Key:	Internal Test System	External Test System	System or Network Error	Build/Test Successful	Build/Test Error	Timing Change

Real work begins at 7:05am

7:00am - A Bad Day

Platform	Build VTK		Test Tcl				Test Cxx	
	Errors	Warnings	Passed	Failed	Faster	Slower	Passed	Failed
irix6	0	286	355	0	7	4	44	2
irix6n32	2	2419	355	0	1	0	46	0
solaris	0	16273	341	2	3	1	41	0
solarisCoverage	0	10	353	2	0	8	45	0
WinNT	Catastrophic failure		Catastrophic failure					
hp	0	4955	0	396	0	0	0	630
linux	0	10	Catastrophic failure					
solaris26	0	0	310	9	0	0		

Key:	Internal Test System	External Test System	System or Network Error	Build/Test Successful	Build/Test Error	Timing Change

We are "prisoners of quality"...

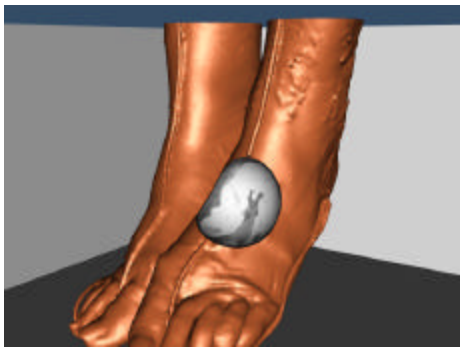
How do we use the system?

- *Track the effects of major changes*
- *Identify what needs to be changed*
- *Portability leads to quality*
- *Navigate software features*

Has truly "Changed the way we work."

Lessons Learned

- *Frequent testing*
- *Automation*
- *Summarization*
- *Navigation*
- *Cross platform*



www.mozilla.org

Continuous Build

Build	Time	Files changed	Build	Test	Who
86	Thu Sep 16 07:28:26 EDT 1999		Waiting		
85	Thu Sep 16 01:57:23 EDT 1999	1	0 errors	4 passed	avila
84	Wed Sep 15 23:58:57 EDT 1999	1	0 errors	4 passed	millerjv
83	Wed Sep 15 17:57:24 EDT 1999	3	0 errors	4 passed	avila martink
82	Wed Sep 15 17:42:08 EDT 1999	11	0 errors	4 passed	avila
81	Wed Sep 15 17:08:55 EDT 1999	4	0 errors	4 passed	millerjv
80	Wed Sep 15 15:09:54 EDT 1999	2	0 errors	4 passed	martink
79	Wed Sep 15 13:49:19 EDT 1999	2	0 errors	4 passed	avila
78	Wed Sep 15 12:31:13 EDT 1999	1	0 errors	4 passed	avila
77	Wed Sep 15 10:38:31 EDT 1999	1	0 errors	4 passed	will
76	Wed Sep 15 10:23:05 EDT 1999	4	0 errors	4 passed	martink will
75	Wed Sep 15 07:27:21 EDT 1999	3	0 errors	4 passed	lorensen
74	Tue Sep 14 20:36:01 EDT 1999	1	0 errors	4 passed	millerjv
73	Tue Sep 14 18:21:57 EDT 1999	14	0 errors	4 passed	lorensen

What's next?

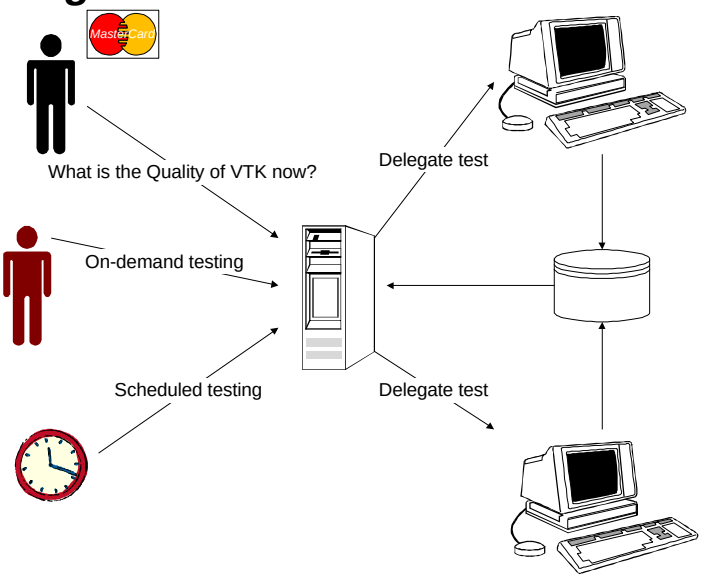
Current system

- *uses: csh, tcsh, tclsh, awk, grep, make, gmake, cvs, rcp, ssh, scp, ...*
- *is "vtk-centric"*
- *monolithic control structure best suited for daily testing of the complete system*
- *dashboard summarizes state as of 6 am*

New goal

- *Client-server model*
- *Dashboard summarized the most recent results*
- *Replace "rsh", "rcp", etc with http*

Testing Architecture



Visualization Toolkit Extreme Testing

A production release every day

http://vtk.scorec.rpi.edu/Nightly/MostRecentResults

Jim Miller
Computer Scientist

Bill Lorensen
Graphics Engineer

n

Visualization Toolkit Extreme Testing: A Production Release Every Day

William Lorensen
lorensen@crd.ge.com

James Miller
millerjv@crd.ge.com

GE Corporate R&D
KW/C223
1 Research Circle
Niskayuna, NY 12309

Introduction

Every software development project requires a process to move from requirements through design, implementation, testing and maintenance. Although tools and methodologies exist to carry out each of these steps, the degree to which these tools are applied is usually dictated by the size of the software product. The Capability Maturity Model [Paulk1994] developed by the Carnegie Melon Software Engineering Institute defines five levels of software process. For the most part, this discipline is applicable to large software development projects that have a separate software quality assurance and testing organization. Smaller projects that do not have a separate testing organization still need to apply sound software testing to achieve low defect software. In this paper we describe a practical, automatic software testing process for a medium sized software package. The software is the Visualization Toolkit [Schroeder1997] (http://www.kitware.com/vtk.html) an open source, C++ class library consisting of about 600 classes. Vtk's class library supports visualization, graphics and imaging. It runs on multiple vendor Unix platforms and Windows 95/98/NT.

In this paper, we describe the Visualization Toolkit (vtk), our motivation for testing and the automatic testing process we use to build, test and distribute vtk. We conclude with a discussion of the benefits of the testing that we provide directly to the development team.

The Visualization Toolkit

The Visualization Toolkit, vtk, is a set of C++ class libraries organized into kits that contain related functionality. These kits are used to build visualization applications for biomedical, scientific and information visualization, Figure 1.

Figure 1. Visualization of a combustor (left), computed tomography (center) and stress tensor hyper-streamlines(right).

Vtk is not an application but rather a toolkit that application developers use with other software to create end-user products. Broadly speaking, vtk presents the developer with sources, filters and graphics mappers that are connected into a visualization pipeline. Data is pulled through the filters by graphical demands that are initiated at the end of the process pipeline, Figure 2. A typical visualization requires from 3 to 10 vtk classes.

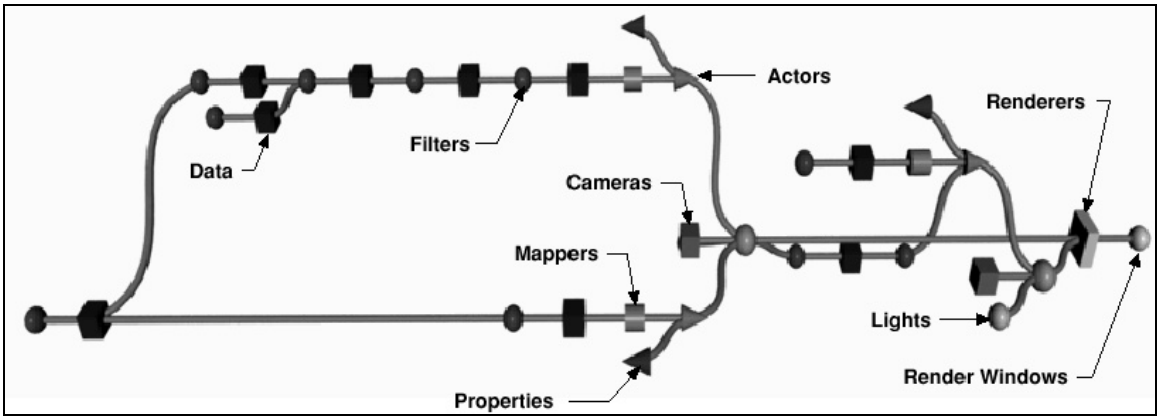


Figure 2. Visualization pipeline with filters, mappers and renderers.

Software Organization

The vtk source code is kept under the cvs revision control system (www.cyclic.com). The source code exists on an Internet site donated by a university user. The source tree is placed on this public site to maintain the Open Source goal of vtk and provide access to vtk developers. Read access is granted to all through an anonymous user id. Members of the development team can read and write to the repository. Some members of the vtk development team reside behind corporate firewalls and get access to the repository through a socks connection.

Currently five kits exist. Figure 3 shows the distribution of lines of code and number of classes for each kit.

1. *Common* contains classes that are used by each of the other kits. These include abstract filter classes, datasets, cells and utility classes. Vtk supports a variety of data organizations that range from unstructured data produced by finite element analysis to

highly structured data produced by medical imaging Computed Tomography and Magnetic Resonance Imaging scanners.

2. *Graphics* contains visualization classes. These classes process vtk's dataset classes and render the resulting polygonal output. The visualization filters in this kit extract surfaces of constant value, generate streamlines, warp surfaces and resample one data set with points from another. Many of the algorithms produce polygonal representations that can be rendered with conventional computer graphics software. Other filters retain the original form of the volumetric data and then apply volume rendering techniques directly to the volume data.
3. *Imaging* contains classes that process volumetric image data. The image data is structured data with implied topology. Because of the uniform storage mechanism, imaging classes can be streamed and threaded. Streaming is a technique that pulls pieces of image data through the pipeline.
4. *Patented* contains classes that implement techniques covered by US Patents. These classes can be used for educational purposes, but require a license for commercial use.
5. *Contrib* contains classes contributed by the vtk user community.

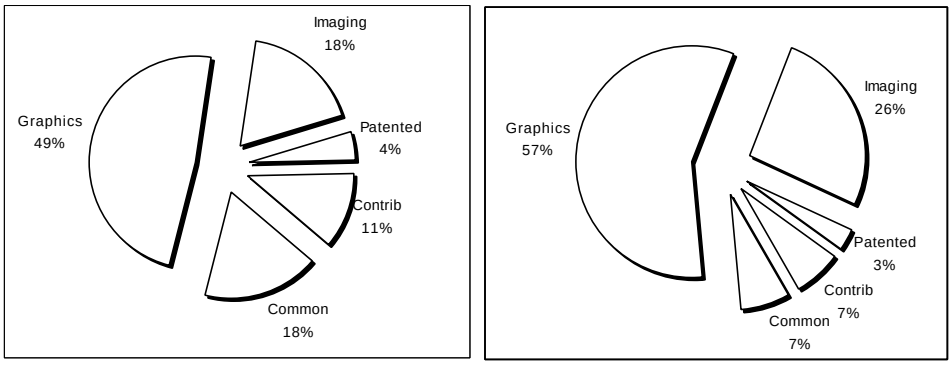


Figure 3. Distribution of Lines of Code (left) and Number of Classes (right) across the kits.

In addition to C++, vtk allows users to access the class libraries through tcl, java and python. This is accomplished through wrappers that are automatically generated as part of the build process. The access to vtk through a script level interface facilitates the testing process but also adds complexity to the testing since multiple language testing must be performed.

User Community

Vtk users range from students to researchers to application software engineers. The level of programming sophistication also covers a wide range. Vtk is written in C++ and developers can write applications in C++. This is the common approach if visualization tasks are embedded in the application. We also provide access to the individual classes in three other languages: java, tcl/tk and python. Distributed and Internet applications prefer java. Using java's JNI (Java Native Interface) a user can access vtk classes from java. To the developer, the classes appear as native java classes. However, the object libraries for

vtk must be available on the client computer. For applications that only use vtk classes, developers often prefer writing in tcl/tk or to a lesser extent python. Both of these languages are interpreted and lend themselves to rapid prototyping.

The vtk user community communicates through a mailing list. Users submit questions, bugs, benchmarks and results to this forum. The community is very supportive to new users and to developers that have OS or hardware platforms not available to the vtk development team.

Distribution Process

The vtk release includes beta, major and nightly releases. The beta and major release process follow common software distribution procedures. The nightly release provides power vtk users with access to the latest software release that has passed vtk's quality control process. The nightly release provides frequent bug fixes and enhancements to users without forcing them to wait for the next major release of the software. The nightly release allows vtk's user community to be tightly integrated to vtk's development cycle. A bug reported by a user in the morning can be fixed and rolled out to the user community that night. This saves our user community the time and effort involved in developing a "work-around" for a bug.

Providing a nightly release is not as simple as just creating a version of the source code every night. For users to have any confidence in a nightly drop, one also has to provide nightly testing. Thus, in vtk's software development process, the software is tested every night and the results summarized on the Internet before the nightly software release is distributed. (For the true vtk devotee, we also provide anonymous access to our source repository via the Internet. This code is made available regardless of any defects that exist.)

Motivation

All computer software contains bugs, inconsistencies and deficiencies. We call these problems defects. Our goal is to minimize the number of defects in the software that we use and release to our user community. To do this we have adopted the Six Sigma Process(http://www.sixsigmaqualtec.com/). Although we have always applied extensive testing to vtk since its inception in the summer of 1994, we started to apply more rigor to our testing in early 1998 as part of a Company-wide quality initiative. Several developers in our group selected six sigma projects that focused on vtk software quality issues. Some of these projects were specific to vtk while others applied to software in general. Each project successfully measured the existing state of the software, analyzed the defect patterns, corrected the defects and provided tools to control the defects. Before we started the initiative, we assumed vtk's software quality was high. After all, the system was nearly four years old and had been used on a number of internal software development projects. From the start, we had designed regression tests and did heavy testing as part of our beta release process. In spite of its widespread use and cream of the crop software development team, vtk remains a dynamic software class library. Dynamic software works properly only within a narrow window of the time it was last tested. Each change or addition to the software is an opportunity to introduce defects. From experience, the

vtk software was “relatively” defect free immediately after the testing for a major release, but then degraded as time passed until we started the testing for the next major release. We followed the normal separation of development and testing.

The reasons for the wide variation in software quality are complicated and hard to address. Every software developer would test the snippet of code they were adding to the system while they were developing it. This is standard practice when developing any software. For complex algorithms, developers often asked colleagues to inspect the code. But when you factor in the dynamic nature of a reasonably size software project, an innocent change or addition by software developer “A” can have a severe impact on the code developed by software developer “B”. It is these small and rather innocent changes that can have a huge impact on software quality and are the hardest to track. It is important to detect the changes and when they occur. A comprehensive and frequent testing system is required to identify exactly when the problems arise.

A second lesson involves cross platform compatibility. Vtk runs on multiple OS and hardware configurations. Once again, this was a goal from the beginning of the system. We found that when the software was tested on a particular platform, the software would work well on that platform. If we stopped testing the software on any one platform, the software’s quality would degrade on that platform. Here the causes were related to changes in the operating system, compilers, graphics libraries such as OpenGL, and changes to third party software such as Tcl/Tk. Since we work in a dynamic research environment, the only constant is that our computer systems are constantly changing. Therefore, to maintain a prescribed software quality, we need to test frequently to keep our software on top of our computer environment.

Finally, maintaining software quality can be a heavy burden. Software developers are pushed to their limits just to keep up with the demands of adding new features to the software. Manual checks of the implications of every change are costly. Furthermore, as a small software development group, we do not have the resources for everyone to compile and test their software on each computer architecture and OS release.

The vtk Software Testing Process

The goal of our software testing is to identify defects as soon as they are introduced into the software. If a developer does not catch the defect before committing the change, the nightly test suite will identify the defect so that it can be repaired before the next nightly build. Our software testing is a combination of regression and functional testing. The purpose of the regression tests is to assure that changes to the system do not change the operation of the software. The functional tests assure that each element of the system meets global requirements.

Regression Tests

Regression testing is the heart of the vtk software testing system and we have more than 500 regression tests. The purpose of the regression testing is to ensure that prior operation of the software has not been affected by changes to the software. Vtk's regression tests are image based and text based. Currently, we use tcl/tk scripts and C++ regression tests. The

tests are available to each developer and they can be run selectively or in their entirety. Developers are encouraged to run tests before checking changes into the revision control repository. Each evening, a control script choreographs the regression testing on 12 different hardware/software configurations. A variety of platforms are selected to use different C++ compilers and different toolkit/system configurations. Text tests generate text output that is compared with baseline results. Each image test creates an image and compares this image pixel-by-pixel with a “valid” baseline image. The resulting image difference is recorded. If a prescribed threshold is exceeded, the test fails. If a test fails, the system save three images: the generated image, the expected image and a difference image. A single line reports the test name, the elapsed and CPU time of the test and whether it passed of failed. If a test fails, the control script saves the generated image and creates an error image that shows the difference between the generated image and the valid image, Figure 4.

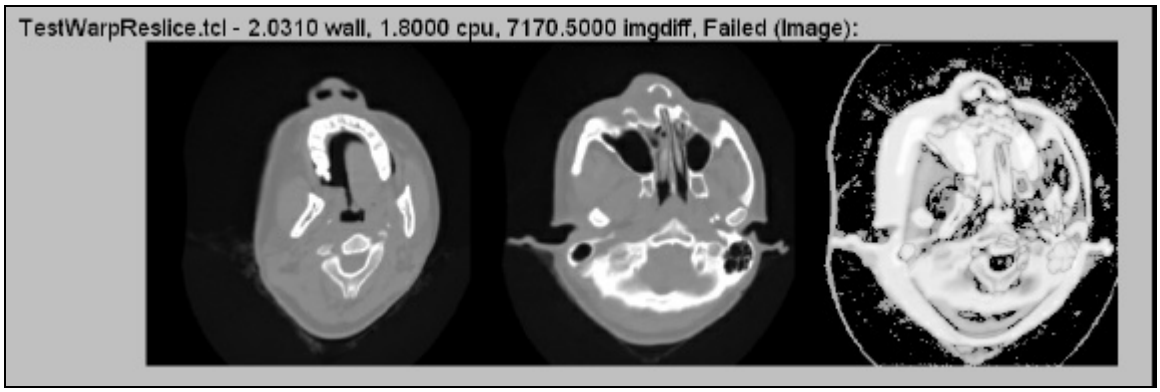


Figure 4. A failed regression test shows the produced image (left), expected image (center) and difference image (right).

Regression tests are also the basis for Dynamic Memory Analysis using Rational’s Purify (www.rational.com), Source Code Coverage and CPU utilization.

Dynamic Memory Analysis

The main purpose of the Purify runs is to catch memory leaks as soon as they are introduced. These runs also identify array bound access problems and memory uses after

```
VTK Dynamic Code Analysis for Sunday March 26, 2000 at 08:24 AM ================================
Total Memory Leaked: 0 bytes
Number of Leaks      : 0
Number of ABRs       : 0
Number of ABWs       : 0
Number of CORs       : 0
Number of FMMs       : 0
Number of FUMs       : 0
Number of FMRs       : 0
Number of FMWs       : 0
Number of MAFs       : 0
Number of UMCs       : 151
Number of UMRs       : 37
Purify Total Critical Defects: 0
```

Figure 5. Summary of the Purify log with defects identified.

the memory has been freed. Figure 5 shows a summary of defects gathered from the Purify log.

Timing

The CPU and wall time for each test is recorded. These times are compared to see if tests are running slower or faster than they have in the past. Figure 6 shows a timing result for an improved test. The plot shows that several days ago someone made a change that improved the times for this test.

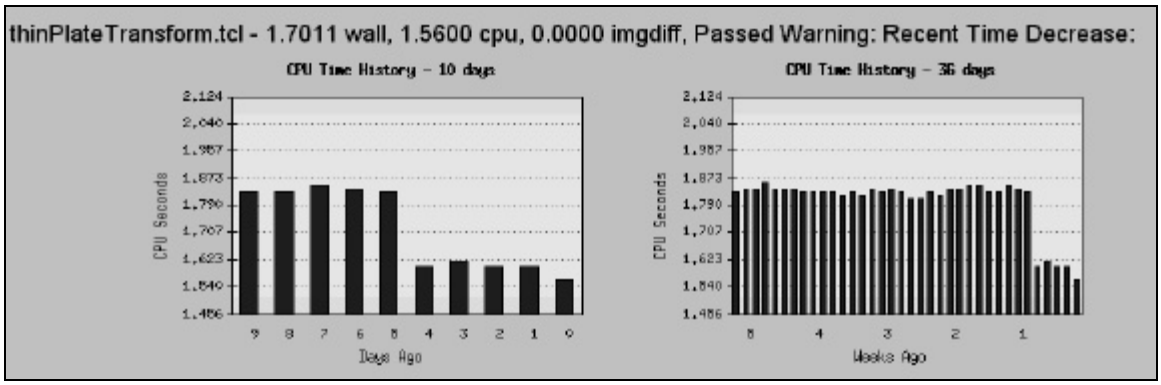


Figure 6. A timing result that shows a recent improvement in CPU time for a regression test.

Coverage

One of the builds is configured to run the regression tests with a compile flag that

vtkOutlineSource.cxx	96.97%	3485 clipRange = camera->GetClippingRange();
vtkPLOT3DReader.cxx	74.84%	3485 if (camera->GetParallelProjection())
vtkPicker.cxx	69.64%	###### {
vtkPiecewiseFunction.cxx	70.34%	###### tF = clipRange[0] - rayLength;
vtkPlaneSource.cxx	88.57%	###### tB = clipRange[1] - rayLength;
vtkPlanes.cxx	79.81%	###### for (i=0; i<3; i++) {
vtkPointDataToCellData.cxx	88.64%	###### p1World[i] = this->PickPosition[i] + tF*cameraDOP[i];
vtkPointLoad.cxx	86.15%	###### p2World[i] = this->PickPosition[i] + tB*cameraDOP[i];
vtkPointPicker.cxx	85.71%	###### }
vtkPointSetSource.cxx	57.14%	}
vtkPointSetToPointSetFilter.cxx	51.79%	3485 else {
vtkPointSource.cxx	95.00%	3485 tF = clipRange[0] / rayLength;
vtkPolyDataCollection.cxx	66.67%	3485 tB = clipRange[1] / rayLength;
vtkPolyDataConnectivityFilter.cxx	75.08%	for (i=0; i<3; i++)
		{
		10455 p1World[i] = cameraPos[i] + tF*ray[i];
		10455 p2World[i] = cameraPos[i] + tB*ray[i];
		10455 }
		}

Figure 7. An excerpt from the coverage pages shows the coverage in percent for each file. On the right, an annotated listing of a file that shows how many times each line was executed or #######'s if the line is not executed.

produces annotated listings of the source code. Each executable line is labeled with a number or hash marks, The number tells how many times the statement was executed. Hash marks indicate lines that are not executed at all.

Style

As with most diverse software teams, each developer has his or her own coding style. For the benefit of the team, however, most groups accept a coding style so that individuals can understand the code written by others. We use a commercial style checker called CodeCheck (http://www.abxsoft.com/codchk.htm) from Abraxas Software. CodeCheck is programmable so that a team can encode the agreed upon coding conventions. We process the output of the checker to summarize defects, Figure 8.

Complete Summary of VTK Style
Number of files 1183
Number of files with Defects 65

Style Rule	Opportunities	Defects
Class name prefix "vtk"	604	0
Class name alphanumerics	604	0
Instance variable begins with uppercase	2675	13
Instance variable name alphanumerics	2675	0
Constructor, destructor and assignment checks:		
Class has member function New()	547	20
Class has constructor	578	4
Class has default constructor	578	4
Class has copy constructor	578	5
Class has destructor	578	5
Class has operator=	578	4
Protected constructor/destructor	1722	3
Protected member data	2675	110
Member function begins with uppercase	12102	0
Member function name alphanumerics	12102	0
Macro name prefix "vtk" or "VTK"	1172	1
Dereferencing member data	39135	37
Dereferencing member function	28975	16
Braces around "if", "else", "for", ...	22754	20
Totals	130632	242

Size:

Executable lines:	93175	Low-level statements:	96169
Comment lines with text:	69574	Non-executable statements:	15656
Code lines with comments:	2618	High-level statements:	69217
Declaration lines:	27487	TOTAL STATEMENTS...	181042
Whitespace lines:	56697		
Preprocessor lines:	6412		
Suppressed lines:	1339	Macros defined:	1172
Punctuation lines:	59802	Functions defined:	14555
TOTAL LINES.....	314486		

Figure 8. Summary of Code Check output tailored for vtk.

Functional Tests

Black box tests are run to check for documentation defects and adherence to functional requirements. For example, vtk requires that each class in the system have a method PrintSelf that prints the values of each instance variable. This is important in debugging and in interpreted applications. Another test exercises each PrintSelf to ensure that the Print method does not fail. A third test checks the integrity of pipeline mechanism. These tests are generated automatically from the source files.

The Nightly Build

A nightly build beings at 8:00PM EST and completes around 7:00AM the following morning. The build proceeds as follows:

1. The cvs repository is queried to see which files have changed since the previous evening. Summaries of the changes are kept in a file that is accessible from the Dashboard.
2. A build is initiated on each platform and the results are collected in a log file for each configuration.
3. The regression suite is run. This includes image based C++ and tcl/tk tests as well as C++ and tcl/tk text tests. The results are accumulated in logs for each platform. Each test reports whether it passed or failed. One of the test platforms includes Purify applied to each regression test. The Purify log is saved in a file for later processing and reporting.
4. After all the builds and tests complete, the log files are scanned for defects. A defect in a build is either an error or warning. An html summary page is created that contains each defect with several lines before and after the defect to give the message context. There are some warnings that cannot be avoided. Patterns of these warnings are kept in an exception file so that they can be removed from the summary page. The number of errors and warnings are computed and stored in an html table.
5. The test logs are scanned for defects. A defect in a test occurs when a test fails. The number of tests that pass and fail are recorded and stored in an html table. For each failed image test, a jpeg is made of the generated image, the expected image and the difference of the two. These are stored in an html page that is linked to from the main dashboard page.
6. A timing analysis is performed on each platform. A summary of tests that are faster or slower is created and links are made to plots that show the variation over the last 10 days and last 12 weeks.
7. The Print test is run and the total numbers of defects are reported on the main dashboard. A link to the detailed results is created if there are any defects.
8. The coverage listings are parsed and summaries are prepared for the overall system, each kit and each source file. All results are available via html links.

9. The results of the style checker are summarized and links placed to the details for each kit.
10. The Purify log is parsed and the defect counts are summarized. A link to the full purify log is available from this summary page.
11. The top level Dashboard and all of its link pages are collected and transferred to the external web site. These pages are available to anyone on the Internet.
12. Html documentation is generated from the C++ header files. These pages are indexed and packaged into a compressed archive.
13. Downloadable files are created as a convenience to users:
    i. Linux RPM's that contain executables, source code and libraries.
    ii. A complete installation for Windows 98/NT/2000 as a single executable.
    iii. A complete source distribution for both Unix and Windows 98/NT/2000.

Figure 9 shows a recent Dashboard. Each number on the Dashboard is linked to a page containing more details.

VTK Dashboard for Tue Mar 14 20:01:40 EST 2000

There were 6 files changed since the last dashboard. Changes for the month.

Platform	Build VTK		Tcl Image Tests				Cxx Image Tests		Other Tcl Tests		Other Cxx Tests	
	Errors	Warnings	Pass	Fail	Fast	Slow	Pass	Fail	Pass	Fail	Pass	Fail
irix65	0	0	462	0	1	11	50	0	7	0	8	0
irix6n32	0	0	462	0	0	2	50	0	7	0	8	0
irix6x64	0	0	462	0	0	0	49	1	7	0	8	0
solaris	0	0	430	2	1	8	44	0	7	0	8	0
solarisCoverage	0	0	459	3	0	5	49	0	7	0	8	0
WinNT	0	0	454	8	0	0						
hp	1	0	459	3	0	1	49	1	7	0	8	0
linuxRH52	0	0	459	1	0	3	50	0	7	0	8	0
solaris26m6	0	0	455	7	2	8	49	1	7	0	8	0
solaris26	0	0	440	19	0	0						

Figure 9. A top level dashboard for one build. Colors highlight exceptions (http://vtk.scorec.rpi.edu/Nightly/MostRecentResults/Dashboard.html).

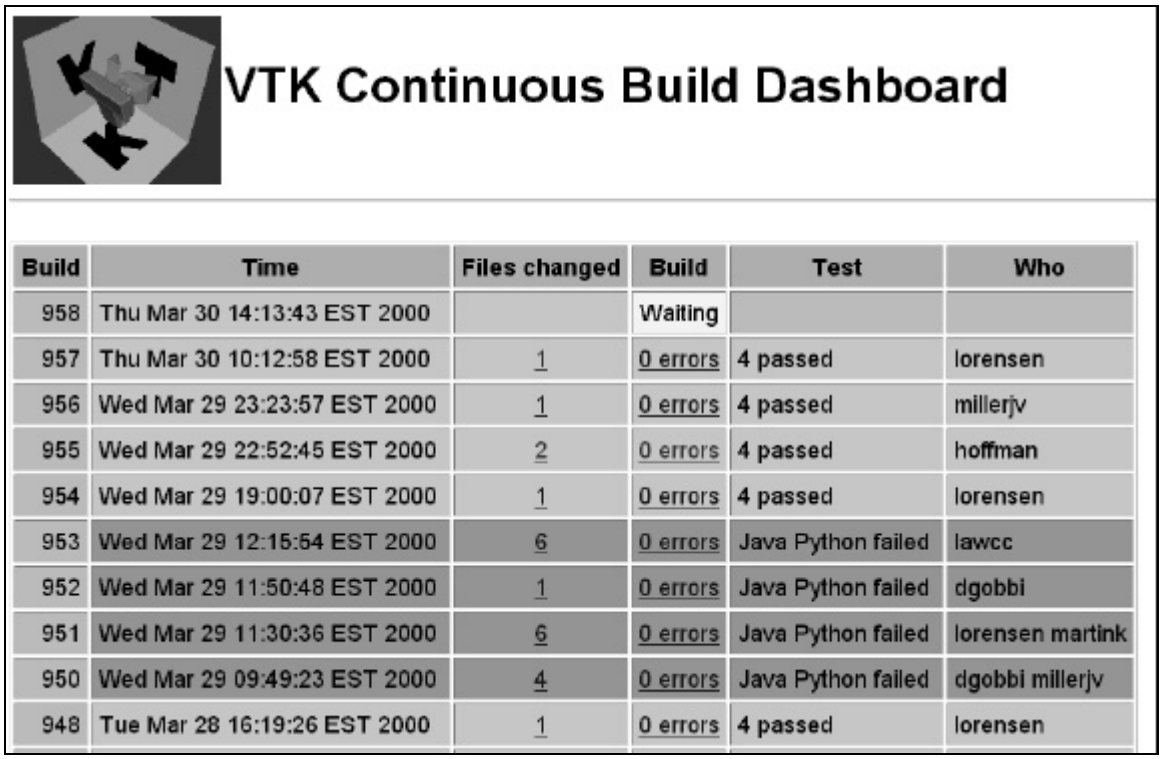
The Continuous Build

The nightly build and test process is critical to our development cycle. The team relies on the results to know the status of the system. After running the nightly testing for 12 months, we added a “continuous build” process to protect the nightly build from failing. Since the open source development team is distributed between several organizations, occasionally, a developer would check in a change that caused a compile to fail. This was usually because the developer did not have access to the all architectures that run the tests.

Periodically, the continuous build process checks the revision control repository for changes. If a change occurs, the process checks out the changes, compiles the new toolkit and runs four smoke tests to ensure a minimal conformance for the nightly tests. The smoke tests show that the toolkit will, at a minumm, run an application in each of the four supported lanuage bindings: C++, tcl, java and python. If the build fails or one of the tests fails, the process sends the following e-mail to the offending party:

> *Subject: vtk Build Problem: 03/29/00 11:50*
>
> *Please look at http://vtk.scorec.rpi.edu/Nightly/ContinuousResults/solaris/ContinuousResults.html. It appears that you may have checked in code that has broken the vtk build. We realize that another developer may have caused the build to break but you are one of the last people to check changes into the vtk repository.*

The developers agree to repair these defects before the nightly build. Since the continuous build was started in September 1999, only one nightly build process has failed completely and one has failed partially. In both cases the failure was caused because a developer checked in changes without watching the continuous build and did not check their e-mail. The *Continuous Build Dashboard*, Figure 10, shows the last 10 builds.



VTK Continuous Build Dashboard

Build	Time	Files changed	Build	Test	Who
958	Thu Mar 30 14:13:43 EST 2000		Waiting		
957	Thu Mar 30 10:12:58 EST 2000	1	0 errors	4 passed	lorensen
956	Wed Mar 29 23:23:57 EST 2000	1	0 errors	4 passed	millerjv
955	Wed Mar 29 22:52:45 EST 2000	2	0 errors	4 passed	hoffman
954	Wed Mar 29 19:00:07 EST 2000	1	0 errors	4 passed	lorensen
953	Wed Mar 29 12:15:54 EST 2000	6	0 errors	Java Python failed	lawcc
952	Wed Mar 29 11:50:48 EST 2000	1	0 errors	Java Python failed	dgobbi
951	Wed Mar 29 11:30:36 EST 2000	6	0 errors	Java Python failed	lorensen martink
950	Wed Mar 29 09:49:23 EST 2000	4	0 errors	Java Python failed	dgobbi millerjv
948	Tue Mar 28 16:19:26 EST 2000	1	0 errors	4 passed	lorensen

Figure 10. The Continuous build compiles the toolkit on one platform and runs a smoke test for each supported language binding (http://vtk.scorec.rpi.edu/Nightly/ContinuousResults/solaris/ContinuousResults.html).

Discussion

We have been running the nightly build since August 1998. Since that time we have gathered and retained almost 4 million test results. The general consensus among the development team is that we could not maintain the pace of enhancements and bug fixes without the process. Each morning, key team members review the Dashboard (http://vtk.scorec.rpi.edu/Nightly/MostRecentResults/Dashboard.html) to assess the state of the system. The review normally lasts 10 minutes. If all is well, the normal workday proceeds. If there are defects that have been introduced since the last build, assignments are made to repair the problems. For the most part, we are able to fix introduced problems within 24 hours. The system has had additional benefits:

1. We are able to make broad changes to the system with confidence. As long as the tests pass, we can proceed.
2. New team members can become productive immediately.
3. We can move to new third party software and upgrades with confidence.
4. Everyone knows the status of the system daily. We encourage uses to check the Dashboard before they download the new version.

Some lessons learned include:

1. The test system must be automatic. The main focus of our team is on development and not testing. The testing has to be minimally disruptive to the workday.
2. The reported defects need to be as close to zero as possible. If there are large numbers of defects being reported daily, they will be ignored. For instance, we went for years with many warnings in the compile logs. Since there were thousands of warnings, no one paid attention to the numbers. Now that we have driven them to zero, the defects are noticed and developers fix them as soon as they appear.
3. The entire team must buy into the process. There is no separation of testing and development. Developers are expected to test before they check in changes and to monitor the Dashboard.

Our system has a number of weaknesses as it stands today:

1. The Dashboard communicates the current state of the system very effectively. It does not show us how we are improving or where we stand with respect to our goals.
2. We continually struggle to improve our test coverage. With new classes, we find we can get near 100% coverage, but the older code is still under tested. Most of the bugs our users report are in untested code.
3. All of our test data is stored in flat files. This is convenient for daily reporting, but trend analysis is difficult.
4. In the current system, the build/test process and the Dashboard generation are tightly coupled. It is difficult to produce a Dashboard without running the tests.

Conclusions

We have implemented an automatic, nightly regression testing process that is suitable for small to medium sized software development teams. The process does not add a burden to the software developer but encourages the developer to perform and monitor testing on a regular basis. We have been using the nightly testing systems for almost two years. The testing system has the following benefits:

- Defects are detected within 24 hours of their introduction
- The readiness of the system for release is available on a daily basis
- The extensive test suite empowers developers to improve the system without fear of breaking it.

We are working on an improved testing framework. We have created abstractions of the testing process and have designed an object model that we can implement using a SQL database. The new system is more flexible, allowing us to experiment with new Dashboards and reporting mechanisms.

Acknowledgments

We appreciate the support of our GE management team, in particular Armin Pfoh, Pete Meenan and Kirby Vosburgh. Many vtk developers have contributed elements of the software testing. We especially thank our colleagues at GE CRD, Majeid Alyassin, Rick Avila, Dan Blezek, Rusty Blue, Jon Davis, Tim Kelliher, Annie Kelly, Steve Linthicum,

Tony Pan, Matt Turek and Boris Yamrom. From GE Medical Systems, we thank Dave Deaven, Darin Okerlund, Chris Slack and Bill Stovel.

Many external collaborators have contributed to vtk testing. From Kitware, we thank Lisa Avila, Charles Law, Ken Martin, Bill Hoffman and Will Schroeder. The Kitware folks contributed to the testing infrastructure, reducing defects and contribing tests. From Brigham and Womens Hospital, we thank Mike Halle and Simon Warfield for running the test suite remotely. From the Robarts Institute, we thank David Gobbi for adopting our process so quickly. From the National Center for Supercomputing Applications we thank Randy Heiland. From the Rutherford Appleton Laboratory, we thank John Biddiscombe. From the Eastman Dental Institute, we thank Timothy Hutton.

Bibliography

[Paulk1994] M.C. Paulk, C.V. Weber, B. Curtis, and M.B. Chrissis, eds.: *The Capability Maturity Model*. Addison-Wesley, Reading, Mass., 1994.

[Schroeder1997] Schroeder, W., K. Martin, and W. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Computer Graphics*, Prentice Hall, 1997.

QW2000 Paper 4W1

Ms. Pat Garverick
(Landmark Systems Corporation)

Testing the Performance Impact of a Web-based Application

BACK TO QW2000 PROGRAM

Key Points

- Describes the approach taken to develop a test plan to test the performance impact of running a new web-based application that has both server-side and browser-side components, including tools needed, and a report template to report the results.
- Describes the real life experience of putting the test plan into use testing the application on the NT 4.0 Server, using IIS 4.0 and IE5.
- Provides a list of suggested test cases and performance metrics to be included in the Report Template.

Presentation Abstract

This paper will describe an approach to developing a test plan to test the impact on the web server and on the client-side browser of adding a new, web-based application. The reality of carrying out the test with a real product under development will then be described. The test platform will include NT Server 4.0, Microsoft Internet Information Service (IIS) 4.0, and Microsoft Internet Explorer 5.0 at the minimum.

The goal includes writing a repeatable test plan that can be re-used as revisions are made to the software.

An additional goal is to gather a baseline measurement of the system running the new application. The baseline measurement will then be compared with future releases of the software to assess the performance impact of modifications to the application.

Included in the detail test plan will be a report template that will be used to communicate the findings each time the performance test is run.

Why a load simulation on the web server is not enough:

In this case, the application has a component that runs as part of the web server as well as Java function calls that are executed as the page is rendered on the browser. A load simulation that simulates "GET's" to the web server exercises the web server, including the application component that runs there. However, if the simulated load does not include the use of real browsers, then a browser is not rendering the web pages and the Java function calls are not getting executed. Therefore, a simulation of a browser load on the web server does not exercise all of this application

and does not give a true performance picture of the application.

However, such a simulated load does serve one purpose in our test scenario. We can use it to baseline the component that runs as part of the web server. This baseline can be used to compare with future releases of this component of the application.

Tools Needed:

The use of several types of tools will be necessary to generate the performance data. Each of the tools used will be described.

The Reality of Running the Test:

The real-world experience of actually making the performance runs to gather data will be discussed. Additional values of collecting performance data, including the uncovering of hidden errors in the software were found The final measurements used in the analysis will be presented in a report template.

About the Author

Pat Garverick is a member of the E-Business Quality Assurance Team at Landmark Systems Corporation (LDMK NASDAQ). Her main professional interest is the methodology of implementation of requirements-based testing in a software vendor environment. Determining the acceptable performance levels for web-based applications as well as developing realistic methods of testing the performance impact of new software being added to an existing system are important components of writing and testing against requirements.

BACK TO QW2000 PROGRAM

E-Business Quality Assurance

Testing the Performance Impact of a Web-Based Application Containing Both Server and Client Side Components

Pat Garverick
Landmark Systems Corporation®
12700 Sunrise Valley Drive
Reston, VA. 20191-5804
703-464-1544
dvpgg@landmark.com

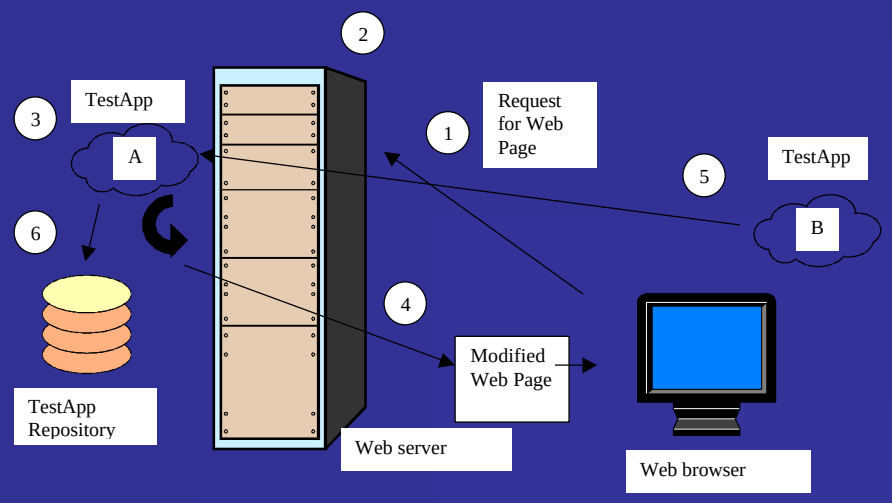
Project Goals

- Test the impact on the web server and on the client-side browser of adding a new, web-based application, TestApp
  - Describe TestApp
  - Describe Test Approach
  - Choose Tools and Metrics
  - Create Test Plan
  - Present Results in Report Template

TestApp Flow

- 1. Web browser requests web page
- 2. Web server assembles page components
- 3. TestApp modifies content of page
- 4. Web server sends modified web page to browser
- 5. Page renders on browser, executing TestApp function calls which communicate to TestApp running on web server
- 6. TestApp running on web server writes data to repository

TestApp Flow



Tools Needed

- During planning phase, needed tools are identified
  - Tool to direct the web browser to step through test web pages over and over
  - Tool to measure web server and operating system statistics on the web server machine
  - Tool to capture elapsed time between clicking on URL and completion of page rendering

Tools Selected

- ***Rational Visual Test***
- ***Microsoft Performance Monitor (PerfMon)***
- ***Microsoft Performance Data Log Service (PerfLog)***
- ***Microsoft Web Capacity Analysis Tool (WCAT)***
- ***Landmark Systems PerformanceWorks® SmartWatch***

Rational Visual Test

- Allows recording of mouse clicks as test web pages are reviewed
- Allows clicks on links, etc, to be written as function calls in the script
- Allows a delay to be used to give time for the pages to render
- Comes with ***Suite Manager*** which can repeat steps over and over

PerfMon

- Microsoft Performance Monitor
  - Provides stats for IIS web server and opsys
  - Performance Object(Instance)\Counter
  - Comes with Microsoft Windows NT 4.0
  - Described fully in Chapter 10 of Microsoft Windows NT Workstation Resource Kit book (Microsoft Press ISBN 1-47231-343-0. Doc can be downloaded from www.microsoft.com)

PerfLog

- Performance Data Log Server
  - Logs statistics from Perfmon
  - Comes with Microsoft Windows NT Workstation Resource Kit (PerfTool\MeasTool\Perflog.exe)

WCAT

- Microsoft Web Capacity Analysis Tool
  - Comes with Windows 2000 Resource Kit
  - Complete user guide included
  - Simplifies the collection of stats from PerfMon

PerformanceWorks® SmartWatch

- Landmark Systems Corporation®
- Captures end-to-end response time
- You define transaction
  - Starts with a window action, such as mouse click
  - Ends with a window action such as status bar indicating the page load is "Done"

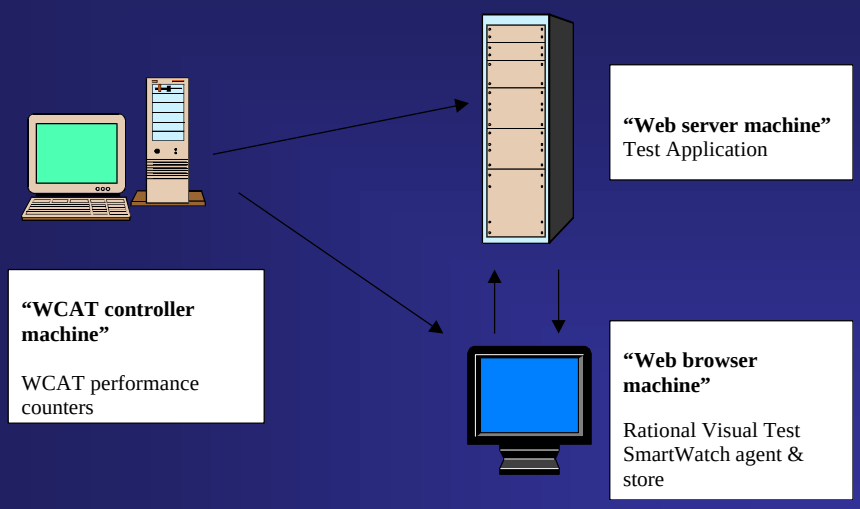
Creation of Test Plan

- Test Plan is major deliverable
  - Can be used repeatedly as test is rerun
  - Specifies exactly the setup and steps to follow
  - Gives consistent results

Contents of Test Plan

- Replay tool and WCAT configured to run repeatable load
- Gather stats without TestApp over multiple runs and average results
- Then, add TestApp - Full Function Mode
- Then, change to User Controlled Mode
- Report Template

Test Configuration



Report Template

- The Report Template is a major deliverable
  - Tools Used: description & release
  - Platform description:
    - machines
    - release of opsys, web server, browser
    - release of application being tested
  - Definitions of all measurements

Report Template cont.

- For each Test
  - Brief description of test
  - Repeatability of results
  - Raw figures without application in the mix
  - Raw figures with application in the mix
  - Any caveats or causes for misleading data

Anticipating Problems

- Part of project planning
  - Tools may not work as planned
  - Repeatability may be difficult to achieve
  - Application may be changing even as you test

Relevant Metrics

- Goal - identify small set that shows performance impact of TestApp
- Based on "what you want to measure"
- Rule of Thumb: Eyeball the detail data
- ***Average Task Response Time*** was the best overall metric

Project Results

- Report Template was created (see full paper for complete results in format suitable for template)
- Performance Test Plan was created

Results

- Project was successful:
  - Test plan built and carried out using planned approach
  - Reliable data was collected
  - Results delivered to development
  - Method in place to measure future improvements
- Consistency is the key

Testing the Performance Impact of a Web-Based Application Containing Both Server and Client Side Components

by Patricia Garverick
Landmark Systems Corporation

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Testing the Performance Impact of a Web-based Application Containing Both Server and Client Side Components

Executive Summary

The task we set out to do is to test the impact on the web server and on the client-side browser of adding a new, web-based application. For the purpose of discussion, the new application will be referred to as Test Application or TestApp.

The paper will describe the tools and performance metrics chosen. The Test Application will be described. The main steps of the test plan will be listed. The results of the actual project will be presented in a format that can be used as the template for reporting all future results of running the same test plan. The results include a brief analysis of each test and an overall summary of the real experience of running the tests. Finally, two appendixes are included. The first appendix supplies a list of resources that are helpful for setting up a performance test for a web-based application. The second appendix provides the definition of a set of performance metrics that are candidates for use in such a performance test.

Description of the Test Application

The new application to be tested, in this case, has a component that runs as part of the web server as well as executable code that runs as part of the browser, as shown in Figure 1, below.

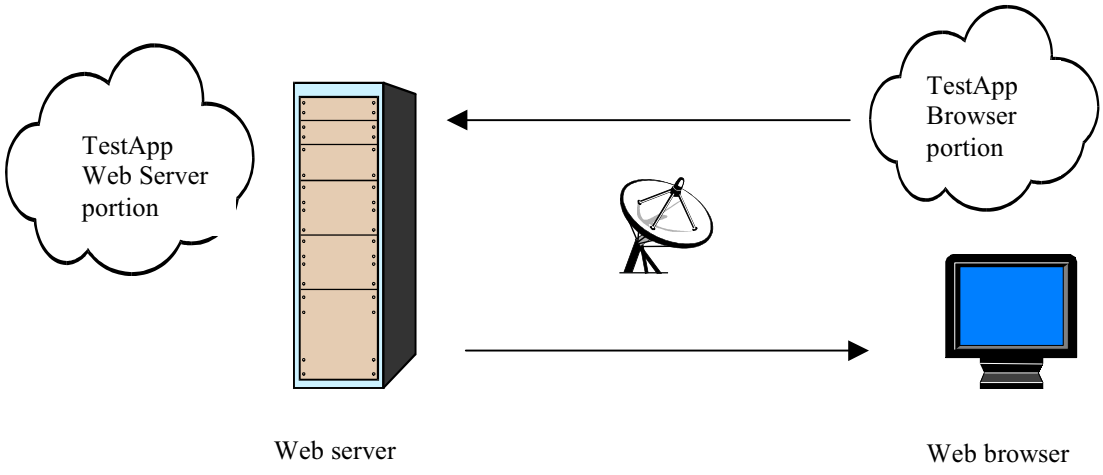


Figure 1: Web-based application running on web server and browser

© 2000 by Landmark Systems Corporation
and Patricia Garverick

The Components that Run on the Web Server Side

The application contains a module that runs as part of the web server. On Microsoft's IIS, the module is an ISAPI filter.

Processing depends on the application configuration switches. The application can run in two modes – the full function mode or the user-controlled mode. In the full function mode, additional web content is added to every page served up. In user-controlled mode, the pre-modified web pages are stored on the server.

The Components that Run on the Browser Side

The browser receives the page with the modified content from the server. This modified content consists of applet definition statements and function calls, which will execute as the page is rendered on the browser and as the user clicks on buttons and urls.

Application Flow

As shown in Figure 2, below, the sequence is:

1. Web browser requests web page.
2. Web server assembles page components.
3. Test Application modifies content of page.
4. Web server sends modified web page to browser.
5. Page renders on browser, executing Test Application function calls which communicate to Test Application running on web server.
6. Test Application running on web server writes TestApp data to repository.

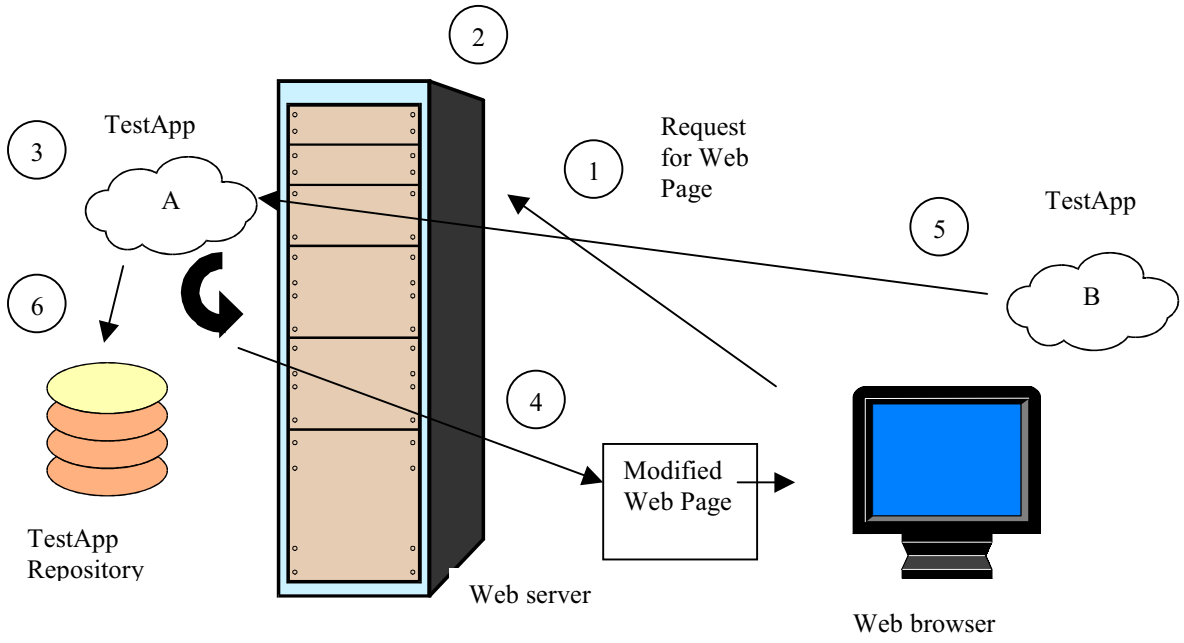


Figure 2. Test Application Flow

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Why a load simulation on the web server is not enough

A load simulation which simulates "GET's" to the web server exercises the web server, including the application component which runs there. However, if the simulated load does not include the use of real browsers, then the pages are not getting rendered by a browser and the Java function calls are not getting executed. Therefore, a simulation of a browser load on the web server does not exercise all of the application and does not give a true performance picture of the application.

Tools Needed

During the planning phase of a test of software performance, tools are identified that will be needed to generate the performance data. For this project, the following types of tools will be needed:

- A tool to simply step through a set of pages over and over. This will cause the actual browser to ask for and render the pages.

- A tool to measure web server statistics and operating statistics on the web server machine.

- A tool to capture the elapsed time between the click on a URL on the browser and the completion of the rendering of the requested page.

Tools Selected

- ***Rational Visual Test*** will be used to step the browser through a set of web pages. This set of steps will then be replayed over and over during the actual test run.

- ***Microsoft "Performance Monitor"*** tool will be used to collect the web statistics for the IIS web server. Performance Monitor is a Microsoft Windows NT 4.0 tool. (Start > Programs > Administrative Tools > Performance Monitor. The tool is described fully in Chapter 10 of the Microsoft Windows NT Workstation Resource Kit book (Microsoft Press ISBN 1-57231-343-9). The resource kit documentation may be downloadable from www.microsoft.com.

- ***Performance Data Log Service (PerfLog)*** will be used to log the performance objects and counters that Performance Monitor reports on. PerfLog is available on the Microsoft Windows NT Workstation Resource Kit 4.0 CD (PerfTool\MeasTool\Perflog.exe.)

- ***Microsoft Web Capacity Analysis Tool (WCAT)*** will be used as a tool to simplify using "Performance Monitor" to collect and summarize statistics. WCAT will also be used to directly run a load on the web server to baseline the application component

© 2000 by Landmark Systems Corporation
and Patricia Garverick

that runs as part of the web server. The tool is included on the Microsoft Windows 2000 Resource Kit CD. A complete user guide is included on the Resource Kit CD.

- ***Landmark Systems Corporation SmartWatch*** will be used to measure browser performance.

Creation of Test Plan

A major output of the project is to create the Performance Test Plan using the standard format for test plans at Landmark Systems.

a) The replay tool and WCAT will be configured to run a load that yields repeatable performance results prior to adding the new application to the mix. The test plan will specify that this step be run three times and the performance measurements averaged.
b) The test plan will then specify to add the new application.
c) The next step in the test plan will be to rerun the initial load three times and average the results.
d) The test plan's next step will be to have WCAT run a built-in test three times to baseline the web server running with the application component that runs on the web server. (This is only done after the application is added, not before. This is because the numbers are meaningless to the application as a whole since the entire application is not being exercised. That is, the numbers cannot be used to measure the performance impact of the application since a large part of the application is disabled when running this way.)

The final component of the test plan will be the report template. The report will contain:

- A description of the tools used, including release information
- A description of the platforms the test is run on, including release of web server, release of browser, release of op sys.
- A description of the software under test, including release information.
- For each test:
  - A brief description of the test
  - A description of how consistently repeatable the results were
  - Raw figures without the application in the mix
  - Raw figures with the application in the mix
  - Any caveats or characteristics that might be giving misleading data
- Definitions of all measurements.

Anticipated Problems

When planning the approach for testing the performance of a new application, assumptions are made based on tool descriptions and input from the authors of the Test Application. An important part of planning is to identify the areas of risk. Here are the areas identified for this project:

- ***The tools may not work as planned.*** The reality of using the selected tools may not be as expected. The tools may have to be changed or used in a different manner than expected.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

- ***Repeatability may be difficult to achieve.*** As we learn more from actually running the steps, we may learn additional steps that need to be followed to ensure repeatability.
- ***The application will be changing.*** Applications change as new requirements are discovered. Comparing future versions of the product to the initial baseline will always require awareness of major new functions that may have been added to the software. Additional test steps may need to be added or eliminated as the application changes.

Determination of Relevant Metrics

The goal is to identify a small set of metrics that distinctly showed the performance impact of the Test Application.

The selection of performance measurements chosen for a project depends on what is being measured. If you want to measure the performance of one web server versus another, then you will use a different set of measures. For example, if you want to measure the resource usage of the IIS web server, you would probably use these and other metrics particular to the web server process:

```
Syntax:  Microsoft Windows Performance Object(instance)\counter

Process(inetinfo)\% Processor time
Process(inetinfo)\% Privileged time
Process(inetinfo)\% User Time
```

The approach taken for this project was to go ahead and collect data for all the performance metrics that seemed like likely candidates for giving us good information. Then, when analyzing the results, all metrics that were of no use were thrown out.

Rule of thumb:

Always eyeball a small subset of detail observations before applying summarizing formulas such as Average.

- Watch for data out of range that needs to be thrown out
- Learn about errors in the software under test
- Look for measurements that measure what you are trying to measure
- Watch for setup problems that skew the data (ex. Failure to reboot before each test)

Average Task Response Time

The overall best metric was the "average task response time" figure that was calculated from the SmartWatch transaction response times. This measurement clearly showed the overall impact, from the user point of view, of the software change. The actual thing being measured was the elapsed time from the moment the mouse click occurred on a URL to the point where the browser finished rendering the requested page. If we were only going to use one number, this would be it.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Metrics Thrown Out:

```
Network Interface/Bytes Received Per Second
Network Interface/Packets Received Per Second
Web Service(_Total)\Bytes Total/sec
```

These really measure the rate that messages hit the network interface. As you will see in the test results, we purposefully showed down the rate of replay of the test suite. Therefore, all these two figures are telling us is that the rate is slower in the runs where we slowed down the rate. In our test scenario, these measurements don't measure what we want to measure, so we throw them out.
In the next round of testing, if we can run the suites at the same rate for all the runs, then these would become valid measures.

Summary of Results

The report template was created. The goal of creating a report template that can be used for reporting future runs of the test was achieved. At the end of this paper, the "Performance Test Results" document is presented in a format that can be used as a template. The document presents the actual results of this project.

The Performance Test Plan created. The Performance Test Plan that was created as part of the project is not included due to space constraints.

The project was successful. The test plan was built and carried out using the planned approach and reliable data was collected. The results of the test were formatted in a manner appropriate for delivery to the departments that can use the data, such as the development organization. As development modifies the software to incorporate performance improvements, we will have a method to measure the improvements made.

Consistency is the key. Consistency in the test cases run and in the reporting method used is the key to having results that can be used as the basis of making measurable improvements to the software over time.

Appendix A: Related Resources

Microsoft Knowledge Base Articles:
Q164018 – "Controlling Remote Performance Monitor Access to Windows NT Servers"

Q146005 – "Optimizing Windows NT for Performance". Applies to Microsoft Windows 2000 as well. Identifies specific objects and counters to look at for specific suspected problems.

Microsoft MSDN Online Library:
"Developing Web Applications with Visual Basic and Visual InterDev", Valor Whisler and Jim Soiland, August 20, 1997 [Includes overview of basic web architecture.]

"Should Your Next UI Be Written in Dynamic HTML?", by Dr. GUI, Microsoft MSDN Online Library, November 3, 1997: http://msdn.microsoft.com/workshop/Author/om/dom.asp.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Performance Test Results

<<Format suitable for use as template>>

© 2000 by Landmark Systems Corporation
and Patricia Garverick

1.0 Test Description

The purpose of the performance test is to identify the impact on end-user performance of adding the test application to the environment.

- The replay tool and WCAT will be configured to run a load that yields repeatable performance results prior to adding the new application to the mix. The test plan will specify that this step be run three times and the performance measurements averaged.
- The test plan will then specify to add the new application.
- The next step in the test plan will be to rerun the initial load three times and average the results.
- The test plan will then specify to configure the application to run in the alternate mode.
- The initial load will be rerun 3 times and results averaged.
- The test plan's next step will be to have WCAT run a built-in test three times to baseline the web server running with the application component that runs as part of the web server.

2.0 Description of the Software Being Tested

The new application to be tested, in this case, has a component that runs as part of the web server as well as executable code that runs as part of the browser. The application running as part of the web server reads the outbound data stream and adds content specified by the user as part of the product configuration. The added content then executes when the page is rendered on the browser.

Please refer to the main paper for details.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

3.0 Tools Used

Tool	Purpose
Rational Visual Test	Record/replay tool used to drive the load by stepping through the test web site using a web browser.
Suite Manager	Part of Visual Test. Drives load by repeating the test to step through the web site the specified number of times.
Microsoft Web Capacity Analysis Tool (WCAT) (included on Microsoft Windows Resource Kit)	Fetches PerfLog statistics. Also used to directly load web server as part of the "baseline" runs.
Landmark Systems "SmartWatch"	Capture end user response time which is the time between a mouse click on a URL and the time the page rendering is complete.
Microsoft Performance Monitor (PerfLog) Included on Microsoft Windows NT Workstation Resource Kit 4.0.	Log performance objects and counters of web server machine.

4.0 Testing Software

Visual Test Suite Manager Test Suite Used	Tour Garden Site Suite, Tour I_Garden Site Suite
Repeat Count Used on "Tour Site" folder	180
Visual Test Suite Manager Version	6.01
Rational Visual Test Version	6.01
WCAT Version	4.25

5.0 Test Platform

Application Component, Version, Release, Modification Level	Test Application Pre-Release Version 3
Application Component, Version, Release, Modification Level	Logging Component Version 2.3.1
Web Server Machine and opsys	Qa95nt NT Server 4.0 service pack 4
Web Server Software	IIS 4.0
Web Browser Machine and opsys	Qapine NT Server 4.0 service pack 6a
Web Browser Software	Internet Explorer 4.0 (4.72.3110.8 -128 bit encrypt - SP1a)

© 2000 by Landmark Systems Corporation
and Patricia Garverick

6.0 Performance Test Configuration

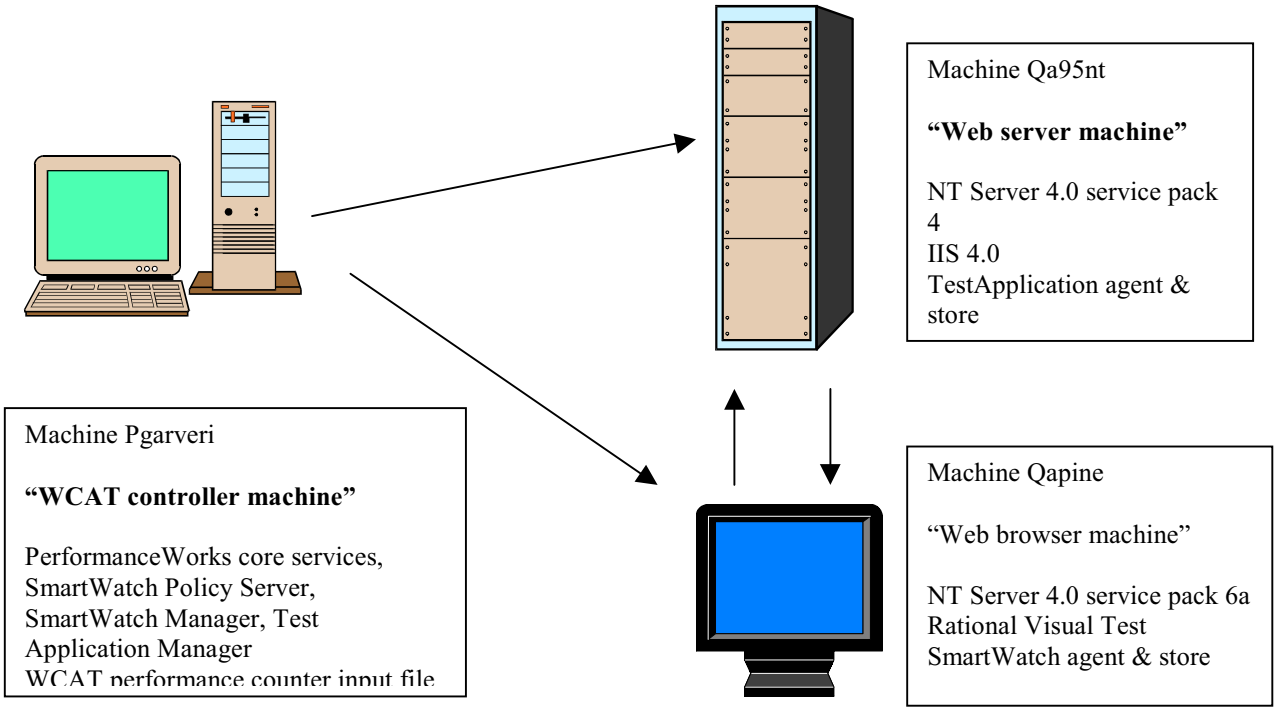


Figure 1: Machine Configuration

7.0 Results

Each test case was run 3 times and then the results were averaged to come up with one set of observations per test case. Repeatability of the results was within 10 percent for the three runs for the metrics that were kept.

The results for each test case follow:

7.1 Test Case 1.1.1 - Basic Measurements without the Test Application

This test case simply captured the performance data of running the test suite without the test application added to the mix.

As you can see from Appendix B through D, there were no suite errors during the replay. The suite looped through the set of web pages 180 times. The average time to complete the suite was 1 hour 4 minutes and 34 seconds.

The average task response time for a one-hour period was .826 seconds. During that one-hour period, the average transaction count was 1,187.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

The IIS Global Cache Misses averaged 30, the Memory/Page Faults/sec averaged 3.14, and the Network Segment/%Network utilization averaged 2.095. The Processor/% Processor Time averaged 99.997.

7.2 Test Case 1.2.1 - Basic Measurements with the Test Application – Full Function Mode

This test case captured the performance data of running the test suite with the test application installed and set to run in the "full function" mode.

The data in the Appendix shows that suite errors occurred on all three runs. In each case, the suite stalled, triggering the suite errors, because a web page did not render in time for the next mouse click. In the three cases, the error occurred after the first hour. Only the first hour of data was used in the data analysis.

The suite looped through the set of web pages 180 times. The average time to complete the suite was 1 hour 15 minutes and 15 seconds.

The length of time to run the suite was longer during this set of runs because of two things:

- The application being tested lengthened the time it took between a page request and the completion of the page rendering
- The suite had to be slowed down by inserting pauses in order to allow time for the page rendering to complete.

The average task response time was 1.535 seconds. This time was not affected by any pauses put in the test suite. During that one-hour period, the average transaction count was 729.

Adding the test application, running in full function mode, caused the amount of transactions processed during the one hour period to decrease by 39%. The average transaction response time increased by 86 percent.

The IIS Global Cache Misses averaged 145.33, a 350 percent increase over running without the test application. This implies that the slower response time was not only attributable to longer page rendering times but was also due to more time being spent waiting for page files to be fetched over the network.

The Memory/Page Faults/sec averaged 10.55, and the Network Segment/%Network utilization averaged 6.199. The Processor/% Processor Time averaged 99.995.

7.3 Test Case 1.2.2 - Basic Measurements with the Test Application – User Controlled Mode

This test case captured the performance data of running the test suite with the test application installed and set to run in the "user controlled" mode.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

The data in the Appendix shows that no suite errors occurred on any of the runs. However, the length of pauses inserted in the suite had been increased. This was because the suite consistently stalled before the one-hour mark until the pauses were increased.

The suite looped through the set of web pages 180 times on two of the runs. The suite was only allowed to run for one hour on one run. On that run, only 104 iterations were processed before the suite was manually stopped. The run that was manually stopped was excluded from the averages.

The average time to complete the suite was 1 hour 55 minutes and 30 seconds.

The average task response time was 1.801 seconds. This time was not affected by any pauses put in the test suite. During that one-hour period, the average transaction count was 380.

Adding the test application, running in user controlled mode, caused the amount of transactions processed during the one hour period to decrease by 68%. The average transaction response time increased by 118 percent.

The IIS Global Cache Misses averaged 173.33, an increase of 478 percent over running without the test application. The Memory/Page Faults/sec averaged 6.9, and the Network Segment/%Network utilization averaged 20.18. The Processor/% Processor Time averaged 67.36, a decrease of 33 percent over running without the application. During the one hour, more time was spent on waiting for data across the network and paging and less time was spent actually processing.

7.4 Test Case 1.3.1 - Baseline the ISAPI Filter – Full Function Mode

There is no "before" data to compare the runs with where we are collecting baseline data to describe performance when the web server runs with the ISAPI filter that is part of the application under test.

Running with the filter configured for "full function mode", the average number of web pages requested in 30 minutes was 3,444. The total responses per second was 1.92. The bytes of data read per second was 31,879. The total bytes per second was 32,316.

The IIS Global Cache Misses averaged 3115.67, the Memory/Page Faults/sec averaged 20.89, and the Network Segment/%Network utilization averaged 28.74. The Processor/% Processor Time averaged 9.6.

This data will be used to compare with subsequent releases of the software being tested to indicate if the performance is improving as changes are made.

7.5 Test Case 1.3.2 - Baseline the ISAPI Filter – User Controlled Mode

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Running with the filter configured for "user controlled mode" the average number of web pages requested in 30 minutes was 3,491. The total responses per second was 1.94. The bytes of data read per second was 32,170. The total bytes per second was 32,612.

The IIS Global Cache Misses averaged 3304.67, the Memory/Page Faults/sec averaged 17.36, and the Network Segment/%Network utilization averaged 6.70. The Processor/% Processor Time averaged 2.11.

This data will be used to compare with subsequent releases of the software being tested to indicate if the performance is improving as changes are made.

8.0 Discussion of the test Process

The results were both expected and unexpected. Actually making a run and collecting the data was more difficult than anticipated.

Unexpected factors

The Test Application had some problems. The good news is that some of the problems were actually uncovered because of the performance testing. That is, some of the timings were large and subsequent investigation showed errors occurring that had previously gone undetected.

- ***The record and replay tool stalled when the performance was dragging.*** During the runs to collect the timings, when the Test Application was put in the mix and performance suffered, the replay tool stalled because a web page wasn't rendered in time for the next mouse click. This wasn't the fault of the replay tool. Time delays had to be added to allow extra time for the web pages to render. Future runs of the Performance Test plan will be made using the longest pauses necessary, throughout the entire test so that all results are consistent.

- ***The early version of the Test Application interfered with some types of web page components.*** So, the replay suite had to be modified by commenting out references to pages that had those types of components. These portions will need to be restored before testing with future versions of the product.

- ***There was competition for resources.*** Three test machines were required for this test project. Other projects were requesting the use of one of the machines. In subsequent drops the schedule will need to set aside the use of 3 machines for one week.

- ***The web server machine was on the small side.*** I had problems with the machine running out of virtual memory, which caused unplanned reboots. To prevent this from affecting the performance data, the test plan will be changed to direct that the web server machine be rebooted before each run.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Appendix A: Metric Definitions

The definitions below, with the exception of "TimeSlice" are copied directly from the PerfLog Explain window for the selected Microsoft Windows Objects and Counters:

Internet Information Services Global\Cache Misses
Cache Misses is the total number of times a file open, directory listing or service-specific object request was not found in the cache.

Memory\Page Faults/sec
Page Faults/sec is a count of the Page Faults in the processor. A page fault occurs when a process refers to a virtual memory page that is not in its Working Set in main memory. A Page Fault will not cause the page to be fetched from disk if that page is on the standby list, and hence already in main memory, or if it is in use by another process with whom the page is shared.

Network Interface(2)\Bytes Received/sec
Bytes Received/sec is the rate that bytes are received on the interface, including framing characters.

Network Interface(2)\Packets Received/sec
Packets Received/sec is the rate that packets are received on the network interface.

Network Segment(\Device\bh_OcTk161)\% Network utilization
Percentage of network bandwidth in use on this network segment.

Paging File(\??\D:\pagefile.sys)\% Usage
The amount of the Page File instance in use in percent. See also Process:Page File Bytes.

Process(inetinfo)\% Processor Time
Processor Time is the percentage of elapsed time that all of the threads of this process used the processor to execute instructions. An instruction is the basic unit of execution in a computer, a thread is the object that executes instructions, and a process is the object created when a program is run. Code executed to handle certain hardware interrupts or trap conditions may be counted for this process.

Process(inetinfo)\% Privileged Time
Privileged Time is the percentage of elapsed time that this process's threads have spent executing code in Privileged Mode. When a Windows NT system service is called, the service will often run in Privileged Mode to gain access to system-private data. Such data is protected from access by threads executing in User Mode. Calls to the system may be explicit, or they may be implicit such as when a page fault or an interrupt occurs. Unlike some early operating systems, Windows NT uses process boundaries for subsystem protection in addition to the traditional protection of User and Privileged modes. These subsystem processes provide additional protection. Therefore, some work done by Windows NT on behalf of your application may appear in other subsystem processes in addition to the Privileged Time in your process.

Process(inetinfo)\% User Time
User Time is the percentage of elapsed time that this process's threads have spent executing code in User Mode. Applications execute in User Mode, as do subsystems like the window manager

© 2000 by Landmark Systems Corporation
and Patricia Garverick

and the graphics engine. Code executing in User Mode cannot damage the integrity of the Windows NT Executive, Kernel, and device drivers. Unlike some early operating systems, Windows NT uses process boundaries for subsystem protection in addition to the traditional protection of User and Privileged modes. These subsystem processes provide additional protection. Therefore, some work done by Windows NT on behalf of your application may appear in other subsystem processes in addition to the Privileged Time in your process.

Process(inetinfo)\Thread Count
The number of threads currently active in this process. An instruction is the basic unit of execution in a processor, and a thread is the object that executes instructions. Every running process has at least one thread.

Processor(0)\DPC Rate
DPC Rate is the average rate DPC objects are queued to this processor's DPC queue per clock tick.

Processor(0)\% DPC Time
DPC Time is the percentage of elapsed time that the Processor spent in Deferred Procedure Calls. When a hardware device interrupts the Processor, the Interrupt Handler may elect to execute the majority of its work in a DPC. DPC's run at lower priority than Interrupts, and so permit Interrupts to occur while DPC's being execute. Deferred Procedure Calls are executed in Privileged Mode, so this is a component of Processor: % Privileged Time. This counter can help determine the source of excessive time being spent in Privileged Mode.

Processor(0)\% Interrupt Time
% Interrupt Time is the percentage of elapsed time that the Processor spent handling hardware Interrupts. When a hardware device interrupts the Processor, the Interrupt Handler will execute to handle the condition, usually by signaling I/O completion and possibly issuing another pending I/O request. Some of this work may be done in a Deferred Procedure Call (see % DPC Time.) However, time spent in DPC's is not counted as time in Interrupts. Interrupts are executed in Privileged Mode, so this is a component of Processor: % Privileged Time. This counter can help determine the source of excessive time being spent in Privileged Mode.

Processor(0)\% Privileged Time
Privileged Time is the percentage of processor time spent in Privileged Mode in non-Idle threads. The Windows NT service layer, the Executive routines, and the Windows NT Kernel execute in Privileged Mode. Device drivers for most devices other than graphics adapters and printers also execute in Privileged Mode. Unlike some early operating systems, Windows NT uses process boundaries for subsystem protection in addition to the traditional protection of User and Privileged modes. These subsystem processes provide additional protection. Therefore, some work done by Windows NT on behalf of your application may appear in other subsystem processes in addition to the Privileged Time in your process.

Processor(0)\% Processor Time
Processor Time is expressed as a percentage of the elapsed time that a processor is busy executing a non-Idle thread. It can be viewed as the fraction of the time spent doing useful work. Each processor is assigned an Idle thread in the Idle process which consumes those unproductive processor cycles not used by any other threads.

Processor(0)\% User Time

© 2000 by Landmark Systems Corporation
and Patricia Garverick

User Time is the percentage of processor time spent in User Mode in non-Idle threads. All application code and subsystem code execute in User Mode. The graphics engine, graphics device drivers, printer device drivers, and the window manager also execute in User Mode. Code executing in User Mode cannot damage the integrity of the Windows NT Executive, Kernel, and device drivers. Unlike some early operating systems, Windows NT uses process boundaries for subsystem protection in addition to the traditional protection of User and Privileged modes. These subsystem processes provide additional protection. Therefore, some work done by Windows NT on behalf of your application may appear in other subsystem processes in addition to the Privileged Time in your process.

Server\Bytes Total/sec
The number of bytes the server has sent to and received from the network. This value provides an overall indication of how busy the server is.

System\% Total Processor Time
The % Total Processor Time is the average percentage of time that all the processors on the system are busy executing non-idle threads. On a multi-processor system, if all processors are always busy this is 100%, if all processors are 50% busy this is 50% and if 1/4th of the processors are 100% busy this is 25%. It can be viewed as the fraction of the time spent doing useful work. Each processor is assigned an Idle thread in the Idle process which consumes those unproductive processor cycles not used by any other threads.

System\Context Switches/sec
Context Switches/sec is the rate of switches from one thread to another. Thread switches can occur either inside of a single process or across processes. A thread switch may be caused either by one thread asking another for information, or by a thread being preempted by another, higher priority thread becoming ready to run. Unlike some early operating systems, Windows NT uses process boundaries for subsystem protection in addition to the traditional protection of User and Privileged modes. These subsystem processes provide additional protection. Therefore, some work done by Windows NT on behalf of an application may appear in other subsystem processes in addition to the Privileged Time in the application. Switching to the subsystem process causes one Context Switch in the application thread. Switching back causes another Context Switch in the subsystem thread.

System\System Calls/sec
Systems Calls/sec is the frequency of calls to Windows NT system service routines. These routines perform all of the basic scheduling and synchronization of activities on the computer, and provide access to non-graphical devices, memory management, and name space management.

TimeSlice
The amount of time of this observation.

Web Service(_Total)\Bytes Total/sec
Bytes Total/sec is the sum of Bytes Sent/sec and Bytes Received/sec. This is the total rate of bytes transferred by the Web service.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Appendix B: Average Task Response Time Data – Raw Figures

Test Run	Config	Suite Elapsed Time	Suite Errors	1 Hour Trans Count	1 Hour Avg Task Resp	Iterations	Notes
1.1.1 Run 1	Without Test App	1 hr 4 min 27 sec	0	1,193	0.828	180	MySleep Time=0.1
1.1.1 Run 2	Without Test App	1 hr 4 min 31 sec	0	1,182	0.824	180	
1.1.1 Run 3	Test App Full Function	1 hr 4 min 43 sec	0	1,185	0.827	180	
AVERAGE		**1 hr 4 min 34 sec**	**0**	**1,187**	**0.826**	**180**	
1.2.1 Run 1	Test App Full Function	1 hr 25 min 29 sec	40	729	1.469	180	MySleep Time=1.5
1.2.1 Run 2	Test App Full Function	1 hr 10 min 26 sec	73	731	1.539	180	Errors after 1 hr mark
1.2.1 Run 3	Test App Full Function	1 hr 9 min 51 sec	75	726	1.598	180	
AVERAGE		**1 hr 15 min 15 sec**	**63**	**729**	**1.535**	**180**	
1.2.2 Run 1	Test App User Controlled	1 hr 53 min 35 sec	0	384	1.770	180	MySleep Time=2.0
1.2.2 Run 2	Test App User Controlled	1 hr 3 min 59 sec (excluded from avg)	0	388	1.776	104	Manually terminated after 1 hour (104 iterations)
1.2.2 Run 3	Test App User Controlled	1 hr 57 min 24 sec	0	367	1.856	180	
AVERAGE		**1 hr 55 min 30 sec**	**0**	**380**	**1.801**	**155**	

Note: MySleepTime is the delay, in seconds, before each movement in the automatic replay of the test suite. The delay allows sufficient time for page rendering and screen draw so that the subsequent mouse click or other screen action will complete successfully.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Appendix C: Averaged Data – Raw Figures

Test Run	Internet Information Services Global\Cache Misses	Memory\Page Faults/sec	Network Segment(\Device\bh_OcTk161)\% Network utilization	Processor(0)\Interrupts/sec	Processor(0)\DPCs Queued/sec	Processor(0)\% Processor Time
1.1.1 Without TestApp Averaged	30.000	3.142	2.095	184.856	80.697	99.997
1.2.1 With TestApp - Full Function - Avg	145.333	10.554	6.199	208.987	104.143	99.995
1.2.2 With TestApp - User Controlled - Avg	173.333	6.942	20.181	383.452	274.784	67.356
1.3.1 Baseline Filter - Full Function - Avg (30 min)	3,115.667	20.898	28.739	319.587	214.120	9.600
1.3.2 Baseline Filter - User Controlled - Avg (30 min)	3,304.667	17.357	6.698	225.740	121.611	3.574

Note: Cache Misses is supplies as an accumulating count over the life of the boot. The amount of the increment over the one-hour test run was captured for each run.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

Appendix D: Filter Baseline Data – Raw Figures

Test Run	Baseline Setup	WCAT Pages Requested	Total Responses/sec	Data Read/sec	Total Bytes/sec
1.3.1 Run 1	Full Function	3,421	1.90	31,746.99	32,180.20
1.3.1 Run 2	Full Function	3,412	1.90	31,565.51	31,997.59
1.3.1 Run 3	Full Function	3,500	1.95	32,325.55	32,768.76
AVERAGE		**3,444**	**1.92**	**31,879.35**	**32,315.52**
1.3.2 Run 1	User Controlled	3,511	1.95	32,412.87	32,857.48
1.3.2 Run 2	User Controlled	3,500	1.95	32,325.55	32,768.76
1.3.2 Run 3	User Controlled	3,463	1.92	31,771.45	32,209.98
AVERAGE		**3,491**	**1.94**	**32,169.96**	**32,612.07**

Note: Only WCAT pages fetched.

© 2000 by Landmark Systems Corporation
and Patricia Garverick

QW2000 Panel 4P

Mr. Doug Whitney (Panel Moderator) (Intel)

Protecting Intellectual Property in an Open Source World

Panelists: Andy Wilson (Intel) Mitchell Baker (Mozilla)

BACK TO QW2000 PROGRAM

Abstract...

This is a case study of how we created a web appliance project using both intellectual property and open source code without jeopardizing the intellectual property. (Please note: This will NOT explain or interpret open source license issues)

Issue 1 - Separate code in a one build environment If we inadvertently link our proprietary source code with GPL open source the proprietary source becomes public property.

Issue 2 - Release requirements to the customer Open source code must independently compile and be available to customers.

Solution 1 - Ensure safety of proprietary IP. We ensure licenses are present for all source files. We use a clean room build environment to identify all linked files. For each linked file, we identify all sources used and flag the link as illegal if it contains both proprietary source AND GPL source.

Solution 2 - Ensure source shipped to customers meets distribution requirements. We identify all open source used in the build. We then ensure all this source is present within the distribution to be given to the customer. We compile this open source and ensure the compilation proceeds without errors.

BACK TO QW2000 PROGRAM

Developing Commercial Software in an Open Source World

A Panel Discussion With:

Doug Whitney

Andy Wilson

Mitchell Baker

Peter Nordquist

Introduction

- Who am I?
- Who is on the panel?
- Purpose
  - To share some real-world experience with a mixed Open Source and commercial SW development project and discuss some of the issues.

Ground Rules

- We are not here to interpret GNU license issues
- We are not here to tell you what to do for your projects
- View it as a case study with commentary

What is Open Source?

- Free SW Foundation General Public License (GPL) mandates that customers have access to source code, not just binaries. GPL applies to the Linux kernel and the FSF/GNU utilities.
- Other flavors of Open Source licenses (most importantly Netscape Public License for Mozilla) also mandate customer access to source.

What are the Challenges of Developing Commercial SW in an Open Source World?

- Each panelist will describe what they feel are the problems and opportunities presented to developers.
  - Mitchell Baker
  - Andy Wilson
  - Pete Nordquist
  - Doug Whitney

Our “Case study”

Project Overview

- Our project is a set top box using Linux, Mozilla, other open source and Intel developed intellectual property
- We are making modifications to existing Open Source code, and
- Our product contains newly created functionality which is not part of Linux or Mozilla
  - User interface for the set top, etc.

Success Metrics for Open Source Validation

- Strict compliance with Open Source licenses, as measured by the ability to hand over complete sources for GPL/NPL SW components when we distribute our products to customers
- High confidence, backed up by automated methods, that we have not unintentionally mixed our proprietary code with Open Source.

Automating Open Source Validation: A three step process

- Tag source code by license type in our source control system
  - “Blue code” is Open Source
  - “Green code” is Intel-written proprietary code
- Modify compiler to produce link maps
- Validate the link maps to make sure that “Blue” code is not inadvertently linked with “green”

Identify

- Create a clean room build system
- Find out from developers/build engineer what files are Intel developed and what directory they are in (“green” files)
- Create scripts to validate our “green” files have proper Intel copyright attribution
- Validate that “blue” directories have an Open Source license in root

Modify Compiler

- Link file maps show what files are linked and validate which libraries are used
- Change compiler options to create link validation files (work in progress)
- Re-run build with the enhanced compiler options
- Validate that link files are created

Check the links

- May be done individually by viewing each link map file. (Perl is our friend. ☺)
- Links made from "green" to "blue" are flagged as illegal
- Validate that all "green" files only link to other "green" files
- Log illegal links as defects

Open Questions

QW2000 QuickStart 4Q

Mr. Tom Gilb
(Result Planning Limited)

Pitiful and Powerful Measures of Software Metrics

BACK TO QW2000 PROGRAM

Presentation Abstract

What are 'Powerful and Pitiful Measures'? Powerful measures help management achieve their purposes as managers of software engineering teams. Powerful measures are not indirect: they measure as close to the customers' experience and need as possible. They help management to attain their customer-related objectives and to control their product development and production processes.

Measures become pitiful when they are not well suited to control what really matters. Often, inappropriate use of measures simply occurs because their use is either conventional or convenient.

To give some examples:

- function points: When used as a language-neutral measure of logic volume for comparison purposes, function points are a powerful measure. But when they are used as a 'primary or sole indicator' for estimating costs for a project, they become a pitiful measure. For any project, numerous cost drivers have to be taken into account (not least, the required quality levels).
- complexity metrics: As primary indicators of 'maintainability' or 'reliability', complexity metrics (such as McCabe's) are pitiful. They might well be extremely convenient to use (as automatic software tools are available to carry out the measuring), but they are much too indirect. Mean Time To Repair (MTTR) and Mean Time Between Failure (MTBF) are more worthwhile measures.
- defect density: Defect density is suited as a measure for controlling software process improvement: it can be used to track the injection of defects in a software process. However, using it to measure software quality is pitiful because it does not reflect 'availability' (net uptime to defined users). A reliability measure, such as MTBF in customer operational conditions, would be much more appropriate (as availability is a function of reliability (failure rate) and maintainability (fixing speed)).

About the Author

Tom Gilb is an independent consultant, teacher and author. He works mainly in UK, Europe and North America. He is resident in Norway.

Tom coined the term 'Software Metrics' with the publication of his book of the same name in 1976 (European edition) and 1977 (USA edition). This work is the acknowledged (by R. Radice and W. Humphrey) as inspiration for much of the Software Engineering Institute's Capability Maturity Model Level 4 (SEI CMM Level 4). His other books include Principles of Software Engineering Management (1988, now in 13th printing) and Software Inspection (1993 with Dorothy Graham). His main professional interest is the development of powerful Systems Engineering methods (covering Requirements, Design, Quality Control and Project Management).

BACK TO QW2000 PROGRAM

Slide 1

“Powerful Measures and Pitiful Measures of Software Engineering”

- ©By Tom Gilb, Consultant
- Gilb@acm.org
- URL: www.Result-Planning.com
  - Where many papers, books and slides are free for your further illumination!

October 12 1999

1976!

Slide 2

Powerful Metrics of Quality?

Excerpt from THE WALL STREET JOURNAL August 13, 1992

18 CHINESE MANAGERS EXECUTED FOR SHODDY QUALITY

BEIJING - Eighteen factory managers were executed for poor product quality at Chien Bien Refrigerator Factory on the outskirts of the Chinese capital. The managers - 12 men and 6 women - were taken to a rice paddy outside the factory and unceremoniously shot to death as 500 plant workers looked on.

Ministry of Economic Reform spokesman, Xi Ten Haun, said the action was required for committing unpardonable crimes against the people of China. He blamed the managers for ignoring quality and forcing shoddy work, saying the factory's output of refrigerators had a reputation for failure. For years factory workers complained that many component parts did not meet specification and the end product did not function as required. Compressors were cracked, leaked freon and the electrical components were sub-standard.

Complaining workers quoted the Plant Manager as saying, "Ship it.”

Customers, who waited up to five years for their appliances, were outraged, he says. "It is understandable our citizens would express shock and outrage when managers are careless in their attitudes toward the welfare of others."

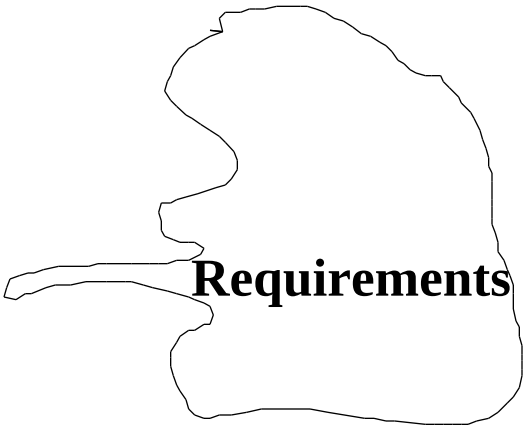
Refrigerators are among the most sought after consumer items in Communist China. "Managers in charge of production and engineering failed to perform any useful corrections to the quality problems for the last 20 years." Haun said. "Our soldiers are justified in wishing to bring proper justice to these errant managers."

The executed included the Plant Manager, the Quality Control Manager,

"Software Engineering" defined

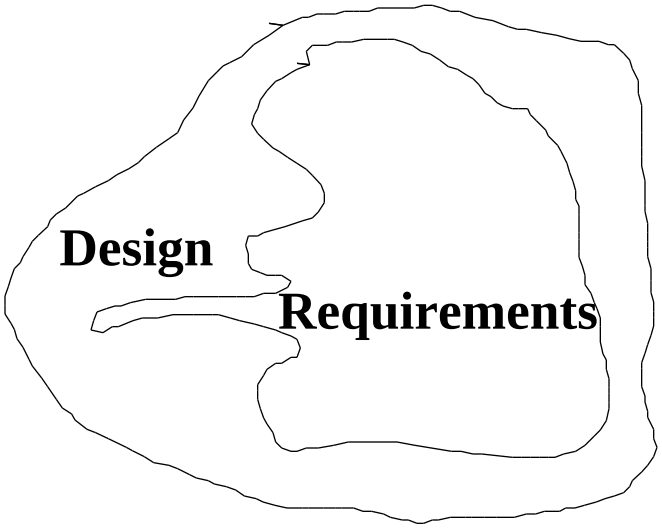
- **Software engineering is**
  - **the use of engineering *principles***
  - **to find *designs***
  - **which meet specified *requirements***
  - **under conditions of *uncertainty*.**
    - **'which meet' implies action until success**
    - **'designs' are anything which meets requirements**
    - **'requirements' are any objective, constraint, cost, time notion**
    - **'uncertainty' means no off-the-shelf product, but combinations of things with complex results**

Software Engineering Requirements



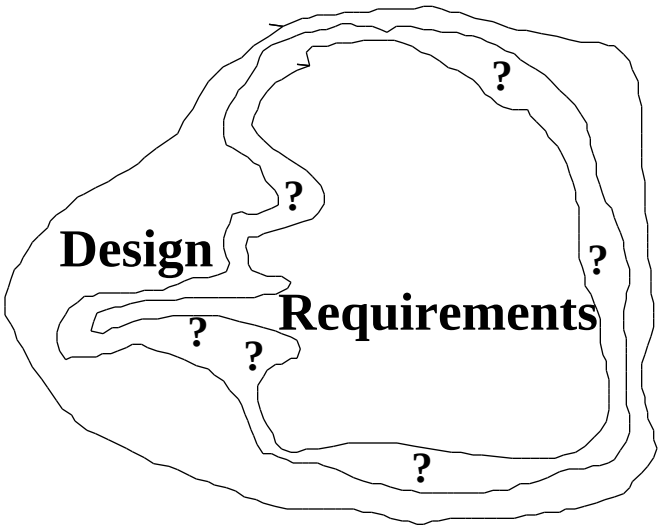
Slide 5

Software Engineering Design fitted to requirements



Slide 6

Software Engineering under ‘conditions of uncertainty’

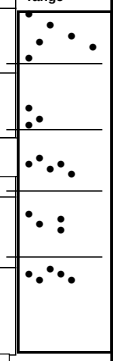


Slide 7

Stages of maturity : SEI Capability Maturity Model
SEI CMM Levels: A Pitiful Measure
But: 'Management Understands it' <--WH

	Maturity Level	Characteristics	Software Inspection Features
1	Initial	Depends entirely on individuals.	none
2	Repeatable	Policies, procedures, experience base	Writing-Task Rules, QA Policies, Inspection Procedures
3	Defined	Defined processes, peer reviews	Defect removal, Entry, Exit
4	Managed	Quantitative goals for product & process	Optimum rates, quality level at exit&entry,data summary, d-base
5	Optimizing	Entire organization. focused on continuous process improvement	Defect Prevention Process Improvements logging, Owners, Proc.. Change Mgt.. Team
(6	Design	Design to quality & cost	none) Not SEI level yet.

Results= Better(+) & smaller range



1-5 Based on Paulk et al, "Capability Maturity Model Version 1.1", IEEE Software, July 1993.

Slide 8

Why Baldrige?

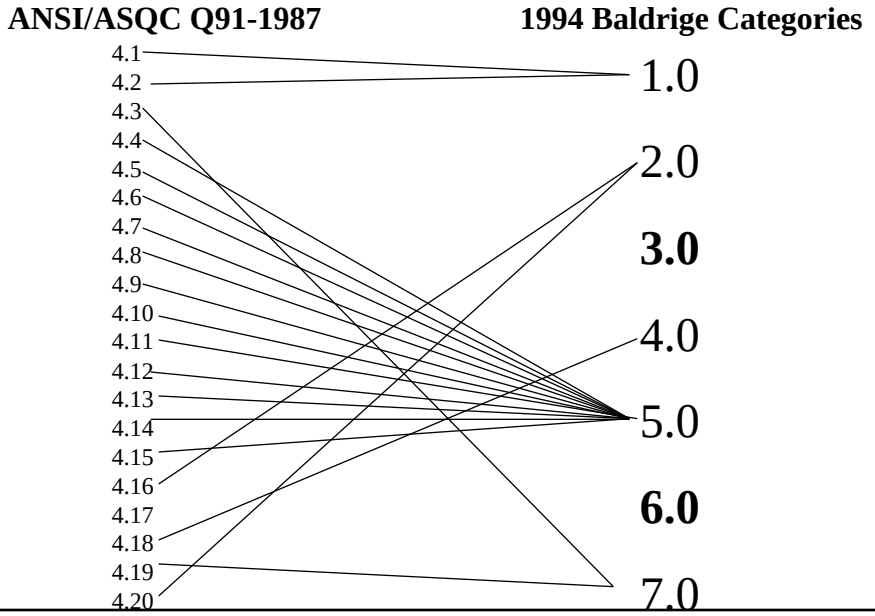
- Widely Used
  - N.I.S.T, Motorola, Xerox
- Proven Track Record
  - GAO Study
- Comprehensive
  - Greater scope than ISO or SEI's CMM
- Focus on Customer Satisfaction & Results

The next few slides about ISO/Baldridge are courtesy of Craig Kaplan, Ph.D. I.Q. Company, Author of 'Secrets of Software Quality' (McGraw Hill 1995)

http:\\www.iqco.com

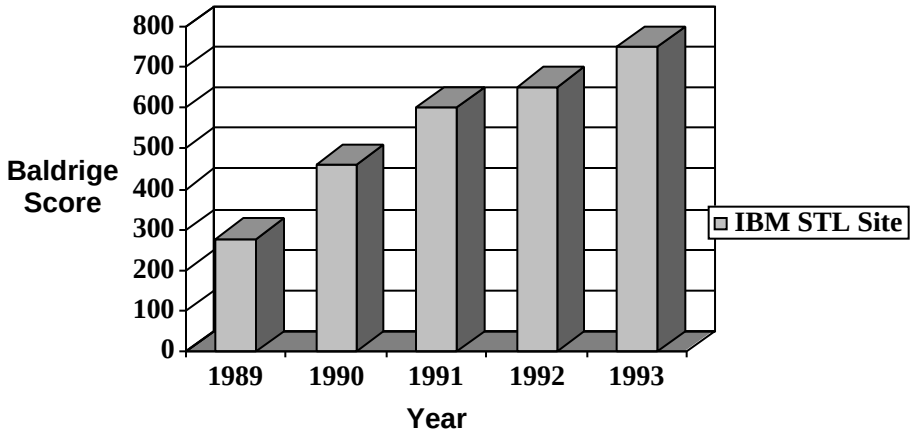
Slide 9

Mapping ISO 9001 to Baldrige



Slide 10

Baldrige Self-Assessment Scores at IBM's STL Site



Software Process Improvement at Raytheon: Some Really Useful Measures

- Source : Raytheon Report 1995: Dion et al
  - http://www.sei.cmu.edu/products/publications/95.reports/95.tr.017.html
- An excellent example of process improvement driven by measurement of improvement
- Main Motor:
  - "Document Inspection", Defect Detection
- Main Driver:
  - "Defect Prevention Process" (DPP)

Cost of Quality over Time: Raytheon 95

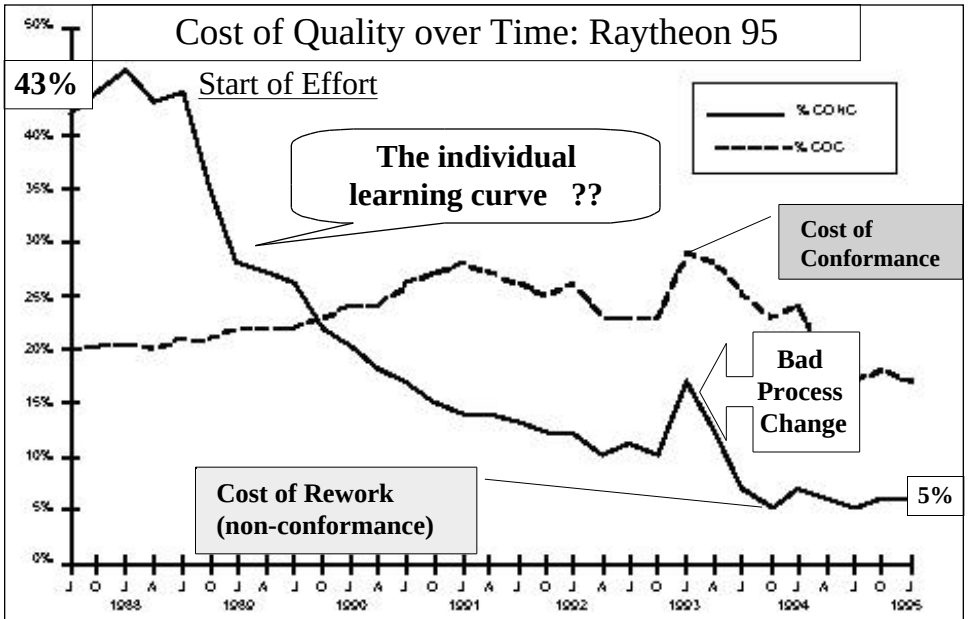


Figure 8: Cost of Quality Versus Time

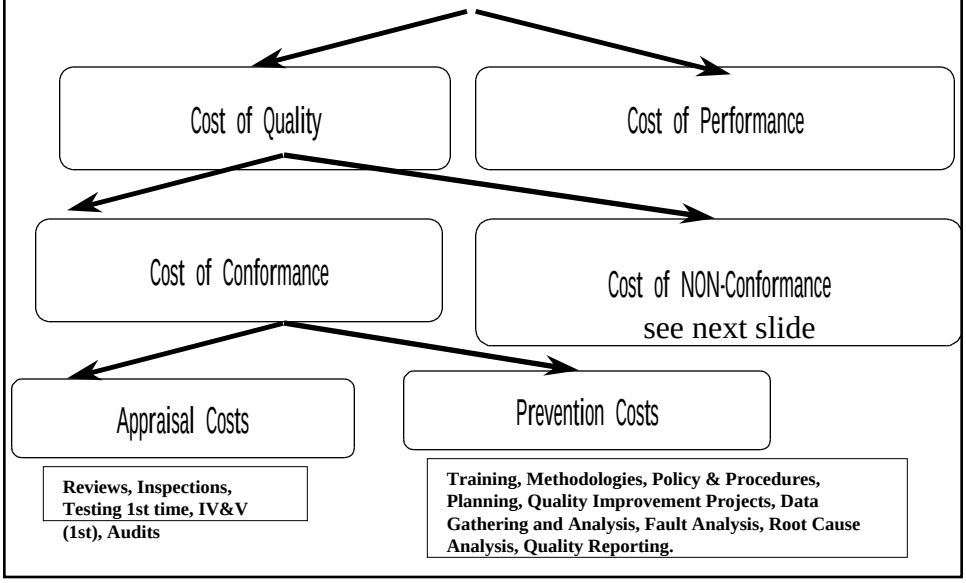
Slide 13

Rework Cost:
How you can set an objective to improve it

- **Gist: reduce by-half wasted development effort due to avoidable errors, if process improved.**
- **Scale: % of total effort which is applied to handling {identifying, correcting, re-testing, reissuing} avoidable errors.**
- **Past [Our test process, 1997] 45%**
- **Plan [Us, 1998 end] 30%,**
  - **[End 1999]20%,**
  - **[End 2000] 10%,**
  - **End 2001] 5%**

14

Project Cost



Costs of Non-conformance Items

- Re-reviews
- Re-tests
- Fixing Defects (code, documentation)
- Reworking any document.
- Engineering Changes
- Lab Equipment Costs of Retests

- Updating Source Code
- Patches to Internal Code
- Patches to Delivered Code
- External Failures
- from Crosby's Model according to Raytheon95 Fig. 7

Return On Investment

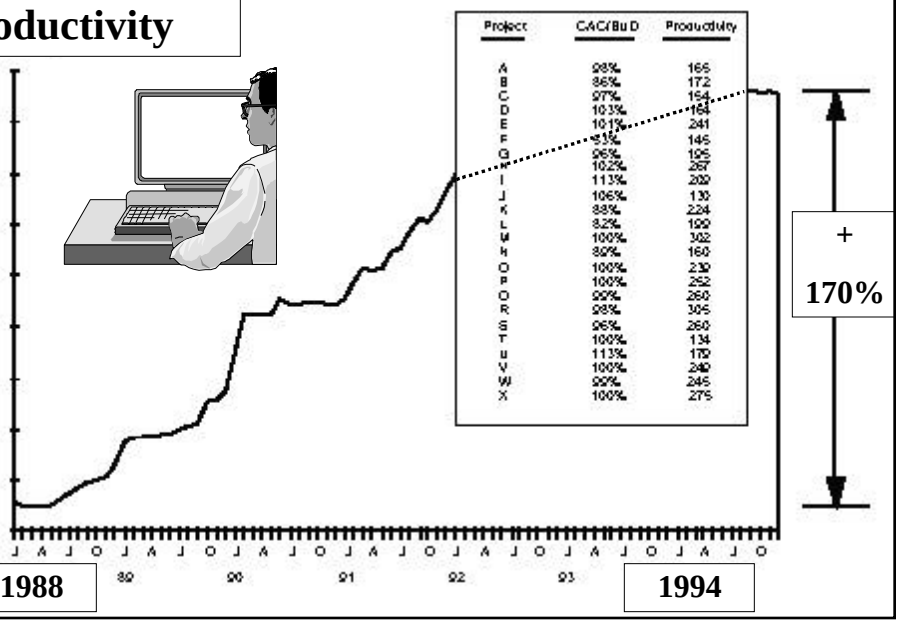
- $7.70 per $1 invested at Raytheon
- Sell your improvement program to top management on this basis
- Set a concrete target for it
  - –PLAN [Our Division, 2 years hence] 8 to 1

Copyright Gilb@acm.org

Raytheon 95 Software Productivity 2.7X better

17

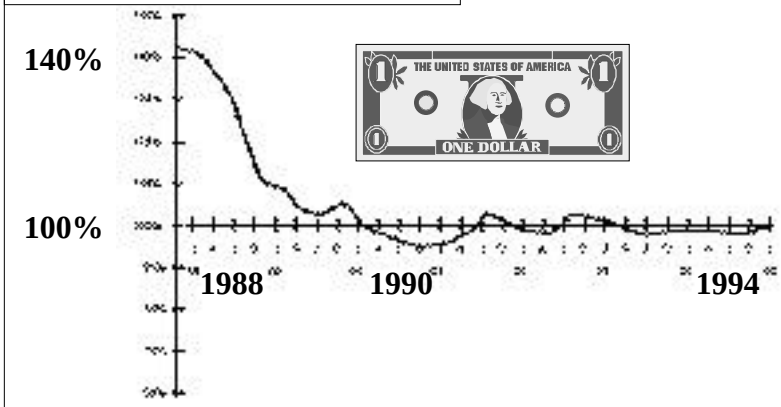
Productivity



Achieving Project Predictability: Raytheon 95

Slide 18

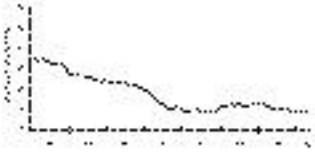
Cost At Completion / Budget %



Slide 19

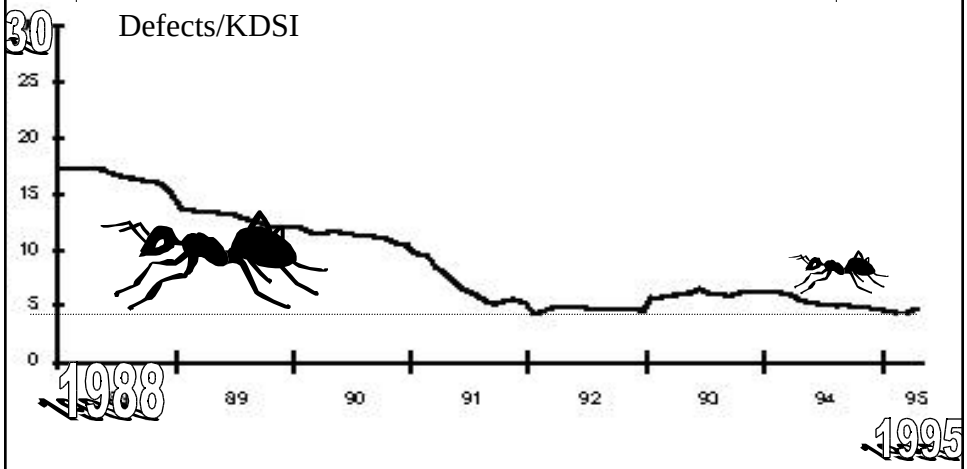
Overall 'Product Quality': Definition at Raytheon

- **'Overall Product Quality' = bug density!!**
- **The primary measure used to assess 'overall product quality' is the defect density in the final software products.**
- **We measure this factor in "number of software trouble reports (STRs) per thousand lines of delivered source code (STRs/KDSI)" on an individual project basis.**
- **The project defect densities are then combined to compute the monthly weighted average**
- **data shows an improvement from an average of 17.2 STRs/KDSI to the current level of 4.0**



Slide 20

Overall Product Quality: Raytheon 95 Defect Density Versus Time



Examples of Process Improvements: Raytheon 95

21

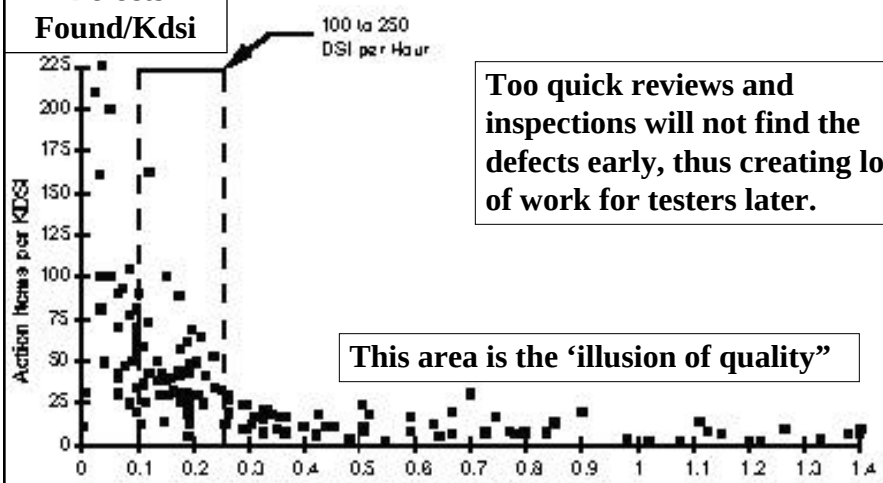
- **Process Improvements Made**

- **Erroneous interfaces during integration and test** -
  - Increased the detail required for interface design during the requirements analysis phase and preliminary design phase - Increased thoroughness of inspections of interface specifications
  - **Lack of regression test repeatability -**
  - Automated testing - Standardized the tool set for automated testing - Increased frequency of regression testing
- **Inconsistent inspection process -**
  - Established control limits that are monitored by project teams - Trained project teams in the use of statistical process control - Continually analyze the inspection data for trends at the organisation level
- **Late requirements up-dates** -
  - Improved the tool set for maintaining requirements traceability - Confirm the requirements mapping at each process phase
- **Unplanned growth of functionality during Requirements Analysis**
  - - Improved the monitoring of the evolving specifications against the customer baseline - Continually map the requirements to the functional proposal baseline to identify changes in addition to the passive monitoring of code growth - Improved requirements, design, cost, and schedule tradeoffs to reduce impacts

Fault Density versus Checking Rate: Raytheon 95

Slide 22

Defects Found/Kdsi



Too quick reviews and inspections will not find the defects early, thus creating lots of work for testers later.

This area is the 'illusion of quality"

Thousands of Statements Checked per hour by a person

Slide 23

Risk Policy Summary

- **.EXPLICIT RISK SPECIFICATION**
- **All managers/planners/engineers/testers/ quality assurance people shall immediately in writing, integrated in the main plan, specify any uncertainty, and any special conditions which can imaginably lead to a risk of deviation from defined target levels of system performance.**
- **NUMERIC EXPECTATION SPECIFICATION**
- **The expected levels of all quality and cost attributes of the system shall be specified in a numeric way, using defined scales of measure, and at least an outline of one or more appropriate 'Meters' (test or measuring instruments for determining where we are on a scale).**
- **CONDITIONS SPECIFIED**
- **The requirements levels shall be qualified with regard to when where and under which conditions the targets apply, so there is no risk of us inadvertently applying them inappropriately.**
- **COMPLETE REQUIREMENT SPECIFICATION**
- **A *complete* set of all critical quality and cost aspects shall be specified, avoiding the risk of failing to consider a single critical attribute.**
- **.COMPLETE DESIGN SPECIFICATION and IMPACT ESTIMATION**
- **A complete set of designs or strategies for meeting the complete set of quality and cost targets will be specified. They will be validated against all specified quality and cost targets (using Impact Estimation Tables). They will meet a reasonable level of safety margin. They will then be evolutionarily validated in practice before major investment is made. The Evo steps will be made at a rate of maximum 2% of budget, and 2% of 'project time', per 'incremental trial' (Evo step) of designs or strategies.**
- **.SPECIFICATION QUALITY CONTROL NUMERICALLY EXITED**
- **All requirements, design, impact estimation and Evolutionary project plans, as well as all other related critical documents such as contracts, management plans, contract modifications, marketing plans, shall be 'quality controlled' using the Inspection method [GILB93]. A normal process Exit level shall be *that 'no more than 0.2 Major Defects per page maximum, can be calculated to remain, as a function of those found and fixed before release, when checking is done properly'* (e.g. at optimum checking rates of 1 logical page or less per hour).**
- **7. EVOLUTIONARY PROOF-OF-CONCEPT PRIORITIES**
- **The Evolutionary Project Management method [Gilb97, Gilb88] will be used to sense and control risk in mid-project. The dominant paradigms will be**
  - **.2% steps,**
  - **.high value to cost with regard to risk delivered first.**
  - **.high risk strategies tested 'offline to customer delivery', in the Backroom of development process, or at cost-to-vendor, or with 'research funds' as opposed to project budget.**

Slide 24

Risk Policy Details

•Risk Policy Details

Skip The Policy Detail -->

Slide 25

1. EXPLICIT RISK SPECIFICATION

- **All**
  - **managers/planners/engineers/testers/ quality assurance people shall**
  - **immediately in writing, integrated in the main plan,**
  - **specify any uncertainty,**
  - **and any special conditions**
    - **which can imaginably lead to a risk of deviation**
    - **from defined target levels of system performance.**

Slide 26

2. NUMERIC EXPECTATION SPECIFICATION

- **The *expected levels of all quality and cost attributes of the system* shall be**
  - **specified in a numeric way, using defined Scales of measure, -|-|-|-|-|-|->**
  - **and at least an outline of one or more appropriate 'Meters'**
    - **test or measuring instruments for determining where we are on a scale.**

Slide 27

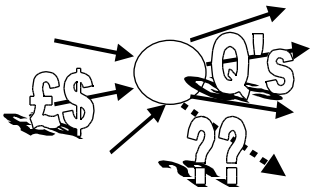
3. CONDITIONS SPECIFIED

- **The requirements levels shall be [qualified]**
  - **with regard to**
    - **[when]**
    - **[where]**
    - **and under which [conditions] the target requirements apply,**
    - **so we don't shoot for the wrong target!**

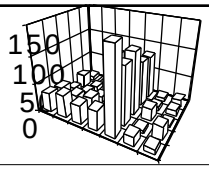
Slide 28

4. COMPLETE REQUIREMENT SPECIFICATION

- **A *complete* set of**
  - **all critical quality and cost aspects**
  - **shall be specified,**



  - **avoiding the risk of failing to consider even one critical attribute.**



Slide 29

5. COMPLETE DESIGN SPECIFICATION and IMPACT ESTIMATION:

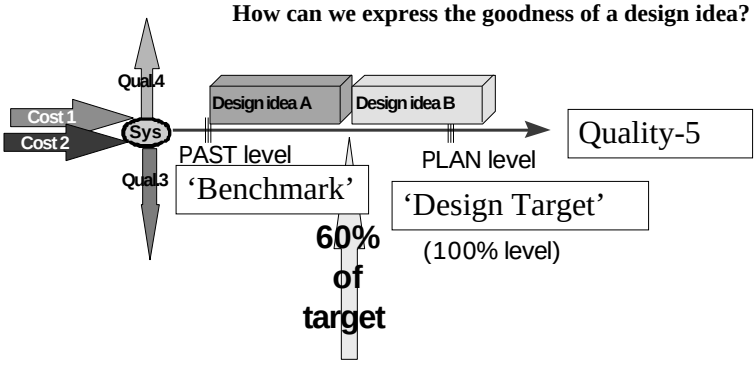
- **A complete set of designs or strategies for meeting the complete set of quality and cost targets will be specified.**
- **They will be validated against all specified quality and cost targets (using Impact Estimation Tables).**
- **They will meet a reasonable level of safety margin.**
- **They will then be evolutionarily validated in practice before major investment is made.**
  - **The Evo steps will be made at a rate of maximum 2% of budget, and 2% of 'project time', per 'incremental trial' (Evo step) of designs or strategies.**

Results Language Icons for requirements, design and design goodness

Slide 30

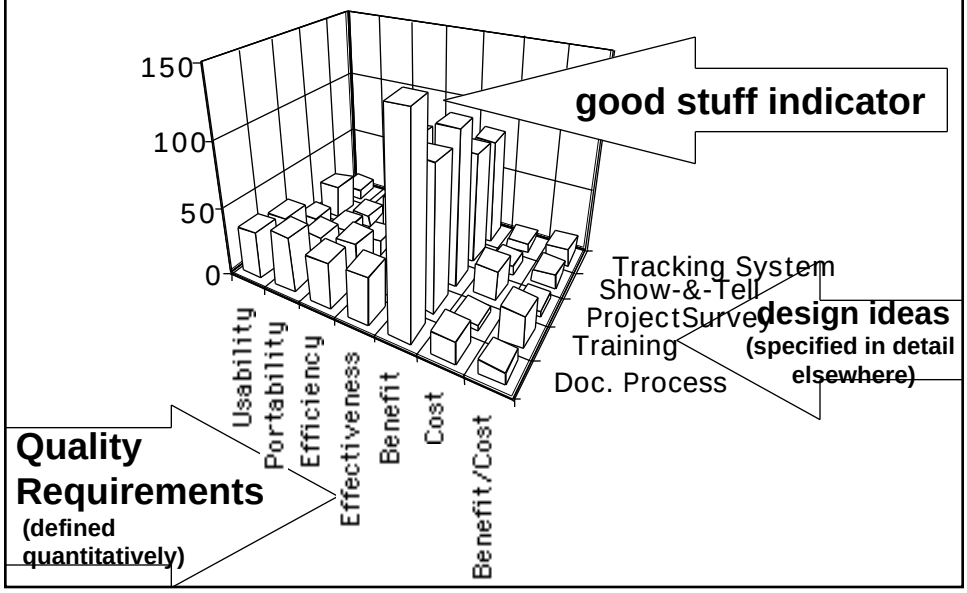
Results Language Icons for requirements, design and design goodness

How can we express the goodness of a design idea?



Slide 31

Impact Table 3D Display:
How good are design ideas compared to your objectives?



Slide 32

"Impact Estimation" concepts: full table

Tags of proposed TOTAL SET of strategies (defined elsewhere) to meet objectives, within resource constraints.

Strategies-> Objectives	A1	B4	CD	DX	Sum
AVAILABILITY "99.9%->99.98%" "PAST->PLAN"	0%	100%	50%	-5%	145%
PORTABILITY "80%->95%"	1	1	1	1	4%
USABILITY "3 mins.->1"	60 ±20	99	41	200	400%
BUDGET "0->1 million"	100	10%	?	n.a.	110?
EMPLOYEES "0->32 people"	0	30%		9±5	
Benefit/Cost->	0.6	5.0			

"Impact Estimation" concepts: detail

Tags of proposed TOTAL SET of strategies (defined elsewhere) for meeting the quality objectives, within resource constraints.

Rough sum of effects of all strategies on a single attribute's planned level.

Quality and Benefit Objectives

Resource Budget tags

Strategies-> Objectives	A1	B4	CD	DX	Sum
AVAILABILITY "99.9%->99.98%" "PAST->PLAN"	0%	100%	50%	-5%	145%
PORTABILITY "90% ->95%"	1	1	1	1	4%
USABILITY "3 mins.->1"	60 +20	99	41	200	400%
BUDGET "0->1 million"	100	10%	?	n.a.	110?
EMPLOYEES "0->32 people"	0	30%		9+5	
Benefit/Costs	0.6	5.0			

Clearly not good enough design yet

Safety margin 4X

Explicit uncertainty estimate

Estimation language:
0% = no effect with respect to PAST level.
100% = expected to meet PLAN level.
negative effect= makes things worse than PAST level.
? = no basis for an estimate.
n.a. = not applicable.

Sum Benefits / Sum resources = rough relative goodness of a strategy with respect to all objectives.

Objective statement example
SCALE: Avg. Minutes for typical user to learn to operate our product.
METER [accept] at least 100 users.
PAST[1993, Old Product] 3 minutes.
PLAN[New Produc,1995t] 1 minute.
[New Product, 1996] 30 secs.

Evidence
USABILITY:A1
Design method A1 in all competitive products and in our lab prototypes shows user learning time to be under two minutes. <- Lab Report U-92

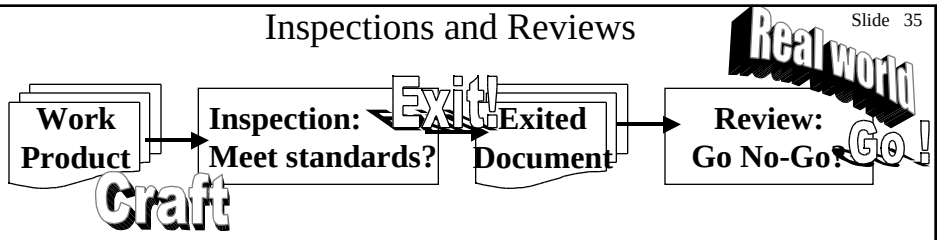
Strategy Definition Example
A1: Graphical interfaces using minimal language, no codes, maximum pictures, maximum user tailoring, maximum learning about particular users.

6. SPECIFICATION QUALITY CONTROL NUMERICALLY EXITED

- **All Specifications**
  - **{requirements, design, impact estimation and Evolutionary project plans,**
  - **as well as all other related critical documents such as contracts, management plans, contract modifications, marketing plans, }**
  - **shall be 'quality controlled' using the Inspection method .**
- **A normal process Exit level shall be**
  - ***that 'no more than 0.2 Major Defects per page maximum, can be calculated to remain,***
  - ***as a function of those found and fixed before release,***
  - ***when checking is done properly'* (e.g. at optimum checking rates of 1 logical page or less per hour).**

Inspections and Reviews

Slide 35



- **Inspections**
  - **Judgement based on conformance to standards**
  - **Well written, clear, complete, trustworthy**
  - **Can be carried out by any of 'intended readership'**
  - **Should be done to guarantee decision-makers a good basis for a decision.**

- **(Go No-Go) Reviews**
  - **Judgement based on goodness in real world**
  - **Content, not format;**
  - **Value, not clarity**
  - **Approval by authorized 'managers'**
  - **Should not be Entered if document not Exited from Inspection**

7. EVOLUTIONARY PROOF-OF-CONCEPT PRIORITIES

Slide 36

- **The Evolutionary Project Management method will be used to sense and control risk in mid-project. The dominant paradigms will be**
  - **.2% steps,**
  - **.high value to cost with regard to risk delivered first.**
    - » **.high risk strategies tested 'offline to customer delivery', in the Backroom of development process, or at cost-to-vendor, or with 'research funds' as opposed to project budget.**

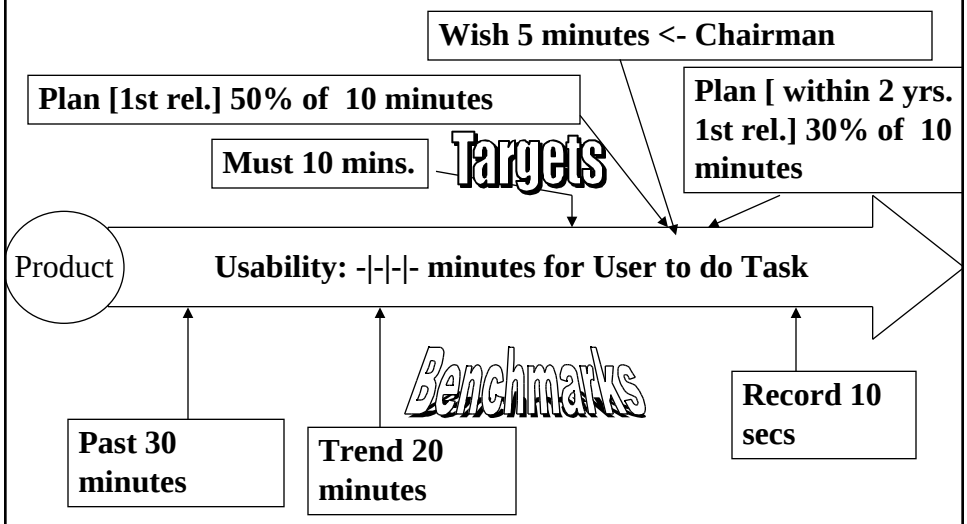
Evo Example:
Usability Goal, Complex example

Slide 37

- **Usability:**
- ***Gist: the relative ease of learning and using a defined product compared to previously used products.***
- **Scale: average minutes per [defined User type] to learn to use [defined Tasks to use the product].**
- **Meter: at least 30 users of representative defined User type will be monitored doing at least 10 defined Tasks of defined function type.**
- **Past [Old Product PP, Home Buyer, Adult, Task: Build telephone number list] 30 minutes.**
- **Record [MM, Adult, Task: Dial Out] 10 seconds ← Consumer Reports, January**
- **Trend [US Market, Children, Mix] 20 minutes ← Market Analysis Feb.**
- **Wish [Our Customers, Mix] 5 minutes ← Chairman's Dream in Speech**
- **MinLevel: Must [Our Customers, Mix, New Product] 10 minutes ← marketing minimum**
- **Plan [Our customers, Mix, New Product, First Field Release] 50% of MinLevel? ← Guess by Project Mgr., [within 2 years of First Field Release] 30% of MinLevel ←Guess.**
  - **Local Definitions of Terms.**
- **Mix: Defined: representative mix of common frequent user tasks.**
- **User: Defined: person who intends to use the product in the long term, not a test person.**
- **First Field Release: Defined: First sold releases to any public market after Field Trials.**

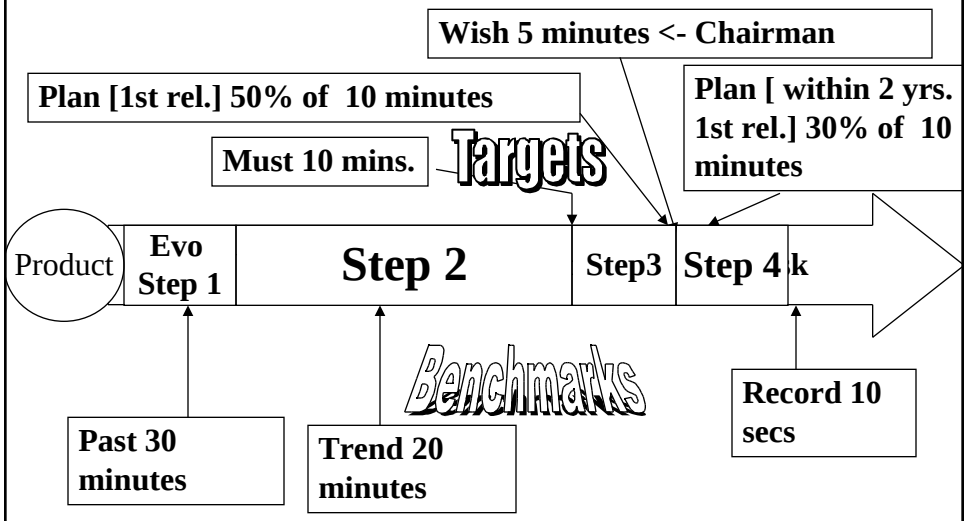
Usability Specification example

Slide 38



Usability Evo Delivery Steps to meet Plan

Slide 39



Impact Table for Step Management

Slide 40

	Step #1 Plan A: {Design-X, Function-Y}	Step #1 Actual	Differe-nce. - is bad + is good	Total Step 1	Step #2 Plan B: {Design Z, Design F}	Step #2 Actual	Step #2 Differe-nce	Total Step 1+2	Step #3 Next step plan
Reliabil-ity 99%-99.9%	50% ±50%	40%	-10%	40%	100% ±20%	80%	-20%	120%	0%
Perform-ance 11sec.-1 sec.	80% ±40%	40%	-40	40	30% ±50%	30%	0	70%	30%
Usability 30 min. -30 sec.	10% ±20%	12%	+2%	12%	20% ±15%	5%	-15%	17%	83%
Capital Cost 1 mill.	20% ±1%	10%	+10%	10%	5% ±2%	10%	-5%	20%	5%
Enginee-ring Hours 10,000	2% ±1%	4%	-2%	4%	10% ±2.5%	3%	+7%	7%	5%
Calend-ar Time	1 week	2 weeks	-1week	2 weeks	1 week	0.5 weeks	+0.5 wk	2.5 weeks	1 week

Slide 41

10 Principles of Risk Management:Summary

- **1. Frequent Feedback**
- **Early frequent and measurable feedback from reality must be planned into your development process, to identify risks before they become dangerous.**
- **2. Rigorous Requirements**
- **All critical success-and-failure quality/performance/cost requirements must be identified, made measurable and tracked through design and evolutionary deployment.**
- **3. Requirement Impact Estimation**
- **A design phase must address all critical few requirements and systematically estimate the impact of all design ideas on all critical requirements.**
- **4. Upstream Pollution Control**
- **All upstream documents (requirements, design) must be thoroughly inspected against a strong set of Rules for Good Practice, and not exited to next phases until they have reached a reasonable level of Major Defect Freeness.**
- **5. Personal Risk Responsibility**
- **People must be give personal responsibility in their sector for identification and mitigation of risks.**
- **6. Design Out Risk**
- **Unacceptable risk needs to be 'designed out' of the system consciously at all levels of engineering, architecture, purchasing, contracting, development process, motivation and maintenance process.**
- **7. Maximum Risk Policy**
- **The total level of risk exposure at any one stage should be consciously reduced to a minimum of about 2-5% of total budget, even with total failure of that stage alone.**
- **8. Maximize profit, not minimize Risk itself**
- **Focus not on elimination of all risk, but on maximization of benefit to cost result delivery, even considering risks.**
- **9. Backups are part of the Price**
- **Conscious planning and development of backup for risks is a necessary minimum cost of planning and projects.**
- **10. Contract Out Risk**
- **Make vendors contractually responsible for risks, they will give you better advice and services as a result.**

Pop Version, Next slide-->>

Slide 42

'Pop' Version Risk Principles 1->5 of 10

- 1. Frequent Feedback
  - **You've gotta shoot real bullets to see if they shoot back**
- 2. Rigorous Requirements
  - **Your project Achilles Heel is the critical requirement that you didn't think was so critical.**
- 3. Requirement Impact Estimation
  - **Design for success, or failure will have designs on *you*.**
- 4. Upstream Pollution Control
  - **If you're infected, don't kiss your friends hello!**
- 5. Personal Risk Responsibility
  - **Take risks personally, or they will take you personally.**

Slide 43

‘Pop’ Version Risk Principles 6->10 of 10

- 6. Design Out Risk
  - **If you don’t choose the risk level, risk will make it’s *own* decisions.**
- 7. ‘Maximum Risk’ Policy
  - **If you might fail, you might as well do it on a *small* scale.**
- 8. Maximize profit, not minimize Risk itself
  - **Risk is just a disturbance, profit is the point**
- 9. Backups are part of the Price
  - **If you leave the umbrella at home, that’s when it rains.**
- 10. Contract-Out Risk
  - **Let *others* bear the risks at hand, so they will join your merry band**

Slide 44

10 Principles of Risk Management: which relate to Software Metrics

10 Principles of Risk Management: •Details

Skip Risk Principles detail ->

Slide 45

1. Frequent Feedback

"You've gotta shoot real bullets to see if they shoot back"

- **Early frequent and measurable feedback from reality must be *planned* into your development process, to identify risks before they become dangerous.**

Slide 46

2. Rigorous Requirements:

"Your project's 'Achilles Heel' is the critical requirement that you didn't think was 'so' critical."

- **All critical success-and-failure quality/performance/cost requirements must be identified, made measurable and tracked through design and evolutionary deployment.**

Slide 47

3. Requirement Impact Estimation:
"Design for success, or failure will have designs on *you*."

- **A design phase must address all critical few requirements and systematically estimate the impact of all design ideas on all critical requirements.**

Slide 48

4. Upstream Pollution Control
"If you're infected, don't kiss your friends hello!"

- **All upstream documents (requirements, design) must be thoroughly inspected against a strong set of Rules for Good Practice, and not exited to next phases until they have reached a reasonable numeric level of Major Defect Freeness.**
- ***(like Max. 3.0 or 0.3 Maj/Pg)***

Slide 49

5. Personal Risk Responsibility:

"Take risks personally, or they will take you personally."

- **People must be given *personal responsibility* in their sector for identification and mitigation of risks.**

Slide 50

6. Design Out Risk:

"If you don't choose the risk level, risk will make it's own decisions."

- **Unacceptable risk needs to be *'designed out'* of the system consciously at all levels of engineering, architecture, purchasing, contracting, development process, motivation and maintenance process.**

Slide 51

7. Maximum Risk Policy:

"If you might fail, you might as well do it on a small scale."

- **The *total level of risk exposure* at any one stage should be consciously reduced to a minimum of about 2-5% of total budget, even with total failure of that stage alone.**

Slide 52

8. Maximize profit, not minimize Risk itself:

"Risk is just a disturbance, profit is the point"

- **Focus not on elimination of all risk,**
  - **but on maximization of 'benefit-to-cost' result delivery,**
  - **even considering risks.**

Slide 53

9. Backups are part of the Price:

"If you leave the umbrella at home, that's when it rains."

- **Conscious planning and development of backup for risks is a necessary minimum cost of planning and projects.**

Slide 54

10. Contract Out Risk:

"Let <u>others</u> bear the risks at hand, so they will join <u>your</u> merry band"

- **Make vendors contractually responsible for risks,**
- **they will give you better advice and services as a result.**

Last Slide

Goodbye!

:-)>

Note this is skipped to by summary of risk principles

QW2000 Vendor Technical Paper 4V2

John Bowman
(Compuware Corporation)

Which comes first, the process or the tool?

BACK TO QW2000 PROGRAM

Key Points

- Establishing your software testing process
- Process Integration with complimentary software processes
- Determining your test automation criteria

About the Author

Mr. Bowman is the Principal Architect for the QualityPoint Software Testing Methodology at Compuware Corporation. He has 15 years of experience in the information technology industry with the last 9 focused on developing solutions for software testing and software test automation. He has applied theories to practice in the areas of establishing requirements based testing systems and the use of automated test tools.

BACK TO QW2000 PROGRAM

Which Comes First, the Process or the Tool?

John Bowman
Compuware Corporation

COMPUWARE

Copyright © 2000 Compuware Corporation

What is a software testing process?

- Webster (circa 1983)
  - Process -
    - a series of actions or operations conducing to an end; a continuous operation or treatment especially in manufacture
  - Procedure -
    - a particular way of accomplishing something
- Software Testing Process
  - a set of procedures that are focused on validating software functionality in support of overall project goals

Copyright © 2000 Compuware Corporation

Project Goals

- Make sure it ...
  - *…is on time!*
  - *…is on budget!*
  - *...works!!!!*
- Software Testing - is a measurement system for determining how well "it works"

COMPUWARE

Copyright © 2000 Compuware Corporation

What should a Software Testing System do?

- Identify the following:
  - What will be tested,
  - How will it be tested,
  - Where will it be tested,
  - Who will test it,
  - What approch(es) will be used,
  - What resources will be needed,
  - When will it be tested,
  - What happened during testing
  - What does is mean to us
- Sound familiar?

COMPUWARE

Copyright © 2000 Compuware Corporation

Establish your Software Testing Architecture

- Get Commitment
  - Senior Management (VP, CIO, CFO, COO)
- Adhere to the 3 R's
  - Repeatable
  - Realistic
  - Relevant
- Provide Valuable Information
  - Project Managers
  - Business Owners
  - Stake Holders

COMPUWARE

Copyright © 2000 Compuware Corporation

Software Testing Architecture

COMPUWARE

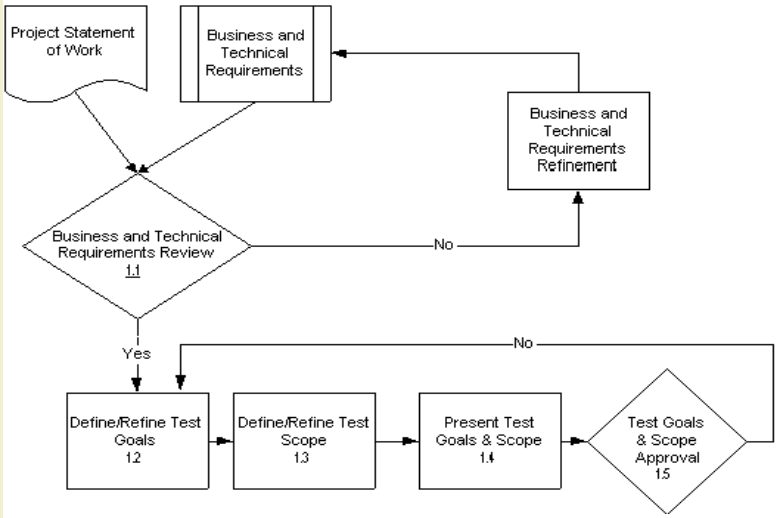
Copyright © 2000 Compuware Corporation

Support the Architecture with Processes

- Test Planning … (Who, What, Why and How)
- Test Case Development … (What)
- Test Environment Preparation … (How)
- Test Execution … (When)
- Test Results Analysis … (What Happened)
- Management Reporting … (What does it mean to us?)

COMPUWARE
Copyright © 2000 Compuware Corporation

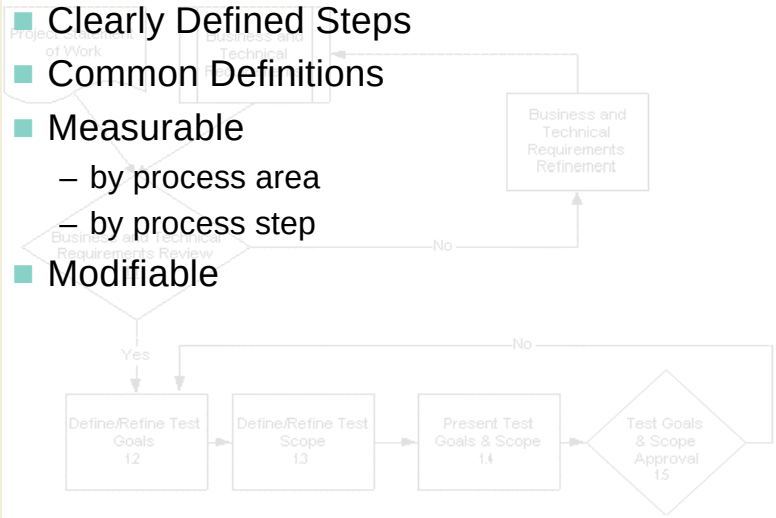
Support the Processes with Procedures



COMPUWARE
Copyright © 2000 Compuware Corporation

Support the Processes with Procedures

- Clearly Defined Steps
- Common Definitions
- Measurable
  - by process area
  - by process step
- Modifiable



COMPUWARE

Copyright © 2000 Compuware Corporation

Support the Procedures with Templates

- Software Testing Deliverables:
  - Test Strategy and Test Plan(s),
  - Test Cases,
  - Test Schedules
  - Test Results Logs,
  - Test Summary Reports,
  - Management Reports

COMPUWARE

Copyright © 2000 Compuware Corporation

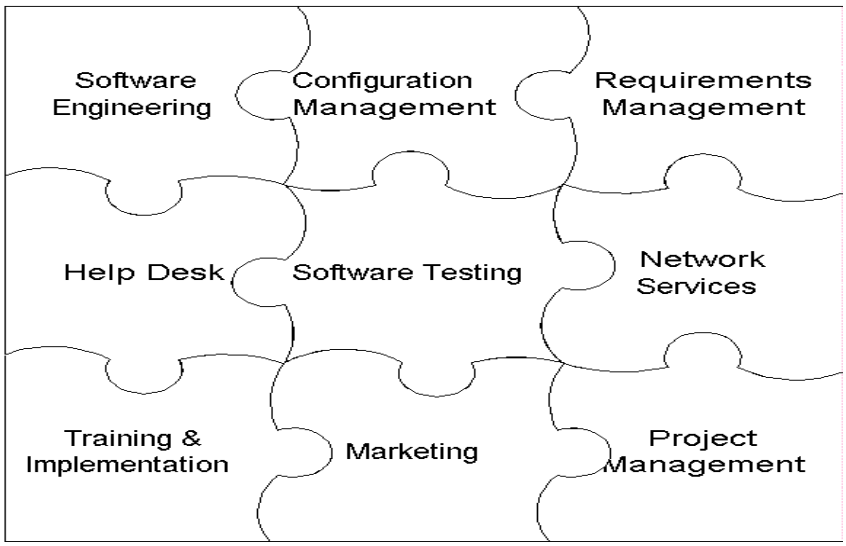
Measure the Effectiveness

- Identifies areas for the following:
  - Improvement
  - Automation
  - Cost Estimation
- Provides for
  - Control
  - Repeatability
  - Knowledge Retention
  - Documentation

COMPUWARE

Copyright © 2000 Compuware Corporation

Integration with Complimentary Processes



COMPUWARE

Copyright © 2000 Compuware Corporation

Automation Criteria

- Where to automate
  - Test Planning
    - Test Management Tools
  - Test Execution
    - Code Coverage Tools, Unit Testing Tools, Capture/Replay Tools, Data Driven Tools, Stress Testing Tools
  - Test Results Analysis
    - Defect Tracking Tools
  - Management Reporting
    - Data Mining tools, Reporting tools (HTML)

COMPUWARE

Copyright © 2000 Compuware Corporation

Automation Criteria (cont'd)

- How to automate
  - constrained by resources
    - Type of Resources assigned to testing
    - availability of technical support
  - impacted by technology change
    - # changes
    - types of changes

COMPUWARE

Copyright © 2000 Compuware Corporation

Automation Criteria (cont'd)

- What to automate
  - Critical functionality
  - High Path Frequency
  - Regulated Applications
  - Large Volume of Data

COMPUWARE

Copyright © 2000 Compuware Corporation

What the Processes Deliver…

- Information to make release decisions,
- Quantification of application risk
- Communication to other areas,
- Repeatability
- Efficiency

COMPUWARE

Copyright © 2000 Compuware Corporation

What the Tools Deliver…

- Repetition
- Precision
- Volume
- Consistency

COMPUWARE

Copyright © 2000 Compuware Corporation

Process Based Benefits

- Provides information for making decisions
  - Risk management
- Knowledge is retained
- Basis for communication to other areas
- Produces documentation
  - Litigation is a ever growing issue
- Facilitates Measurement
- Facilitates Automation
- Facilitates Process Improvement

COMPUWARE

Copyright © 2000 Compuware Corporation

Questions & Answers

COMPUWARE

Copyright © 2000 Compuware Corporation

Which Comes First, the Process or the Tool?

Introduction 3
Software Testing Processes and Project Goals 3
Software Testing Architecture 4
Key Process Area 1 - Test Planning (Who, What, Why and How) 4
Key Process Area 2 - Test Case Development (What) 5
Key Process Area 3 - Test Environment Preparation (How) 5
Key Process Area 4 - Test Execution (When) 5
Key Process Area 5 - Test Results Analysis (What Happened) 5
Key Process Area 6 - Management Reporting (What does it mean to us?) 5
The Role of Tools in Support of the Process Model 6
Automation Criteria 6
Conclusion 6

Introduction

The pervasiveness use of software in support of business operations has been well documented. However, most organizations that either develop or implement software to support their operations are not satisfied with results that come from their software testing system. The term "system" is used intentionally to open a new view as to the contribution of software testing and quality to the deployment of software systems.

Do any of the following questions sound familiar?

Does the software testing process provide them with the information they need to make sound release decisions?

Can the Project Manager accurately predict everything required to adequately test the software in the time allotted—the people, software, hardware, tools, effort, money, etc.?

Do the testing project team members know what is expected from them?

Do all project members, including senior management, know the status of the project and progress towards meeting expectations?

Upon implementation, does the software application actually meet the needs of its users?

Many times, software testing is not considered until the end of the development process and only given the resources "available at the time" to complete. As business processes have become more complex, the software applications necessary to support those processes have become more complex. The software testing systems must keep pace with the software in order to be effective.

To that end, there have been a plethora of tools that "solve the problems" of the software quality industry. If the tools solved the problems, everyone would be following the same processes, using similar tools, achieving similar results. Since that does not appear to be the case, there must be a deeper correlation between processes and tools.

Software Testing Processes and Project Goals

For most projects, there are three (3) basic goals.

1) Deliver it on time…
2) Deliver it on budget…
3) Make sure that it works!!!

Of these, there are already measurement systems in place to assist project managers and executive management in monitoring #'s 1 and 2. However, most organizations do not have a measurement system in place that allows them to measure how well it (the software systems) work. That is value that software quality assurance and software testing activities add to these projects.

Software Testing Architecture

A Software Testing Architecture allows the Process Model for software testing to be established. As with any successful implementation, the degree and level of support provided by senior management as well as the development and testing is critical. Without the required support at all levels, effectively establishing this architecture and its supporting processes will be impossible.

The individual procedures contained each process area need to reflect the needs of the specific project. The role of this architecture is to establish a structure around which individual procedures can be constructed.

The following architecture represents the QualityPoint™ Software Testing Methodology offered exclusively by Compuware.

Each of the Key Process Areas needs to be supported by tailored process flow diagrams, documentation and appropriate templates. The purpose of which is to allow the process model to become specific to the organization so that it meets their needs.

Key Process Area 1 - Test Planning (Who, What, Why and How)

The first purpose of Test Planning is to identify the items being tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan. The second purpose is to develop the scope, approach, resources, and schedule of the testing activities. Thorough test planning is critical to the success of the testing process.

Ideally, the Test Plan is developed at the time the application requirements are defined. This provides the test team adequate time to define the tests, locate and configure the hardware and software resources, locate and train the human resources, and schedule the tests. The test plan is a "living document" that will change as the application functions become more clearly defined and stable.

The process for creating the test plan, along with the test plan itself, should be reviewed and agreed to by the Test Team, the Development Team, the User/Customer Team, and ultimately, the Management Team. This step cannot be overemphasized because it establishes commitment from each contributing team.

Key Process Area 2 - Test Case Development (What)

The purpose of Test Case Development is to document the test cases that will be executed during this testing effort. These test cases will be used to test the application functions identified by the Test Plan. The number and type of test cases created are dictated by the test plan. A test case will identify the input values to be provided to the application, the procedures for applying those inputs, and the expected application values for the procedure being tested.

Key Process Area 3 - Test Environment Preparation (How)

The purpose of Test Environment Preparation is to document the steps necessary to prepare the environment for test execution. This process describes the environment, defines the initial setup, identifies data requirements, dependencies, and recovery points, and establishes procedures for backup and restoration of the environment.

Key Process Area 4 - Test Execution (When)

The purpose of Test Execution is to apply the test cases and log results. The following items are examples of results typically documented during test execution:

- Actual Results achieved during test execution compared to expected application behavior from the test cases.
- Test Case completion status (Pass/Fail).
- Actual Results of test execution that trigger suspension criteria.
- Deviations taken from the test plan or test process.

Key Process Area 5 - Test Results Analysis (What Happened)

The purpose of Test Results Analysis is to compare results of the test execution with the acceptance criteria defined by the Test Plan. Test Results Analysis includes a review of the Test Execution Log, Test Execution Statistics, and Application Defects. Test Case completion status (Pass/Fail) is also evaluated during the course of this Key Process Area. Defects are logged into the defect control system as appropriate.

Key Process Area 6 - Management Reporting (What does it mean to us?)

The purpose of Management Reporting is to communicate the results of the Test Execution and Test Results Analysis KPA's to management. It communicates progress towards the test goals and business and technical requirements fulfillment. In other words, does the application perform the business function for which it was defined? Additionally, it is used to communicate the conclusions and

recommendations of the test team to the application development team and the user community.

The Role of Tools in Support of the Process Model

The "testing tools" currently available on the market are primarily focused on the test execution process area leaving the other aspects of the overall software testing architecture uncovered. That fact has lead to the "creation" of tools using word processors, spreadsheets, small database systems and HTML to support the remaining architectural areas.

The tools that are available from vendors are very effective when properly focused and supported. They provide the repetition, volume, precision and consistency that would be very difficult to achieve in a completely manual software testing system.

Automation Criteria

The software testing process needs to provide the measurements necessary for management to use testing tools effectively. With that in mind, there are some common criteria that can be used to determine the need and level of automation that your testing organization can support.

Typical candidates for automation include tests that have the following characteristics:

- Application functions that are critical
- Execute high path frequency
- Demonstrate functionality that is regulated
- Large data volumes
- Stress or Performance test related

Additionally, there are criteria that should be reviewed that will help determine how the automation will be performed as well as what will be automated. Using the wrong automation approach can have a devastating impact on the testing effort.

Typical items that should be reviewed before an automation approach should be determined:

- Resources available for testing and support of test automation
- Volume and types of changes being made to the application under test

Conclusion

An effective software testing system can become an integral part to the software development or deployment process. These systems can deliver the vital information necessary to make release decisions and assess the risks associated with the release. The efficiency and precision of the system is made more

effective with the use and support of the appropriate tools. So, the answer to the title of this paper is that the process comes first, paving the way and establishing fertile ground for maximum benefits to be harvested from the use of the tools.

QW2000 Paper 4T2

Dr. Mark Blackburn & Joseph Fontaine (Software Productivity Consortium)

Application of the Test Automation Framework for Model Analysis and Test Generation

BACK TO QW2000 PROGRAM

Key Points

- Overview describing the basic notions of model-based development and how model analysis and test generation can significantly reduce time and effort
- Summarizes some of the benefits obtained by some members that have used the Test Automation Framework. One member project achieved 2 orders of magnitude greater test coverage (higher quality and reliability). With five+ experiments performed, they indicated that the modeling of such requirements could be done in 50% of the time (reducing cycle time).
- Explains how using the Test Automation Framework changes the lifecycle processes; provides a perspective on how the process changes the approach from traditional testing to continuous model-based V&V.

Presentation Abstract

The following outline provides a high-level summary of the "overview" describing applications of the Test Automation Framework (TAF). The briefing starts with an overview describing the basic notions of model-based development and how model analysis and test generation can significantly reduce time and effort. It also summarizes some of the benefits obtained by some members that have used the TAF. One member project achieved 2 orders of magnitude greater test coverage (higher quality and reliability). With five+ experiments performed, they indicated that the modeling of such requirements could be done in 50% of the time of existing (traditional) testing process (reducing cycle time). They were able to unifying an adhoc process of test script development from incomplete textual requirements, some of which were in the heads of the testers, and eliminated the need for several internally maintained tools.

Through deployment of the TAF we have gained insight into how models will become important mechanism for capturing the domain-specific intellectual property of a business unit. We explain how using the TAF changes the lifecycle processes from a traditional testing effort to one based on a continuous model-based V&V approach.

About the Author

Dr. Blackburn is President of T-VEC Technologies, Inc. and co-inventor of the T-VEC system and serves as Chief Technologist at the Software Productivity Consortium. He has twenty years of

software systems engineering experience in development, project leadership and applied research in object technology, requirement and design specification, formal methods, and formal verification. He is also involved in developing strategies for integrating knowledge management and e-business, and has also been involved in applied research and technology demonstrations in web-based knowledge engineering, domain engineering, and reverse engineering. He earned a BS in Mathematics from Arizona State, MS in Mathematics from Florida Atlantic University, and a Ph.D. in Information Technology from George Mason University.

BACK TO QW2000 PROGRAM

SOFTWARE PRODUCTIVITY CONSORTIUM

Applications of the Test Automation Framework for Model Analysis and Test Generation

www.software.org

Mark R. Blackburn
blackbur@software.org

Joseph S. Fontaine
fontaine@t-vec.com

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

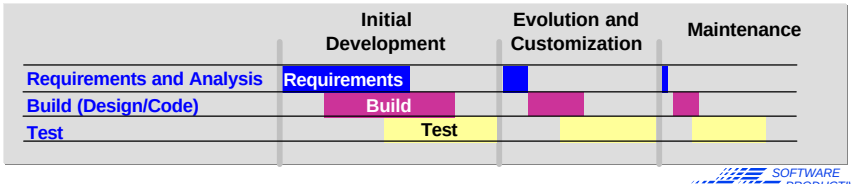
Problem

- Reduced cycle time of product development is critical to a companies'success
- Testing accounts for 40-75% of the effort
- Requirement defects account for 40-50% of failures detected during testing*
- Members spend 50% of test time debugging test scripts
- Member reported that feature interaction problem have resulted in as many as 30 test iterations prior to release

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Test Effort Increase Over System Life Cycles

- Initial development
  - Test approximately 30% of effort/cost
- Evolution
  - Test is relatively the largest activity >50%
- Maintenance
  - Test can now consume >70% of the time, effort and cost



SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

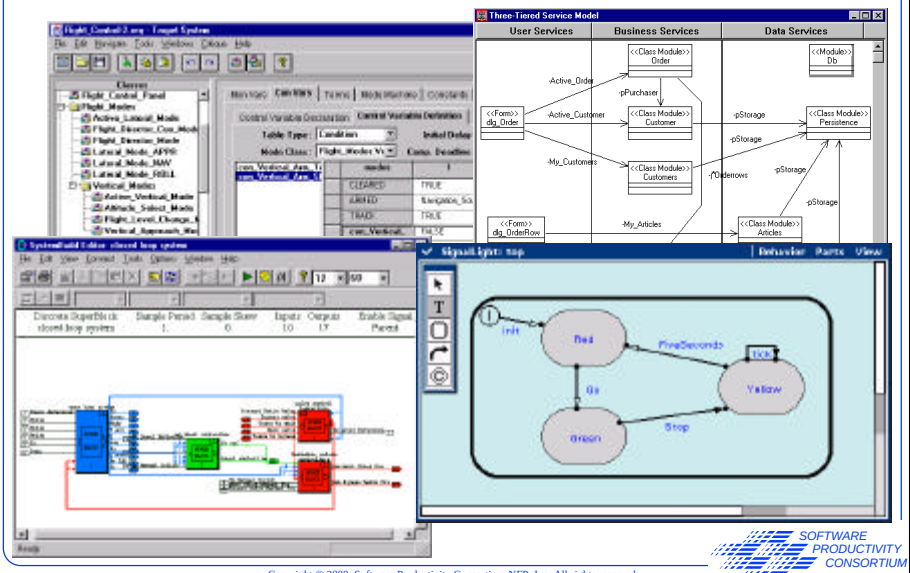
Solution

- Integration of model development tools with:
  - Model analysis capabilities to remove requirement defect
  - Automated test vector and test driver generation
    - Eliminates manual development of test cases and determination of expected outputs
    - Eliminates manual development of test scripts for target and host-based implementation
    - Provides requirements to test coverage analysis
    - Provides model simulation data to help validate the model through execution
  - Has been used for components evaluation
- Model translators provide the glue to integrate these capabilities

SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Visual Modeling Examples



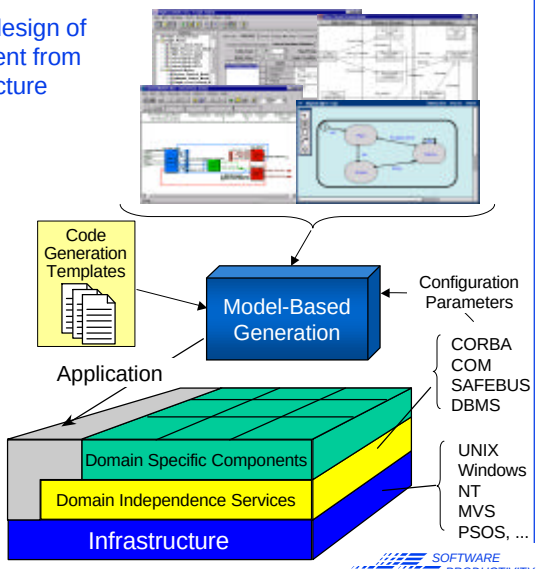
SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Why Model-Based Development ?

Models contain requirements & design of information/algorithms independent from language, platform, and infrastructure architecture

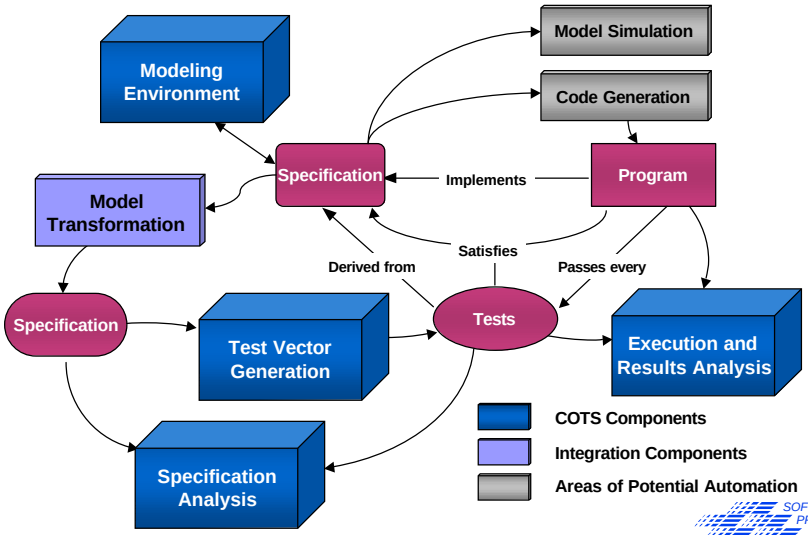
- Company assets are captured in models
- Models are translated into implementations
  - Tailorable code generation templates map to specific architectures and languages
  - Configuration parameters dictate platform details, services, and components
- Model-centric development supports robust software and process optimization



SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

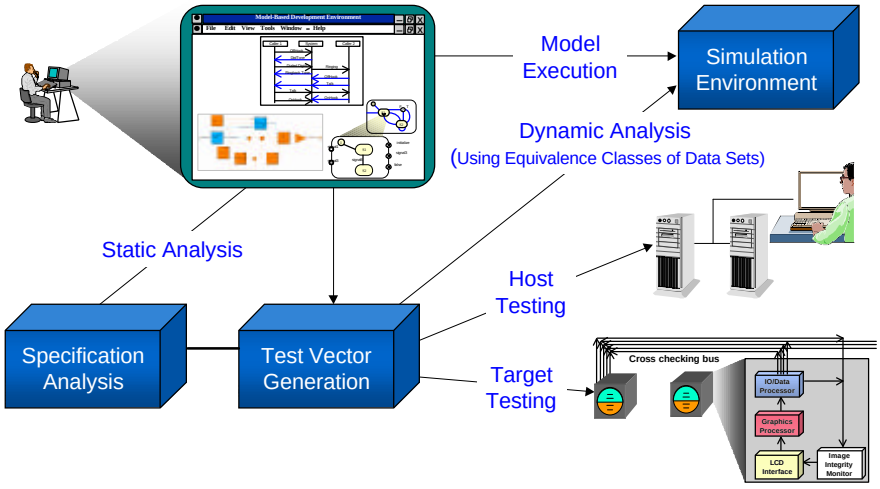
Tool Types for Model-Based Analysis and Test Automation



SOFTWARE PRODUCTIVITY CONSORTIUM

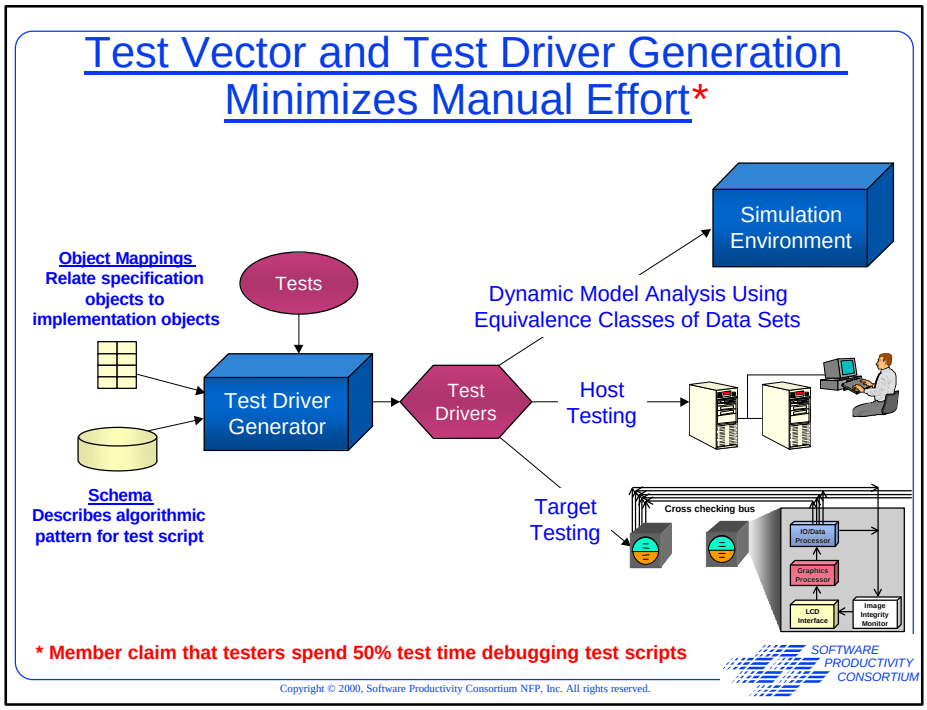
Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

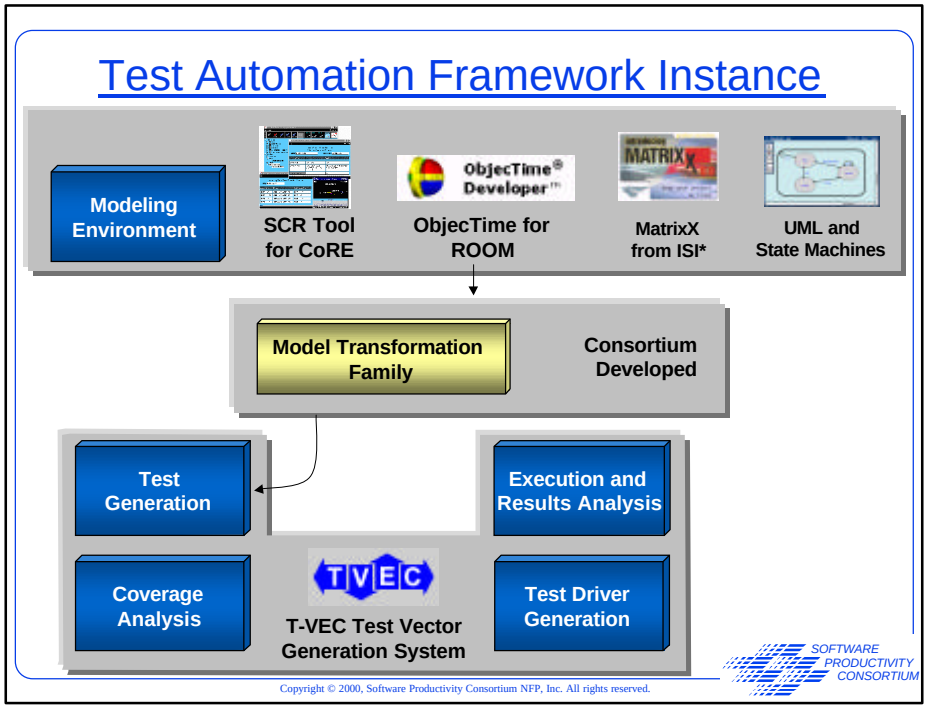
Using Models and Applying the TAF



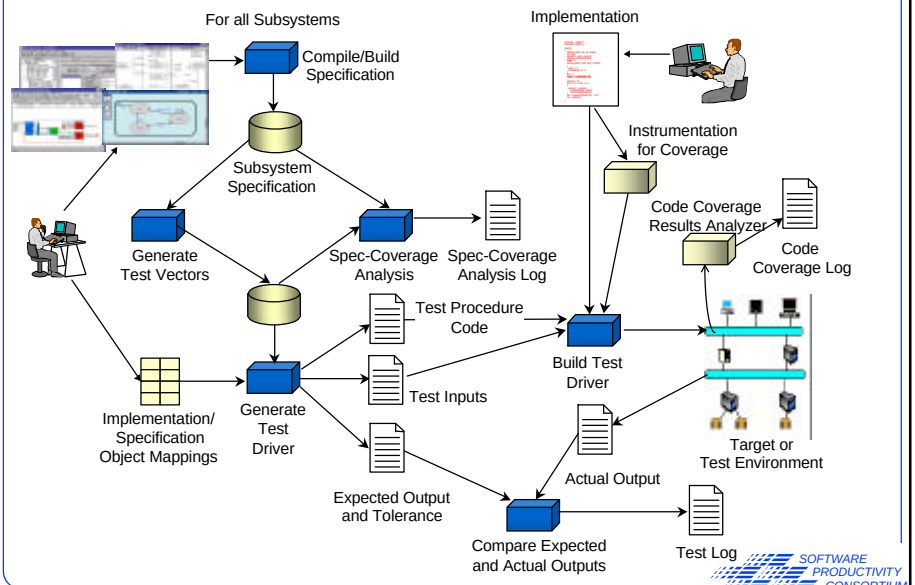
SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.


Test Vector and Test Driver Generation Minimizes Manual Effort*
Simulation Environment
Object Mappings
Relate specification objects to implementation objects
Tests
Dynamic Model Analysis Using Equivalence Classes of Data Sets
Test Driver Generator
Test Drivers
Host Testing
Schema
Describes algorithmic pattern for test script
Target Testing
Cross checking bus
IO/Data Processor
Graphics Processor
LCD Interface
Image Integrity Monitor
* Member claim that testers spend 50% test time debugging test scripts
SOFTWARE PRODUCTIVITY CONSORTIUM
Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.


Test Automation Framework Instance
Modeling Environment
SCR Tool for CoRE
ObjecTime for ROOM
MatrixX from ISI*
UML and State Machines
Model Transformation Family
Consortium Developed
Test Generation
Execution and Results Analysis
Coverage Analysis
T-VEC Test Vector Generation System
Test Driver Generation
SOFTWARE PRODUCTIVITY CONSORTIUM
Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Summary of Detailed Process Steps



Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Typical Organizational Situation

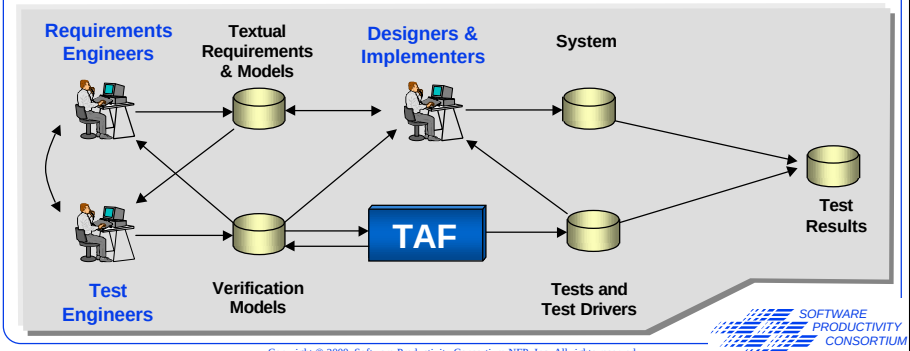
- Systems engineers develop textual requirements
  - Comfortable with textual requirements, function list, etc.
  - Pseudo rigorous graphics
- Software development and testing organizations work directly from textual requirements
- Nearly 50% of defects traceable back to requirements
  - Inconsistent or contradictory requirements
  - Poor documentation and incomplete requirements
- Testing effectiveness below goals
  - Requirement test coverage unknown or short of goals
  - Test effort over budget

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Effective Approach for TAF Deployment

- Tester involved in development of verification models by formalizing textual requirements
- Perform model analysis and test generation during development
- Feedback results of model analysis to improve both textual and modeled requirements

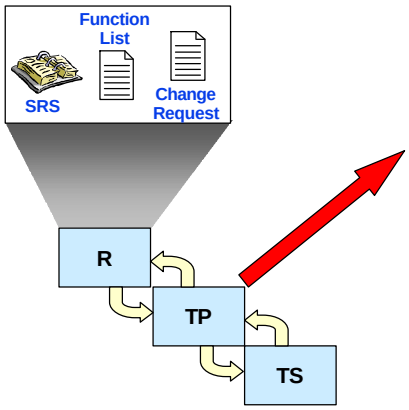


Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Traditional Requirement-Driven Test Planning

- Traditional test planning determines functions and associated conditions for each requirement



Test Plan Matrix	1	2	3	4	5
Conditions/Events					
Condition 1	T	T		F	F
Condition 2	F		F	T	F
Condition 3		T		T	F
. . .					
Function/Action					
Function 1	R				R
Action 1	R	R		R	
Function 2			R		
Function 3				R	
Function 4		R	R		
. . .					

Function 1 is Required when Condition 1 is True and Condition 2 is False and Condition 3 is a "dont care"

R = Requirement - TP = Test Plan - TS = Test Set

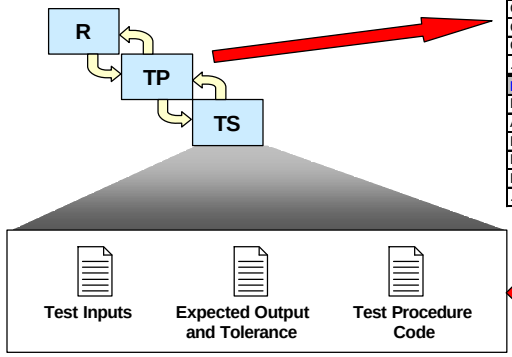
Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Traditional (Manual) Test Development

- Test inputs must satisfy the conditions
- Expected outputs calculated with respect to test inputs
- Test procedure (scripts) defined to execute test in environment

R = Requirement - TP = Test Plan - TS = Test Set



Test Plan Matrix	1	2	3	4	5
Conditions/Events					
Condition 1	T	T		F	F
Condition 2	F		F	T	F
Condition 3		T		T	F
. . .					
Function/Action					
Function 1	R				R
Action 1	R	R		R	
Function 2			R		
Function 3				R	
Function 4		R	R		
. . .					

SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Test Development Redundancies

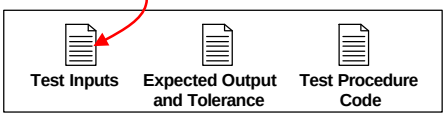
- Functions with common conditions often result in redundant tests

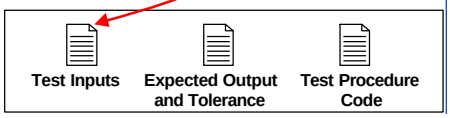
Test Plan Matrix	1	2	3	4	5
Conditions/Events					
Condition 1	T	T		F	F
Condition 2	F		F	T	F
Condition 3		T		T	F
. . .					
Function/Action					
Function 1	R				R
Action 1	R	R		R	
Function 2			R		
Function 3				R	
Function 4		R	R		
. . .					

Test Plan Matrix	1	2	3	4	5
Conditions/Events					
Condition 11	T	F		F	F
Condition 4	F		T		
Condition 1			F	T	F
. . .					
Function/Action					
Function 11					R
Action 1	R			R	
Action 2					
Function 31	R				
Function 6		R	R		R
. . .					

Test Engineer 1

Test Engineer 2





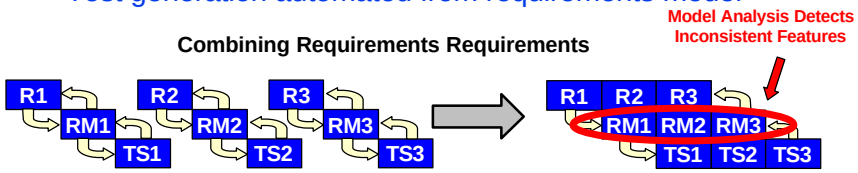
SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

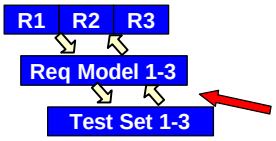
Model-Based Requirements Testing

- Model developed to characterize each requirement
- Test generation automated from requirements model

Combining Requirements Requirements



R = Requirement - RM = Requirements Model - TS = Test Set



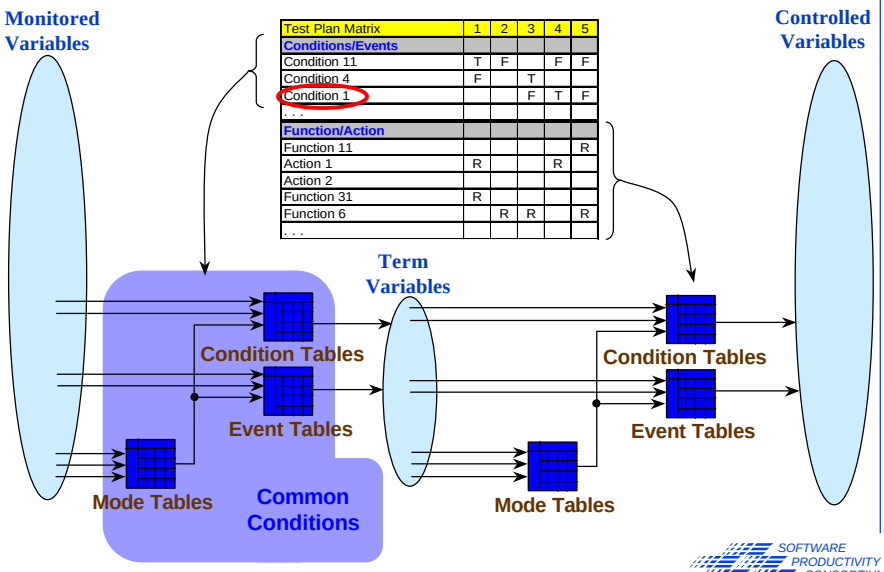
Requirements Development Not Serial.
As requirements are introduced, they are continuously modeled, integrated and refined are part of the requirements model.

Continuous automated verification of the model ensures requirements are well-formed.

SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Relating Test Procedures to SCR

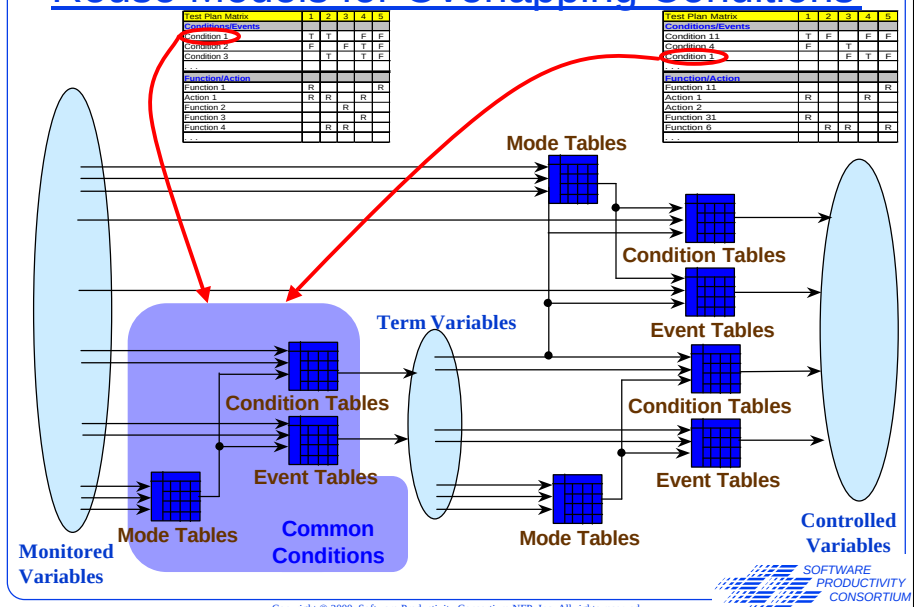


Test Plan Matrix	1	2	3	4	5
Conditions/Events					
Condition 11	T	F		F	F
Condition 4	F		T		
Condition 1			F	T	F
. . .					
Function/Action					
Function 11					R
Action 1	R			R	
Action 2					
Function 31	R				
Function 6		R	R		R
. . .					

SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

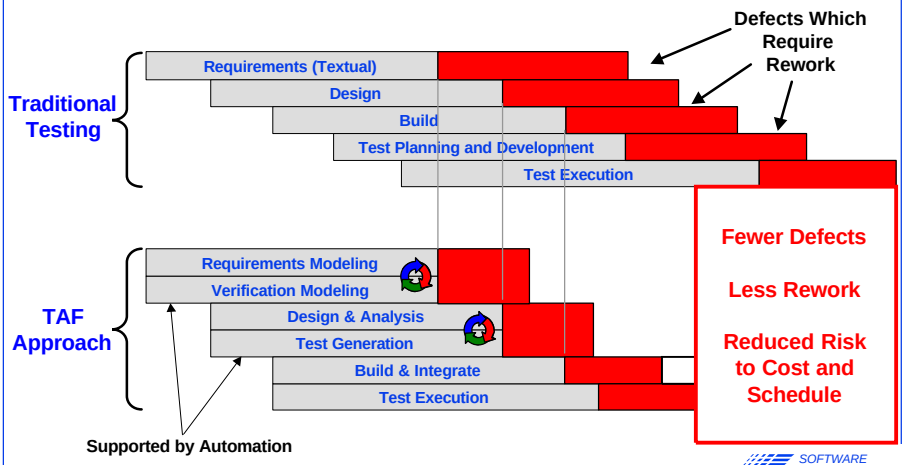
Reuse Models for Overlapping Conditions



Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Comparison of Traditional Approach versus Continuous Verification and Validation

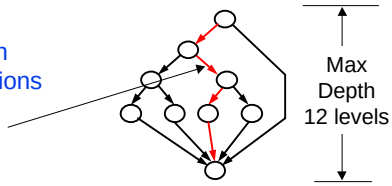


Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Member Applications of TAF

- Flight guidance avionics system
  - SCR model/Java implementation
  - Numerous modeling problem discovered after SCR model checks performed
  - 92% test coverage (MCDC) even though model still had contradictions
    - 6 paths out of 4620 not covered
    - Some utilities not specified
  - Developer told management that TAF approach "could save 40% of the full development cost, where SW Verif, as a whole, is currently at 50% total"



Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Member Applications of TAF

- Applied to system testing in an integrated set of systems
- 2 orders of magnitude increased test coverage
- Half order magnitude reduced effort
- SCR modeled by testers who have previously not modeled
  - Ramp-up about 3 weeks
  - Other models developed by person with CS degree (ramp-up only 2 weeks)
- Experiment showed that crucial business domain knowledge was not document - "only in the heads of the tests"
  - Captures legacy assets in models
- Eliminates needs for several internally maintained tools
- Unifies entire process by modeling time, module load, etc.

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

SOFTWARE PRODUCTIVITY CONSORTIUM

Member Applications of TAF

- MatrixX model
  - 200+ superblocks translated and vectors generated
  - Model anomalies identified (on model with long history)
- Code was generated for execution on Native NT target.
  - MS Visual C++ development environment used to build code for select superblocks
- Test Process applied… test drivers automatically generated
  - 185 super blocks verified successfully OVERNIGHT on the 1st attempt
- Another member applying to product line

SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Summary of Benefits and Risk Management Arguments

- Requirements defects or anomalies identified early – prior to design or implementation
- Requirement modeling appears to take no longer that traditional test planning, but reduces redundancy, and build on existing model
  - Model analysis eliminates requirement defects typically found during testing
  - Test generation from requirements model eliminates most manual test effort
  - Test generation provides measurable coverage of requirement-based tests
- Parallel test development permits "relatively" longer development time reducing risk of program schedule overrun

SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

Benefits and Results Summary

- Test automation to support manually intensive test generation and model-based analysis is feasible and practical for system and software testing
- Demonstrated for functional, OO, control system and hybrid modeling approaches
- Better quality requirements for design and implementation help eliminate rework in those phases as well as during test
- Verification modeling reduced the time normally spent in verification test planning about 50%
- Demonstrated to reduce time and effort by 50% while increasing coverage leading to increased quality and reliability by 200%
- Demonstrated to eliminate need for internally maintained tools
- Demonstrated as approach for modeling legacy assets

SOFTWARE PRODUCTIVITY CONSORTIUM

Copyright © 2000, Software Productivity Consortium NFP, Inc. All rights reserved.

QW2K Paper 4A2

Mr. Kevin VanFlandern
(Microsoft, Inc.)

Benchmarking Large Windows Based Applications

BACK TO QW2000 PROGRAM

Presentation Abstract

This paper covers methods used by the Office Performance team for retrieving and analyzing benchmark data against large Windows based applications. The contents include the objectives of the team, how we narrowed down our key goals and objectives, and how we set about, from both a hardware and software perspective, resolving each one.

The goal of our team is to gather and report comprehensive performance information about the applications that ship with Office for the purpose of tracking, identifying, and isolating performance issues that occur during the product cycle.

Given the many ways available to extract performance times we create a list of goals in priority order so that we will design the correct system to suite our needs. At first glance it may appear that simulating a real world user system, and providing close to accurate numbers would be our goal. Unfortunately since the data cannot consistently be reproduced, we are unable to track down and fix issue that are found, or even determine if the anomalies are in fact real issues. With that in mind we prioritize consistency of the data (or reproducibility), at the top of the list. The next priorities are being able to isolate an issue once it is found, then accuracy, and last real world.

The methodology we used in obtaining these goals will be covered including the following areas. The impact of hardware, specifically memory, processors, disk, and net works, on the application performance. How Operating system concerns such as configuration, virtual memory, disk layout and thrashing can affect your results. I will also cover our experiences and how we resolved the automation needs including preparing and starting the test, driving the application and reporting/logging the results with minimal effect on the data. Lastly I will briefly discuss tips on reporting and analyzing the results.

About the Author

(To Be Supplied)

BACK TO QW2000 PROGRAM

Benchmarking Overview

Kevin Van Flandern

4/26/2000 Office Fundamentals Performance 1

Project Goals…

- Consistency of gathered data
- Isolation of anomalies
- Precision of the Benchmark data
- Reporting data (volume)
- Accuracy or “real world” data

4/26/2000 Office Fundamentals Performance 2

Consistency of gathered data

- Brief history of past problems
- Consistency vs. Accuracy
- Controlling the hardware
- Controlling the software
- Quiet Network

History

- Very high STDEV from one build to the next
- Scenario based tests
- Similar hardware but radically different results
- Developers did not trust the data
- App groups started their own performance efforts

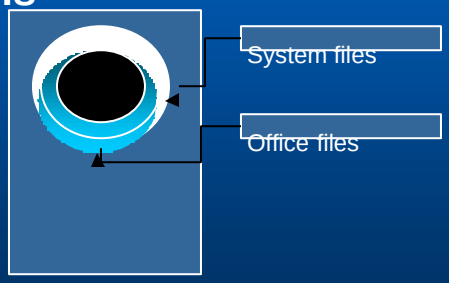
Consistency vs. Accuracy

- Actionable results
- Reproducibility
- Regressions
- Historical trends

4/26/2000 Office Fundamentals Performance 5

Controlling the hardware

- Build the system same components
- Tier Configurations
- Multiple Disks
- Disk Layout
- X10 technology



4/26/2000 Office Fundamentals Performance 6

Controlling the Software

- Minimize software running during tests (Jump Start)
- Performance SDK / Performance markers
- Driving the applications Merlin
- Creating and restoring clean defragged images
- Plugging into an automation system

C:

D:

Source OS

Target OS

4/26/2000 Office Fundamentals Performance 7

Controlling the Software 2

~~Clean Target Drive~~

~~Restore OS/Office image~~

~~Bring down test specific files~~

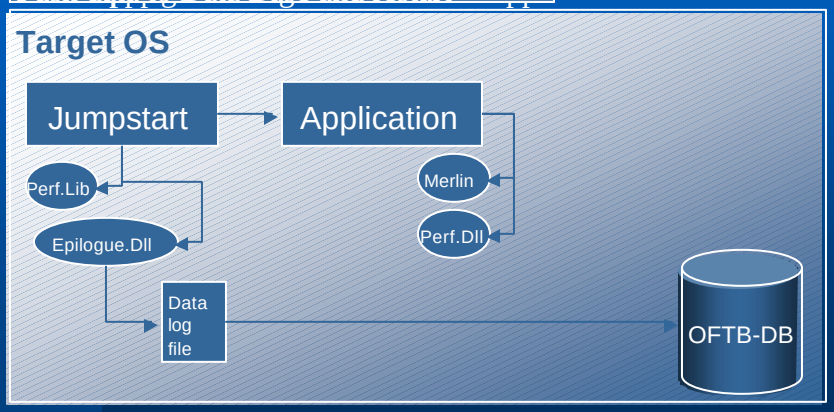
~~Boot to Target OS~~

D:\

4/26/2000 Office Fundamentals Performance 8

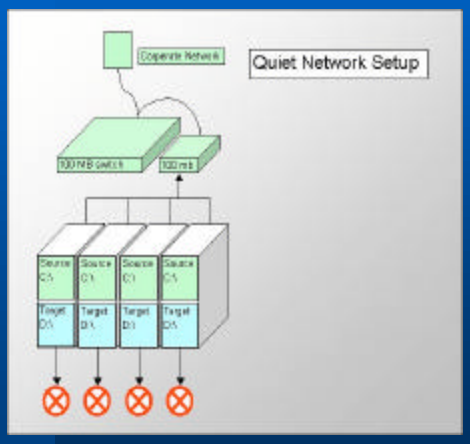
Controlling the Software 3

Push data to Data Base and restart



4/26/2000 Office Fundamentals Performance 9

The quiet network



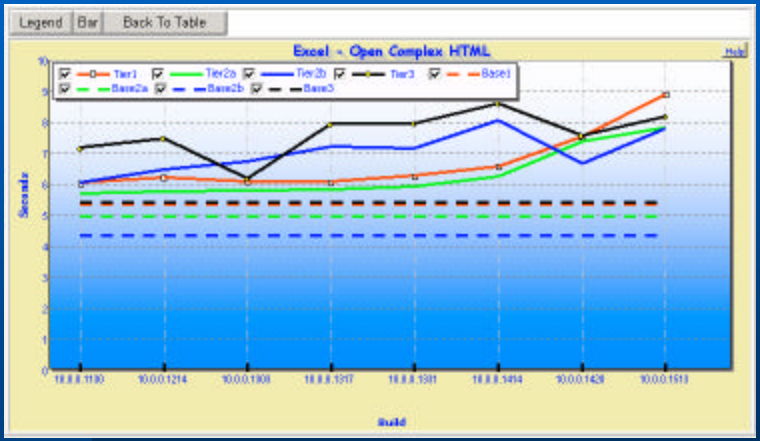
- **Ideally benchmark clients would run with no network. However...**
- **Short of a private Network and fully switch network will effectively eliminate net traffic impact on tests.**

4/26/2000 Office Fundamentals Performance 10

Isolation of anomalies

- Historical trends
- Validation of tests statistically
- Non scenario based tests
- Investigate fluctuations

Historical Trends



Validation of tests

- Verify the environment
- Verify the tools
- Check the Iterations
- Check the standard deviation of the averages
- Check standard deviation percentage of average

4/26/2000 Office Fundamentals Performance 13

Non scenario tests / Investigate fluctuations

- Scenario based tests create their own fluctuations
- Profiling tools
  - OffProf
  - Regmon
  - Filemon
  - Workset
  - Perfmon
  - RPC count
  - …

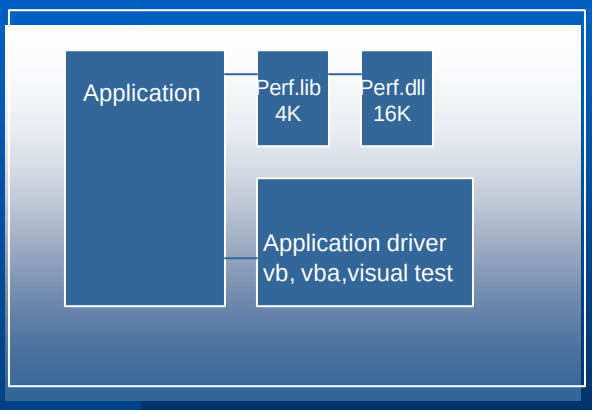
4/26/2000 Office Fundamentals Performance 14

Precision of the Benchmark data

- Application performance markers
- Perf.lib 4k (1 page)
- Perf.dll 16k (4 pages)

4/26/2000 Office Fundamentals Performance 15

Application Perf Markers



4/26/2000 Office Fundamentals Performance 16

Perf.lib

- Application load library if reg key exists
- Library preloaded into memory
- Perf .lib exposes Perf.dll API calls
- Under 1 page of memory 4KB

4/26/2000 Office Fundamentals Performance 17

Perf.dll

- Holds 150 64 bit marker pairs
- Supports the following API calls
  - Marker Morphing
  - Marker on off
  - Marker filter
  - Marker insert
- Logs to file on application close

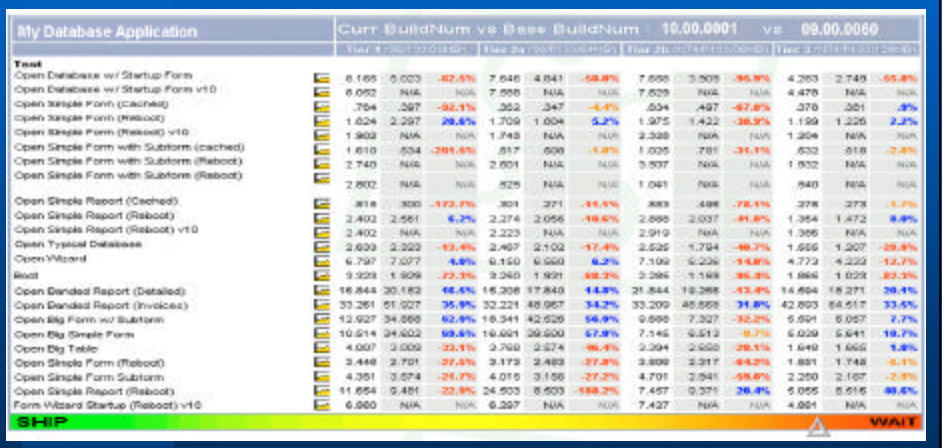
4/26/2000 Office Fundamentals Performance 18

Reporting data

- http://Perf/perf10
- Ship Criteria
- Full suite
- Data analysis
  - Metrics
  - Dev investigation tools

Demo Reporting tool

Reporting Page



Benchmarking large Windows based application

Kevin Van Flandern
KevinVF@Microsoft.com
(425) 703-1339

Level: intermediate

Keywords: Performance, Benchmarking, Automation

Biographical Sketch: Software Test Lead at Microsoft. I have been with MS for almost five years.

Abstract

This paper covers methods used by the Office Performance team for retrieving and analyzing benchmark data against large Windows based applications. The contents include the objectives of the team, how we narrowed down our key goals and objectives, and how we set about, from both a hardware and software perspective, resolving each one.

The goal of our team is to gather and report comprehensive performance information about the applications that ship with Office for the purpose of tracking, identifying, and isolating performance issues that occur during the product cycle.

Given the many ways available to extract performance times we create a list of goals in priority order so that we would design the correct system. At first glance it may appear that simulating a real world user system, and providing close to accurate numbers would be our goal. Unfortunately since the data cannot consistently be reproduced, we are unable to track down and fix issue that are found, or even determine if the anomalies are in fact real issues. With that in mind we prioritize consistency of the data (or reproducibility), at the top of the list. The next priorities are being able to isolate an issue once it is found, then accuracy, and last real world.

The methodology we used in obtaining these goals will be covered including the following areas. The impact of hardware, specifically memory, processors, disk, and net works, on the application performance. How Operating System (OS) concerns such as configuration, virtual memory, disk layout and thrashing can affect your results. I will also cover our experiences and how we resolved the automation needs including preparing and starting the test, driving the application and reporting/logging the results with minimal effect on the data. Lastly I will briefly discuss tips on reporting and analyzing the results.

Objectives:

Consistency:

When we first looked at trying to setup a performance lab we had thought that real world simulation was extremely important. The idea being that by simulating as close to real world as possible, we would catch performance anomalies in slow user situations that would normally be lost in a more clean environment.

It was not long after we started the process that we realized our primary goal as a test organization was not only finding performance issues but they needed to be actionable, and reproducible as well. We also found that there were many contributing factors in a "real world" environment that detract from the consistency of the data.

In our first system we set up several off the shelve Personal Computers (PC's) of different makes and models. The basic configuration of the PC's were the same i.e. same processor speed, memory configuration and drive size, however the manufacturer's and models were different. We were surprised at both the substantial difference from one PC to the other on a particular action timing (several hundred percent), as well as the number of hardware differences in the system.

The other immediate and glaring problem was the way the test scenarios were written. The scripts were set up to drive an application through 50+ actions and measure each one. Unfortunately each variance in performance created artifacts that affected the results of later actions. By the time we completed any given scenario, the memory had been so taxed and fragmented that the last handful of tests were no where near consistent. An example of this would be, if we opened a typical document first then a complex document later, much of the complex document load code would be cached. If the typical load failed to run properly then the complex document could take several hundred percent of it's normal load time. Even after only a handful of actions you are almost certain to see this sort of problem occur on scenario based tests.

It became very clear that when we started reporting results on a system build in the real world model, which 90% or more of the bugs were created in the lab, and required a huge amount of effort to track down. Even real bugs required an enormous effort to verify that the anomaly was real and warranted further investigation. When it came time to regress and track the actions, the same problem applied.

We solved these issues by concentrating our efforts on the problem of consistency in data, and began a several phase process of cleaning up the system up from a hardware and software perspective.

Isolation:

The next concern we faced in the lab was isolating the problem. In a non-clean environment this problem is monumental, so it is necessary to resolve the clean up portion of the environment before trying to tackle the isolation.

What we found was that after a problem occurred it was very difficult to track the problem down. This is an extension of the consistency issue, and required that we solved that concern first.

We found if we measured a degradation of an action by several hundred percent we then had to try and resolve the question. What caused this degradation? Was it the action itself? Was it a spike caused by the many unknown variables that are not being monitored around it? For example many people run a mail program and a browser in the background. However having such programs running can cause spikes in your data when their processes, or polling.

The solution here is simple in principle. Eliminate as many non-predictable influences to the environment so that you can confine performance anomalies to your tests and the action(s) within those tests. The benefit is that we can then automate the investigation process and use tools that monitor the key PC performance bottlenecks like Registry read writes, File I/O, Memory consumption, and processor usage.

Precision:

Precision falls third on the list because in addressing precision we found that you must first address consistency and isolation. In this phase of the process we were concentrating on getting timers that reflect exactly what the end user perceives that action to be. There were several blocks that we ran across.

The first was that we had no way of triggering the timers in the correct location. We would use an application driver like Visual Basic (VB) or Visual Test and attempt to measure File Open just to find that VB was unable to tell us precisely when the file open would start or stop. For some of the tests we tried to hook system idle to figure out when it was completed but that did not work very well. When we actually ran a profile on the application between the given markers we found that we were missing large chucks of code, and adding lots of extraneous code just in the timer placements alone.

Using Visual Test or VB plagued us with problem. The Dynamically Linked Library files (DLL's) that come with these programs uses five Megabytes (MB) of working set space, and constantly polls the system causing unpredictable

fluctuations. In the timers, VB drives the application via the object model (assuming you have one), which in many cases exercises radically different code paths than what the User Interface (UI) would use causing you to measure the wrong thing and Visual test scripts constantly break during the product cycle. Because it drives the applications UI these scripts are vulnerable to failure due to changes in the product.

Lastly, the timers themselves were very low resolution and could not time relatively small actions such as refreshing a page in a document or recalculating the value of a cell in a spreadsheet.

We approached this problem in two ways. The first was to create a timer solution independent of the product used in driving the application. We developed a very low impact Software Developer Kit (SDK) tool set that allowed the application developers to actually insert markers in specified locations throughout their code.

The second portion was that we developed a tool that drove the application using the application UI itself. (I won't go into to much detail here except to say that this tool is complied with the application guaranteeing that scripts do not fail with the application. Again, this tool has an almost immeasurable impact on the shipping product.)

Real World

The "real world" test cases are an important part of ad hoc benchmarking. However, what we found was that by creating a clean and reproducible harness to run the scripts we could come up with many actionable items and the results are very measurable. The idea being a much larger set of real world anomalies caused by our application would be fixed implicitly through the clean action based test suite than if we concentrated on them alone.

Methodology

Cleaning up the hardware:

This was done in several phases. The first step was to get machines that would produce similar data on a given action. We did this by building the systems from the ground up. Every component had to be the same make, model, and even lot number. We found that any minor deviation no matter how small gave inherently different results. Even with identical systems, there was a certain error margin measured from one system to the next, which we resolved by testing the systems and grouping them accordingly.

We then used a series of tools to measure the processor, memory, video, and drive performance.

Each machine was given a second hard drive that would be used to isolate the test environment from the rest of the system. (see OS and application imaging)

We choose a processor that represented the lower end configuration of our standard users system, and then created a set of "tiers" based on operating system and memory configuration.

There are four primary tiers targeting to operating systems. For each OS, there is a low memory system and a memory unconstrained system. This allows us to distinguish disk bound problems from processor bound problems without the use of additional tools.

We also found that the network provided some level of overhead so we virtually eliminated that by moving to a fully switched network eliminating local corporate traffic.

Cleaning up the software:

When running benchmarks, the virtual environment needs to be as clean as the physical environment to minimize the impact on the data being gathered. To run through these tests we need an OS, a set of tools that drive the application, and additional tool(s) to extract the times.

Operating System: We set up the system in three phases; the first is to prepare the machines. Each machine has an operating system installed on the target hard disk. All memory resident programs that normally load on boot are removed as well as a handful of other checks. The OS is then defragged and stored as an image for future runs.

The second step is to prepare the systems for a suite or run for a given build. In this step the image is restored and the current build of your application is installed as well as all static tools needed for the run. This image is defragged again and the image is stored on the hard disk to be used for each test.

The last step is running the test itself. In this phase we format the drive and the clean prepared image is restored. Additionally, all the needed script files and documents are copied to the drive in a specified folder. Then a short cut to your driver is added to the startup group and we reboot into that drive triggering the start of the test.

Tools for driving the application: In the past we have tried manually testing the application and timing with a stopwatch, Visual Test (MSTest at the time), and Visual Basic. There are several problems with using these programs. Manually driving the application is extremely inconsistent. There are too many factors to try it this way, the least of

which is a .5 second standard deviation introduced by the tester themselves. Visual Basic has at least five megabytes of working set impact on your system, which heavily perturbs the environment. Another common problem is that if your application supports its own object model (OLE, COM) then you are usually exercising a different code path than the end user is when performing a specific function. And Visual Test, although it does exercise the UI as a user would, it often breaks due to changes in the UI, and carries approximately five megabytes worth of working set as well.

We addressed this by writing a tool the drives the applications through their user interface using a lookup table that is compiled with the applications. However that tool is not the focus of this paper. If you do not have the resources to write such a tool, we recommend you use a Visual Test type application to drive your test suite.

Performance SDK: ***Patent pending***
Once the environment was set up for consistency, precision becomes meaningful again. It is necessary to not only have very precise timers, but accurate as well. We developed and SDK to resolve this problem.

The tool set is comprised of the following components.

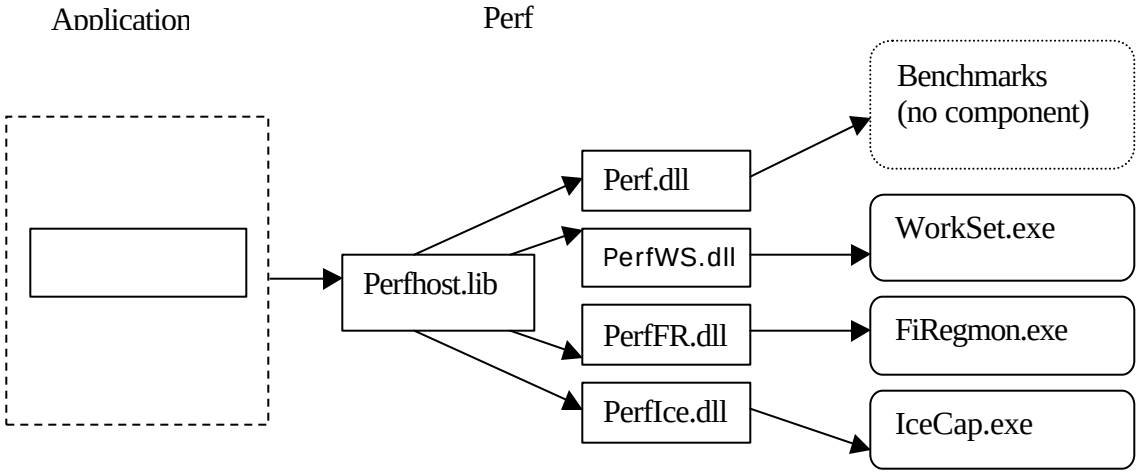
Perf.lib: The Perf.lib file is linked to any application using performance markers. This library exposes the API calls found in the Perf.dll. We preload the Perf.lib into memory before running the application to avoid adding the 4K (1 page) hit during boot.

Perf.dll: The Perf.lib contains all of the calls used in taking the time stamps and storing them in memory. It also contains all of the logic used in the Perf SDK and handles any special marker modification requests. This Dll, can be replaced with a comparable DLL that hooks into another tool you might have. For example we run tools that record registry read/writes, file i/o, memory usage, and others. Each of these tools has their own version of the Perf.dll and can be used to measure activity of any type between two markers.

Jumpstart/Seatbelt: These two tools work in tandem. The seatbelt tool is a very low profile tool whose job is to fire off the application and hand it a command line. In our case we run our driver tool. Seatbelt has a very small overhead of 20k, and nothing else is running in memory by the time it loads.

Jumpstart is a bit bigger, it has the logic to open an INI file and create the command line for seatbelt. Part of that command line is settle time for the OS. The standard time we wait before running a test is 10 seconds.

Although the OS has not completely settled by this point, the background noise should be at least consistent. Jumpstart then preloads the perf.lib so it is effectively cached, and drops from memory as it calls seatbelt with the created command line. Seatbelt counts the appropriate amount of time and calls the application directly. The Driver tool is loaded as part of the application in this case.



Reporting data

Additional Tools used in gathering data

DatParse: When all of the data has been gathered the program exits and the data is written to a log file from memory. In the automation system, we then push that log file to a database where it is parsed via a series of stored procedures into the appropriate tables. However we also created a tool for developers to use that allows them to parse the raw data into legible marker sets and view the deltas between those markers.

Code Markers: The code markers or triggers are API calls put in the applications at key locations. For example if you wanted to measure "open a document" time, a developer would put a code marker at the File Open start location in the code, and a second marker at the file open stop location in the code. Note Markers pairs are not defined explicitly in the code itself. That is done post processing of the data, so any marker can be pared up with another.

Log Spinner: This tool does exactly what it sounds like it does. It collects data log files, parses them and feeds the to the database using a series of stored procedures.

Performance database repository: Although this is a key piece used in reporting the data after it has been gathered, the schema details are outside the scope of this paper.

When the data is pushed to the database we store it as both raw data that can be reprocessed at another time and we also store deltas of specific marker pairs. There is a good deal of communication between the automation system and the data repository because of the complexity and the volume of the data stored. For example we may know that we want to measure "file open". In this case we create a "marker pair" in the database and associate the two relevant markers with that id. When those markers are detected in a test, the data is stored with the marker pair. However, this is not enough to uniquely identify what was done. We do not know what type of document was opened, and if we want to differentiate between actions of the same type, but with different objects, we need a second level of distinction. We solved this by creating a "actions table". Actions associate a Marker pair with a given script.

There is also a preprocessing way of differentiating marker pairs of the same type. This is called marker morphing and is built into the Perf.dll. If your application driver loads the Perf.dll it can tell the dll to look for specific markers and change their ID to an alternate one if triggered. You can also toggle between ignoring all markers and recognizing all markers.

Validating data: Once the data has been gathered it can be viewed dynamically, however we always take a first pass at the numbers before releasing it. Validating the results can prevent a lot of wasted dev cycles if there was a problem with the environment.

Analysis: Part of validating the data and turning it into actionable items is the analysis. We do this statistically, but it requires more in depth knowledge of the different types of number we gather.

In almost all cases we take a minimum of six iterations for each action ID. There are three types of iterations the Perf SDK supports.

OS Reboot: In this style we reboot the application between each snap shot of the action. Because the state of the disk and memory are restored to the same state with each reboot, the consistency is very high. These are some of the most reliable data types we can gather, and they are also the most expensive in terms of time.

Application Reboot: In this case the application is shut down and restarted. Although the memory is still fragmented and much of the code is cached via the operating system, aside from the first number the iterations are fairly similar. Since it does not require a reboot, it takes a lot

less time to gather the data; unfortunately it also produces results with a high standard deviation between iterations.

Cached: In this case the application is not exited until the test is complete. The action being timed is contained within a loop and all of the data is retrieved at the same time. This is obviously the fastest way to gather the data, however it also is the least consistent, and most unlike a real user situation. Generally this type of test is not recommended unless you are trying to time a very Central Processing Unit (CPU) intensive, low memory usage action.

Exposing the Data: When looking at this data we stack up the iterations and derive three numbers.

The Average: Sum of all iterations multiplied by the total number of iterations.

Standard Deviation: This number represents the what your next data point should be with a +/- error margin. We use this to determine the volatility of the data which also gives us an idea of how reproducible the results are.

Percent difference of the average the standard deviation represents: Since the Standard deviation is relative to the average, it requires more than just a glance to understand it relevance. If the number is greater than 5% then our data is too erratic. We know that next time we run our results we can expect up to a 5% fluctuation to occur from our last data point without a real change in performance.

Percent Diff: This number is used to tell us how much performance has increased or decreased from one build to the next.

(Example)

	Baseline	**Current Build**
Iteration 1	1.336	2.43
Iteration 2	1.459	2.431
Iteration 3	1.339	2.564
Iteration 4	1.375	2.789
Iteration 5	1.421	2.549
Iteration 6	1.299	2.544
Average	**1.372**	**2.551**
StDev	**0.059**	**0.131**
%StDev	**4%**	**5%**

Open a HTML Document		
Baseline	Current Build	%Diff
1.372	2.551	-46%

In this case we experianced a degradation of 46%. We feel comfortable that the data is actionable because it is relativley consistant from one iteration to the next.

We are not sure that the application change is responsible for the degradation, but we do know that what we are measuring is really this different.

Identifying the problem:

The percent Difference is the primary indicator that we use to determine if a anomaly in the code has occurred. The first step in investigating that anomaly is to check the environment and the percent standard deviation. If there are no global problems, and the statistical data seems within an acceptable range we can use additional tools to expose what caused the time difference and verify it is the code itself that is the problem.

By producing logs that show the Registry, memory, file, processor usage, and total time spent in functions between two markers we can isolate the change that has caused the problem.

QW2000 Paper 4W2

Mr. Steven Rabin
(Interworld Corp.)

eCommerce Performance Management Lifecycle -- Benchmarking, Methodology and Criteria

BACK TO QW2000 PROGRAM

Presentation Abstract

One of the key aspects of a successful eCommerce initiative is performance. While this is often overlooked or taken for granted very few customers will return to a site that exhibits sluggish performance. Features, functions, robust tools, etc. are all important to the success of an eCom site but in the long run don't mean much if customers don't get efficient responses from the site their shopping. This session discusses the eCommerce Performance Management Lifecycle. This includes the key elements in understanding customer/site dynamics and how to use this information to design an eCom based performance testing benchmark (including the methodology that goes along with it). This includes workload characteristics, transactional definitions and site assumptions. A framework for eCom performance benchmarking and the criteria that need be measured is also introduced. Real world scenarios from a variety of sites are analyzed from the lessons learned perspective.

The life cycle takes into account the 8 key elements of delivering and maintaining a robust eCommerce environment. This includes understanding customer behavior, setting performance goals, benchmarking, tuning and adjusting for peak periods. The eCommerce Benchmark must be designed to measure the key processes utilized during the customer shopping experience (B2B and/or B2C). Industry experience and customer input must be incorporated into the benchmark's design to represent the typical workload placed on a Commerce server. Based on the results organizations can predict how the site will perform under tough, real-world conditions. For example, the benchmark must consider and measure the following granular activities:

1. authenticating an existing user
2. registering a new user or changing the profile of an existing user
3. browsing a catalog by category and/or product
4. processing an order
5. accessing key site pages
6. searching for products, full text and parametric
7. adding, changing, deleting items from a shopping basket (or purchase order)
8. purchasing/processing a committed order.

These activities must be incorporated into a series of transactions that measure round trip

performance, execution under load and scalability. The benchmark must be designed to measure transactional workloads that exercise the key components utilized during an eCom session. This includes:

* concurrent browser sessions
* dynamic page generation with database access and update
* processing multiple web objects
* on-line transaction execution scenarios
* multiple databases consisting of complex relationships
* efficiently resolving resource contention

Consideration must be given to capturing and interpreting the collected data. Translating these metrics into meaningful statistics provides the information required to make key decisions. The following are examples of what need be measured and reported:

1. orders per minute
2. registrations per minute
3. complex browsing per minute
4. simple browsing per minute
5. searches per minute
6. transactions per minute
7. dynamic vs. static pages served per minute

Finally, software and hardware environmental factors must be taken into account to insure the measured results are meaningful. Operating System, Web Server, Database Server, SSL (Secure Socket Layer) are software examples. RAM, CPU, disk access speeds and HTTP routing are hardware examples.

About the Author

Steven Rabin is Chief Technology Officer of InterWorld Corp., a New York firm offering a variety of electronic commerce application technology solutions.

He is a certified CDP and has published articles in Computer Language, Data Based Advisor, and Systems Development as well as authoring the "Mission Critical Views" column in *Application Development Trends*.

BACK TO QW2000 PROGRAM

eCommerce Performance Management LifeCycle

Steven Rabin,
Chief Technologist

INTERWORLD

How Much of Your eCommerce Sales are at Risk?

- 44.1 million on-line shoppers in the U.S.
- 37.5 million more will go on-line in the next 12 months
- on-line buyers spend an average of $200/month
- an estimated $762 million/month in eCommerce sales is lost due to slow, spotty, inefficient sites

Zona Research, 7/99

INTERWORLD

Commerce Server Site Analysis

Home Page Size	Bail-Out Rates
70 KB *	50%
40 KB *	30%
34 KB	8%

* other pages on site were 32-35KB and showed a 6-8% bail-out rate

vendors must heed the 8 second rule

INTERWORLD

Keynote Systems, 11/99

Actions Taken After Abandoning eCommerce Site

Demographic	Did Not Buy Item	Bought Item at Other Web Site	Bought Item at Brick and Mortar
All	34%	24%	37%
Age < 25 years	27%	23%	53%
Age 25 to 34	43%	25%	30%
Age 35 to 44	27%	24%	43%
Age 45 to 54	35%	21%	38%

Zona Research, 9/99

INTERWORLD

Top Commerce Site Performance (Xmas 99)

Site	HomePage Delivery (seconds)	Availability (percent)
Amazon	4.9	99.4
Barnes & Noble	7.3	99.2
Dell	8.3	99.1
Gateway	6.4	86.7
Lands End	9.0	91.5
Toys R Us	5.7	98.1
Business 40 index	9.2	97.5

INTERWORLD

The eCommerce Performance Life-Cycle

PLAN

- Define your terms
- Understand your customers
- Set your goals
- Plan your infrastructure

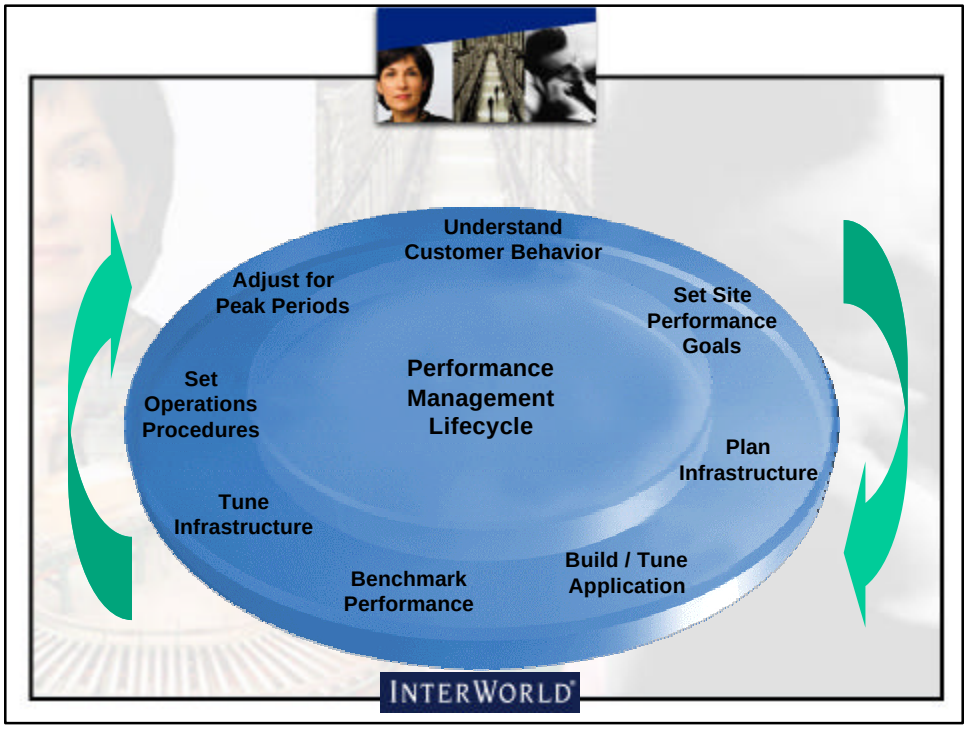
BUILD

- Build and tune your application
- Benchmark your system

GO

- Build operations procedures
- Adjust for special events

INTERWORLD



Define Your Terms

- Simultaneous users:
  - Orders / day? , Pages / second?, Visitors / day?
- What is the difference between a page?
  - Hit?, Impression?
- How complex are the pages?
  - Frames?, GIFs?, Applets?
- Are system capacity numbers measured in:
  - pages / machine?, pages / CPU?
- Understand units of commerce work
  - loading modules
  - building templates
  - round trips (ie. database)
  - transaction types

INTERWORLD

Understand Your Customers

- You can't predict your traffic unless you understand your customers.
- You must predict how many will come and what they will do.
- Rely on past experience, industry statistics, competitors, and all other information available.
- Consider season, time of day, marketing campaigns, and other factors.

INTERWORLD

Set your goals
Sizing the System by Orders / Day

Orders / Day	1000	[GOAL]
Conversion Ratio	2%	[ESTIMATE]
Pages / Visit	10	[ESTIMATE]
Peak Ratio	10%	[ESTIMATE]
Pages / Peak Hour	50000	[CALCULATED]
Pages / Second	14	[CALCULATED]
Pages / Second / CPU	2	[IW ESTIMATE]
CPU's Needed	7	[CALCULATED]

INTERWORLD

Set your goals
Sizing the System by Simultaneous Users

• Simultaneous Shoppers	1000	[GOAL]
• Desired Response Time	5	[ESTIMATE]
• Dwell Time	45	[ESTIMATE]
• Pages / Second	20	[CALCULATED]
• Pages / Second / CPU	2	[IW ESTIMATE]
• CPUs Needed	10	[CALCULATED]

INTERWORLD

Build and Tune Your Application

- Budget time for performance tuning
- Understand the trade-off between performance and functionality
- Consider impacts on other systems
- Focus on areas that will be the hardest hit by customers.
- Small decisions can have a huge impact

INTERWORLD

Benchmark Your System

- Benchmark each component in isolation.
- Benchmark the system as a whole.
- Invest the time in accurately modeling your users with a load testing tool.
- Assemble a team of experts to help with the final acceptance test.
- Measure the system against the original goals.

INTERWORLD

Performance & Stability Testing (1/2)

– Simultaneous Users:

Threaded commerce software guards against serialized resources. However, that requires careful management of common services to avoid deadlock. Test for concurrent usage.

- database connection pools
- CPU
- components/objects

– Transactions per second:

Stress testing for peak performance. Turnaway capabilities based on estimated peak availability.

- common HTML refusal page
- where is the knee of the curve

INTERWORLD

Performance & Stability Testing (2/2)

- Server Sizing:
  - Memory/CPUs?
  - What is running on each server?
- Database Sizing:
  - Capacity planning using sample data.
  - Foot-print per user.
  - Trigger threshold notices.
- Real-world Simulations:
  - Server log analysis
  - Classify types of transactions:
    - Example: Search versus Orders?

INTERWORLD

Benchmark Configurations

Scenario 1:	Search to Buy Ratio 99 : 1	Static to Dynamic Pages 1 : 99 5 : 95 10 : 90
Scenario 2:	Search to Buy Ratio 95 : 5	Static to Dynamic Pages 1 : 99 5 : 95 10 : 90
Scenario 3:	Search to Browse to Buy Ratio 60 : 39 : 1	Static to Dynamic Pages 1 : 99 5 : 95 10 : 90
Scenario 4:	Search to Browse to Buy Ratio 50 : 45 : 5	Static to Dynamic Pages 1 : 99 5 : 95 10 : 90

INTERWORLD

eCommerce Benchmark Transactions

- Home Page Transaction
- Product Page Transaction
- Section List Transaction
- Shopping Basket Transaction
- Buy Transaction
- User Registration Transaction
- Search Transaction
- Check Status Transaction

Static and Dynamic Units of Work - Simple and Complex Transactions

INTERWORLD

Transaction Type Percent Frequency (bookseller)

- Search on product characteristics (e.g. title 43%)
- Display particular product 32%
- View Homepage 10%
- Display Shopping Cart Contents 8%
- Add Item to Shopping Cart 5%
- Buy a product 1%
- Register new user .3%
- Display order .3%
- Error .3%

INTERWORLD

Benchmark Scripts
(Search-Buy)

Home Page
Search
Basket
Order Process

OP1 =	ObjBuilder, OrderProcess	initialize
OP2 =	" "	verifyOrderDialog
OP3 =	" "	verifyBillingDialog
OP4 =	" "	verifyShippingDialog
OP5 =	" "	setShippingMethodDialog
OP6 =	" "	confirmationDialog
OP7 =	" "	paymentMethodDialog
OP8 =	" "	orderCompleteDialog
OP9 =	" "	deleteOrder

INTERWORLD

Web Site Performance Q&A

- Do customers experience consistently good performance from your web site?
- How does your web site compare to your competitors or to industry benchmarks?
- Do customers in certain cities have more trouble with performance than from other cities?
- How does your site deal with heavy traffic and/or peak load periods?
- How reliable is your site in terms of connections refused, connection time outs and page time outs?
- Are certain pages consistently slow?
- Should geographic mirroring be considered for your site?

INTERWORLD

Benchmark and Performance Technology Assessment

- **Site Characteristics**
  - Determine customer value of each feature
  - Remove non-essential steps
  - Dynamic page analysis
  - Turn off features (based on peak load scenarios)
  - Remove or disable personalized pages (peak load scenarios)
  - load characterization, site characterization
- **Assessment**
  - Determine customer expectations and evaluate the impact
  - Who are you selling to and what do they expect to see next?
  - Does the current infrastructure support the future?
  - Cost and benefit analysis of new features vs. performance
- **Simulate Real World Load**
  - Number of users
  - Types of transactions
  - Database access
  - Legacy connection(s)
  - Network/infrastructure issues

INTERWORLD®

Operating System: Solaris
Number of CPU's: 2
Memory: 1.4 GB RAM

Element	1 user load	5 user load	10 user load	20 user load	50 user load
Home Page					
CatalogList					
SectionList - Cached					
CatProduct - Cached					
SectionPage - Cached					
ProductPage - Cached					
Basket (IWX)					
BasketSteps					
OrderProcess (totals)					
- Initial Step					
- Verify Order					
- Verify Billing					
- Verify Shipping					
- Shipping Method					
- Confirmation					
- Payment Method					
- Order Complete					
- Delete Order					
ObjectBuilder					
- NOOP process					
- Static Template					
- Static Template + Database access					
Search					
Category Search					
Customer Registration					
Mix 85% to 15%					
Mix 95% to 5%					

Results vary based on site customizations

Use benchmarks as a guide. Your mileage may vary!

Units = Dynamic Pages/sec (transactions/sec)

INTERWORLD®

eCommerce Benchmarking Methodology
and Criteria
INTERWORLD

QW2000 Keynote 5P1

Mr. Leon Osterweil
(Unviersity of Massachusetts)

Determining the Quality of Electronic Commerce Processes

BACK TO QW2000 PROGRAM

Presentation Abstract

The goal of this talk is to emphasize that a discipline of assuring the quality of electronic enterprise systems and transactions can be built on top of the very large base of technologies that have been developed for more traditional application software.

The talk suggests that internet transactions and enterprises should be treated as technologically based objects that we call processes. The talk further suggests that it is feasible and beneficial to consider such processes to be software.

Viewing these processes as software immediately suggests that such software engineering approaches as phased development, computer-based execution, and rigorous testing and analysis are all applicable, and that the vehicles we have used to increase our confidence in the quality of application software should all be applicable to increase our confidence in the quality of processes. This talk will motivate the above intuitions, and will then demonstrate how to make them tangible. We provide as a specific example the application of dataflow analysis to the verification of critical properties of auctions.

About the Author

LEON J. OSTERWEIL is currently a professor in the Department of Computer Science at the University of Massachusetts, Amherst. Previously he had been a Professor in, and Chair of, Computer Science Departments at both the University of California, Irvine, and University of Colorado, Boulder.

He was the founding Director of the Irvine Research Unit in Software (IRUS) and the Southern California SPIN. He has been Program Committee Chair of ICSE 16, TAV 2, ISPW4, and SDE2, and General Chair of FSE 6. He has also presented keynote talks at such meetings as CASE 92 in Montreal, Quality Week 96 in San Francisco, the Inaugural Symposium of JAIST (the Japan Advanced Institute for Software Technology) in Kanazawa, Japan, and ICSE 9 (the Ninth International Conference on Software Engineering) where he introduced the concept of Process Programming.

His ICSE 9 paper has been awarded a prize as the most influential paper of ICSE 9, awarded as a 10-year retrospective. He has consulted for such organizations as IBM, Bell Laboratories, SAIC, MCC, and TRW, and SEI's Process Program Advisory Board. Prof. Osterweil is a Fellow of the Association

for Computing Machinery.

BACK TO QW2000 PROGRAM

Determining the Quality of Electronic Commerce Processes

Leon J. Osterweil, Co-Director
Interdisciplinary Center for Electronic Enterprise
University of Massachusetts
Amherst, MA 01002
USA

ABSTRACT

As increasing numbers and types of enterprises and transactions in our society continue to migrate towards the internet, the quality of these transactions and enterprises becomes increasingly important. It is now feasible for banks and brokerages to hemorrhage money at microsecond speeds, for government agencies to corrupt millions of vital records in seconds, and for businesses to compromise secrets and intellectual property (both their own and those of others) without even knowing it. Certainly there is wonderful promise in the internet's incredibly fast and easy access to unimaginably rich troves of information and human resources. But with this immense power also must come sobering realizations about the vulnerabilities that such power creates, and the need for assurances that this power will not become a dangerous instrument of harm and evil.

We suggest here that the transactions and enterprises that are increasingly being hosted on the internet should be treated as technologically based objects that we call processes. We further suggest that the base of process technology that has been built by the software engineering community can be used effectively to assure the quality of these transactions and enterprises. In earlier work we have demonstrated that it is feasible and beneficial to consider such processes to be software. Viewing these processes as software immediately suggests that such software engineering approaches as phased development, computer-based execution, and rigorous testing and analysis are all applicable. Indeed this analogy suggests that the vehicles that we have used to increase our confidence in the reliability of application software should all be applicable to increase our confidence in the reliability of the processes that we are implementing as internet-based transactions and enterprises. This talk will motivate the above intuitions, and will then demonstrate how to make them tangible and how to realize the benefits of this point of view.

A major focus of this work is the need for languages that can be used as the basis for achieving demonstrable internet process quality. A key tenet of the talk is that a suitably powerful language is needed to define the full complexity of processes, and that such a language can then be used as the basis of establishing various quality attributes of such processes. Once these quality attributes have been established, that same language can then be used as the basis for supervising the execution of processes that are then known to have these quality attributes. We demonstrate the need for this process definition language to incorporate such features as the specification of reactive control, resource

utilization, artifact flow and structure, and realtime constraints. We illustrate this with the example of the Little-JIL language that we have developed.

Another key focus of this work is the consideration of approaches that can be effective in assuring quality in such processes. Dynamic testing is widely used to ascertain application software quality, and an analogous approach, namely realtime process monitoring, can be useful in establishing the quality of processes. We note that monitoring statistics and summaries are more precise and definitive when they are gathered by instrumentation of an explicitly defined process, such as results from definition in a language such as Little-JIL. But, just as in the case of application software, dynamic testing has serious limitations, especially when applied to concurrent systems. Thus, since most enterprise processes are concurrent, it becomes particularly important to consider the application of complementary approaches to quality determination. In this talk we emphasize the use of static analysis as such a complementary approach. We explain the applicability of such static analyzers as dataflow analyzers in establishing the freedom of processes from certain serious types of faults such as deadlocks and race conditions. Accordingly, we next explain that the Little-JIL language has a rigorous semantic definition that can then be used as the basis for definitive reasoning about the quality of processes defined in the language.

The net effect of defining processes using a rigorously defined language such as Little-JIL is that quality characteristics of the processes can be proven using automatic analyzers, and the processes can then be automatically translated into systems running under control of an interpreter that guarantees adherence to these characteristics. As Little-JIL is a very powerful and comprehensive language, the processes at issue here can be very realistic electronic commerce processes.

In the talk we will use as an example a family of processes for describing and controlling auctions. Our example will demonstrate the use of Little-JIL to define precisely a small set of auctions that differ from each other in seemingly small, but nevertheless important, ways. The different auctions have different runtime behaviors, and some of them have faults that are not immediately obvious. We will then demonstrate how these different auctions can be statically analyzed. We indicate how we can use FLAVERS, a dataflow analyzer that was developed for the analysis of complex concurrent software systems to analyze these auction processes. We demonstrate how to analyze for the presence of undesirable race conditions, and to assure such properties as not accepting late bids, and assuring that the high bid always wins. Once these properties have been established, it is safe to assume that the auctions so defined will actually execute without these flaws once the auctions have been translated to be run by the Little-JIL runtime execution system.

The goal of this talk is to emphasize the point that a discipline of assuring the quality of electronic enterprise systems and transactions can be built on top of the very large base of technologies that have been developed for more traditional application software.

QW2000 Keynote 5P2

Mr. Rainer Pirker
(IBM / Austria)

The Need for Quality -- e-business Performance Testing

BACK TO QW2000 PROGRAM

Key Points

- How do customers perceive performance
- Measuring performance
- Using the right tools
- Getting advantages from an independent test team
- Answering security issues

Presentation Abstract

Customers expect the best in e-business. This includes, above all, the first contact they make with your organization. The first impression is key, the user interface, the performance, the right response, right away. Get ready for interfacing with your customers at first click or call - only one chance to attract the customer with good usability, fast and reliable response. One click, one chance, period.

The time and quality of response and the perceived performance to customers is essential in e-business. With e-business solutions you win with an innovate and high quality solution but you loose immediately customer confidence if your system breaks down with complex user interfaces, or long response times - so customers will cancel their transactions which drag the performance of your e-business solution further down. Your customer care center gets flooded with angry customer calls or e-mails.

We'll help you to get the best out of your system - for your customers' benefits. We show you the critical success factors and the measurements which allow to decide what good quality means for e-business testing. Learn about independent test teams working in different locations, with the help of leading edge tools. We also take a look at security issues in e-business testing.

About the Author

Rainer Pirker is a consultant with IBM Application Development Effectiveness Practice. He specializes in test concepts and strategies to improve software quality. He has more than 10 years of experience in information system and application development, and has in-depth knowledge of banking, insurance and public institutions in the fields of executing, developing and improving tests.

Software Quality Week 2000 - San Francisco

The Need for Quality - e-business Performance Testing

Rainer Pirker / Andreas Rudolf
IT Consultants
IBM Global Services Austria

Copyright by IBM 2000

Presentation overview

- **Our e-business experience**
- **Why focus on quality?**
- **Key quality aspects for e-business solutions**
- **Complexity of e-business solutions**
- **Case Study:**
  - e-billing project / Telecom
  - test strategy
  - test concept
  - test cases
  - skills needed
  - test environment and tools
  - current project status
- **Critical success factors**
- **Any questions ...**

Copyright by IBM 2000

We have had experience of conducting large scale testing projects in many different industries

Our Department is strongly focused on:

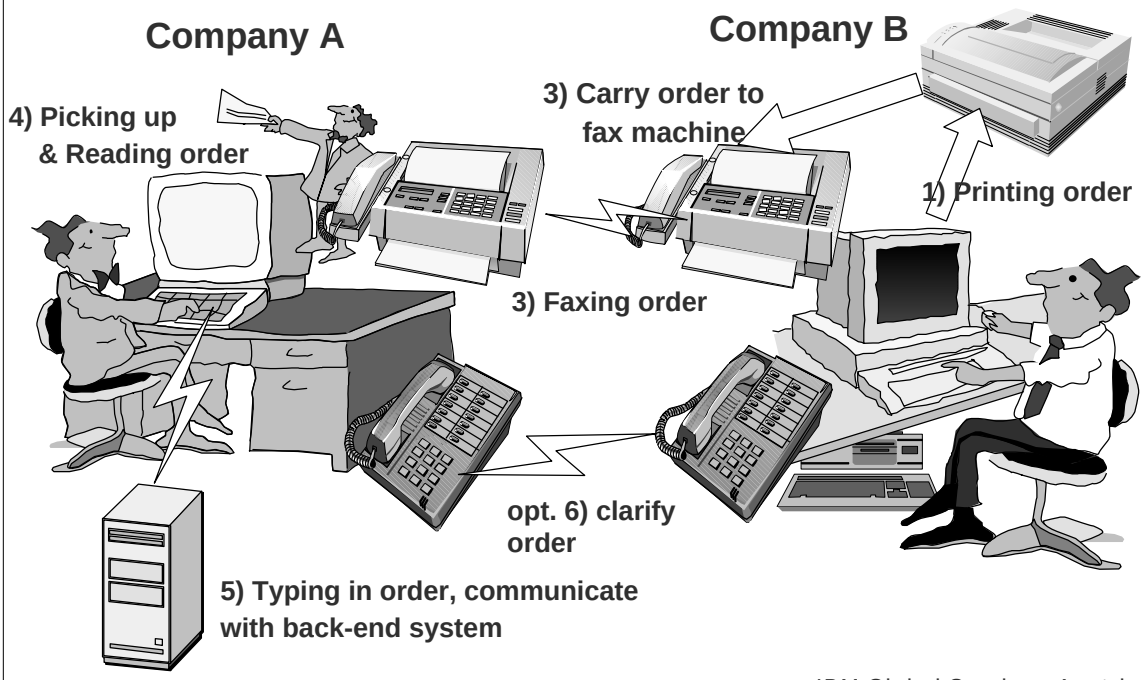
- Test consulting (e-business, EURO, Y2k, new product developments etc.)
- Application Development Effectiveness consulting (supporting the customer to assess and improve their application development processes)

Our working experience is based on:

- a dozen large e-business/Y2k/EURO test projects since 1996 where we worked as consultants and as test managers
- dozens of projects for marketing and workshops for customer
- projects in many industries: finance, insurance, transport, utilities

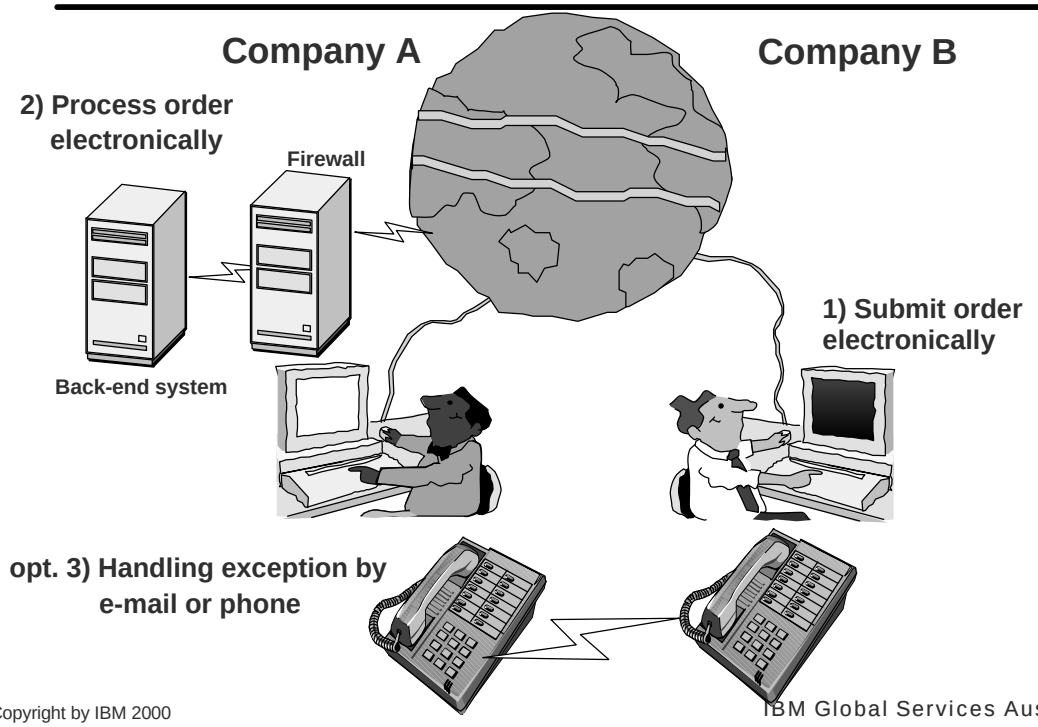
Copyright by IBM 2000

The traditional order placing and order receiving process is time consuming and error prone



Copyright by IBM 2000

In the e-business world the order process is fast but security and performance issues have to be addressed



Copyright by IBM 2000

Only if you address the e-business risks from the beginning is a high return on investment in sight

Traditional solution:

- (+) creative, experienced human being, good exception handling
- (-) expensive workforce, processing is mainly done by humans
- (-) medium data quality, frequent 'media breaks'
- (+) "Human" firewall, resistant against "hacking"
- (-) service only during business hours

e-business solution:

- (RISK) single path, weak exception handling without human support
- (+) Processing is done by computers
- (+) high data quality and no media breaks (single electronic process)
- (RISK) hacking and spoofing are possible without a good security solution
- (RISK) customer expectations are 24h x 7 days service

To target these e-business risks, testing is key!

Copyright by IBM 2000

Key quality aspects for e-business solutions are: scaleability/performance, security, usability and reliability

Copyright by IBM 2000

Key is how users perceive the performance of your site and service

Scalability and Performance:

- Permit a maximum of 8 seconds for loading a web page. When the loading time increases just 10% the bailout rate goes from 6% to 40%
- Knowing your requirements is key (number of concurrent users, user profiles, desired response time, throughput, type of sessions, transaction mix, modem speed etc.)
- Focus on the user's perception of performance:
  - loading the whole java applet vs. loading JIT the classes needed for each transaction
  - loading large HTML tables vs. loading smaller ones as required by the user
  - the user will accept longer download times when the site and form are perceived as secure

Copyright by IBM 2000

Taking care of your customer's security fears is a further key to success

- **User security fears**
  - Security perception is key!
  - Good: inform the user of your security solution (SSL, length of key used for encryption etc.)
  - Bad: your site's security problems are made public

Only a customer who trusts your e-business will continue to do business with you!

- **Company security**
  - passive Web attacks: wire tapping of password and data, analysis of the structure of the data stream
  - active Web attacks: spoof, mutilate and delete data, password guessing
  - Use a secure system architecture (i.e. firewalls)
  - For sensitive data (credit card numbers) use a secure protocol like SSL, a secure protocol like SSL can have a big performance impact if not handled by cypto hardware
  - Do not allow access your companies internal functions. Do not test call center interfaces over the internet

Copyright by IBM 2000

Usability is also key as users need to quickly and easily find the information they seek

Usability aspects:

- **Navigation:** usability is improved when the site supports a search facility, a site map and buttons for navigating to and from each page.
  The site should never contain any broken links.
- **User interactions:** the user's ability to interact with the system must be kept very simple and easy i.e. "one click" shopping.
- **Help messages**: especially business to customer (B2C) users cannot be trained in the normal way. Extensive help messages and FAQs are useful
- **Computer literacy**: users find it important to be able to quickly locate the information they require on a Web page - graphical design is not the crucial point for most e-business users.
- **Printing**: users tend to print Web pages. Reading a page on the screen is about 25% slower than reading a hardcopy of the same text
- **Support channel**: however good your usability is, strong e-mail and telephone support remains a necessity

Copyright by IBM 2000

24x7 operation and the high visibility of e-business solutions drives reliability requirements ever higher

Reliability:

- B2C: customers from across the world expect 24h x 7days operations which leaves little time for even short service windows
- Customers expect short down times and high mean time between failure rates (MTBF)
- Outages have a serious business impact:
  American Express outage cost: 167,000 USD / min
  Charles Schwab outage cost: 1M USD / min

Copyright by IBM 2000

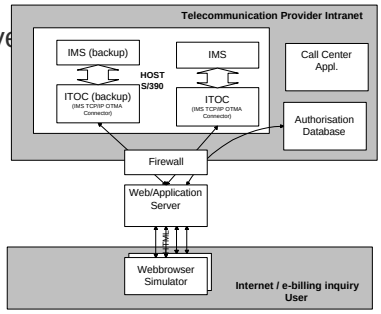
The complexity of e-business architectures is such that end to end performance tests are definitely needed

There are two basic types of e-business solution:

- new e-business solutions build new from ground up
  - Risk: high -> completely new system
- Web enablement solutions of existing legacy applications (e-legacy)
  - Risk: medium: -> interface to existing production systems

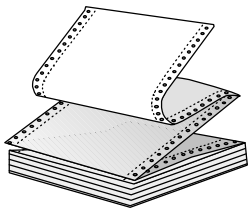
The structure of an e-business solution is:

- a three-tiered architecture similar to Client/Serve
  PLUS:
  - firewalls
  - distributed objects
  - authentication, credit card verification
  - ISPs
  - etc.



Copyright by IBM 2000

Our case study is based on a telecommunication provider's first e-business project



The customer

- Telecommunication company
- 5M customers
- Their first e-business project

The project

- e-billing solution which enables customers to obtain their telecommunication bill via the internet
- pilot targeted 500 customers

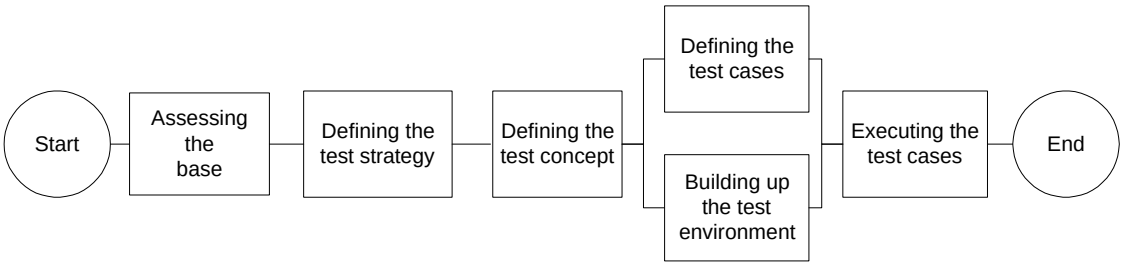
IBM

- A request to independently validate the solution was not received until after the pilot implementation had been completed

Copyright by IBM 2000

For rapid e-business testing a clearly defined approach is the foundation for success

e-business testing approach



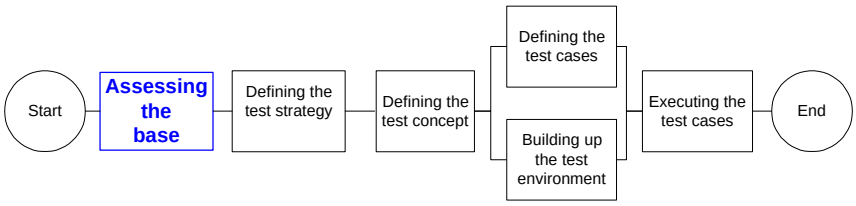
This approach has been proven in many successful projects - at this level there is nothing e-business specific in there

Copyright by IBM 2000

First we assessed the existing base in terms of test cases, documentation, environment and tools

Assessing the base situation:

- Test cases: there were none documented
- User documentation: an user handbook was not available however, the functional specification was in good shape
- Requirements, design and product documentation: The functional specifications were sufficient however, the non-functional requirements, like performance, transaction rate, reliability etc. were missing
- Test environment: we agreed with the customer that an isolated test environment should be built. Only the mainframe connection was to the production environment.

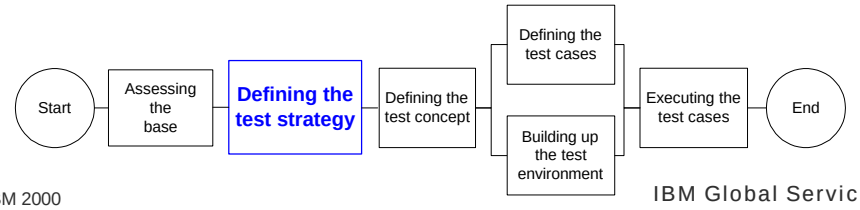


Copyright by IBM 2000

Then we defined the test strategy, the heart of any testing project

Defining the test strategy

- Draw a system structure diagram - a diagram of the architecture was not available so we draw the picture to obtain an insight into possible performance bottlenecks and security implications
- Fast start: a very tight time schedule of about 4 weeks forced us to do a really fast start in order to finish the engagement in time
- Performance tests first: due to the customer's priorities we had to swap around the traditional approach and do the performance testing before functional testing
- Black box end to end test: the customer was interested in the end to end results and how the end users perceived the system's performance



Copyright by IBM 2000

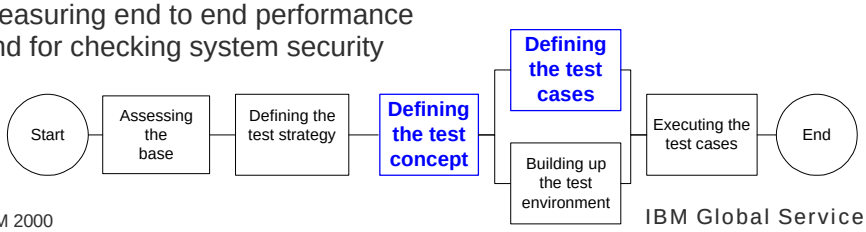
In the test concept we focus on What and How

Defining the test concept

- Define the What and How
- Highlight 1: Check functional and non functional requirements - for the functional requirements the application and it's specifications were available and sufficient. Non functional requirements like performance, response time, transaction mix, reliability etc. were only partially available and had to be detailed out by us.
- Highlight 2: Define test cases for measuring end to end performance and for checking system security

Defining the test cases

- The test cases were for business transactions which we found documented in the functional specifications and by exploring the live application
- Headlines for the test cases were documented in MS Excel

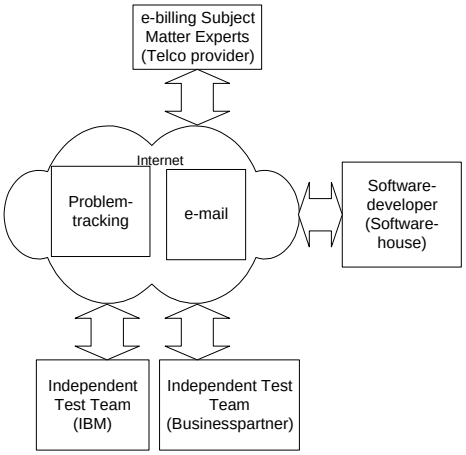


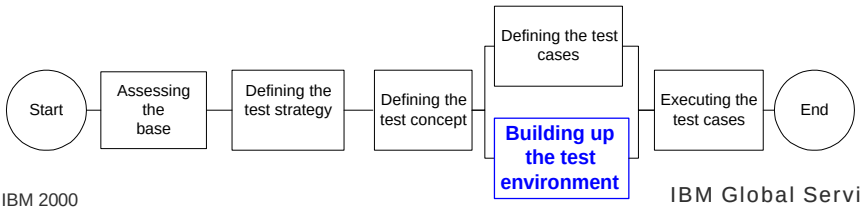
Copyright by IBM 2000

The test environment has to be isolated from the production environment to get reproducible results

Building the test environment

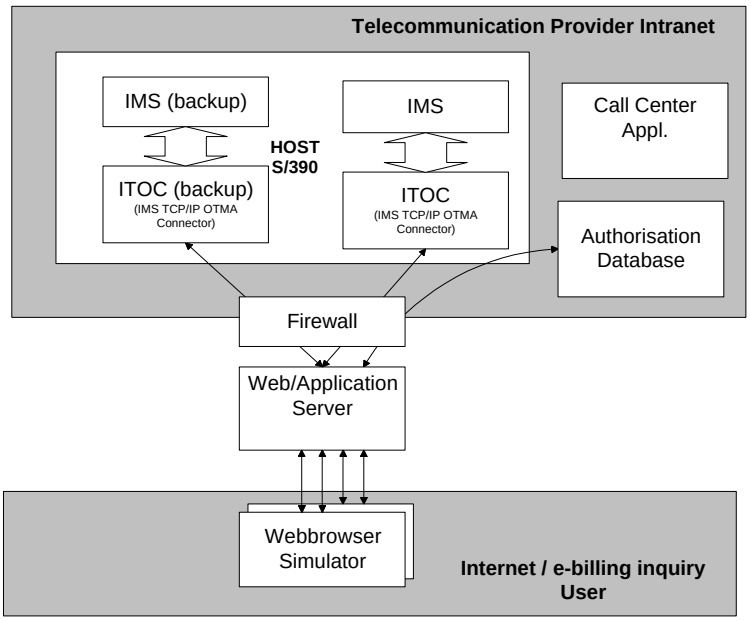
- Test environment: a dedicated and isolated test environment was built quite separate from production except for the connection to the mainframe
- For defect tracking we used a Web based tool as the developers, customers and testers all worked at different locations.





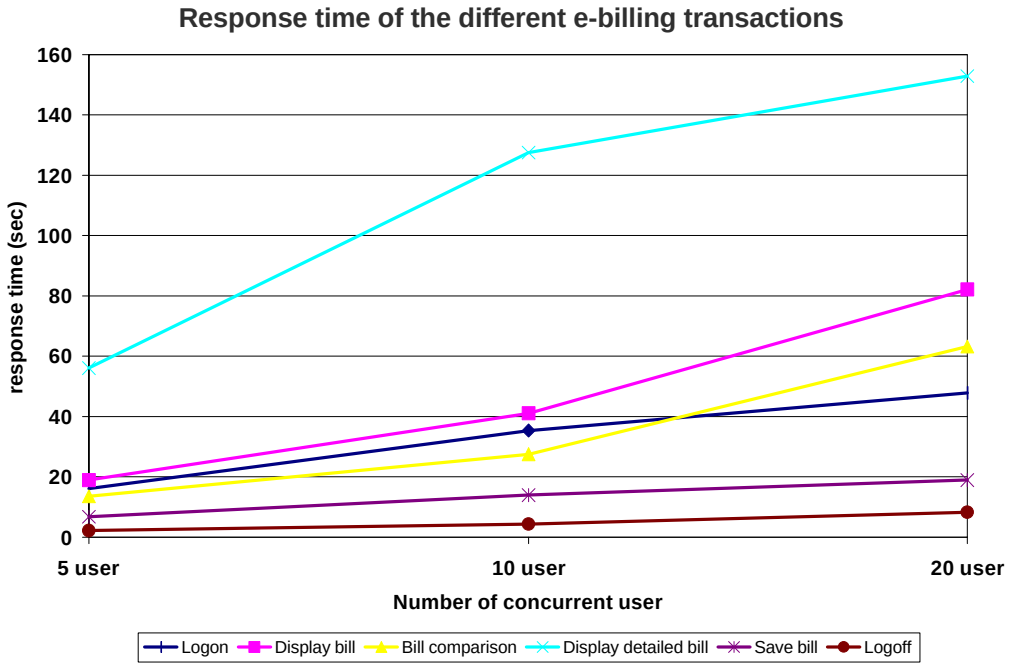
Copyright by IBM 2000

In the case study the test system was a very close copy of the production system



Copyright by IBM 2000

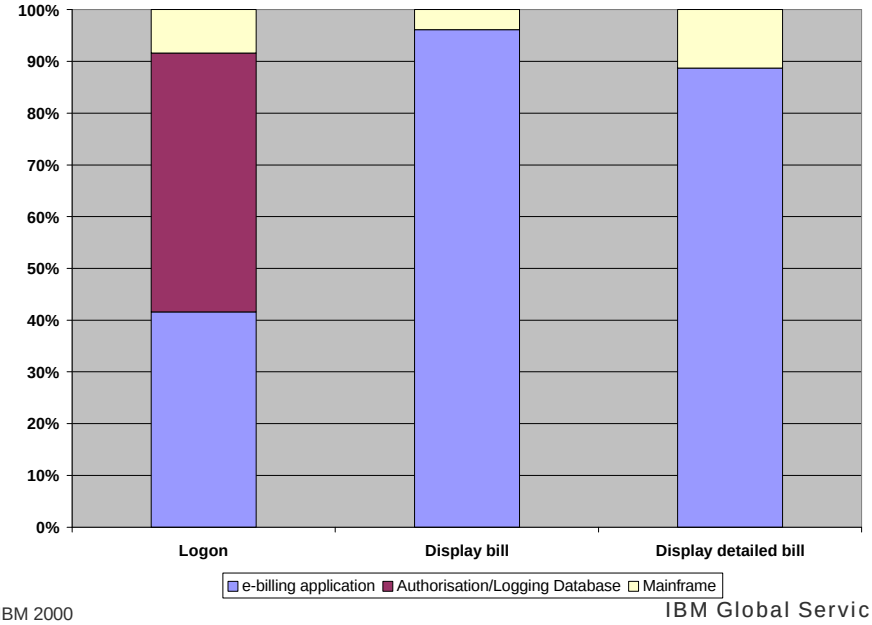
Response times were measured for different user profiles and different numbers of concurrent users



Copyright by IBM 2000

End to end measurements identified the application server (data conversion) as the performance bottleneck

Response time distribution of different e-billing transactions measured at the application server



Copyright by IBM 2000

Within four weeks the test was completed and the final report given to the customer

During this time the independent test team accomplished all of the planned tasks:

- Test strategy defined
- Test plan defined
- End to End Performance Test cases executed
- Metrics and final report delivered

Copyright by IBM 2000

Quality testing had shown up a very large gap between actual performance and the requirements

- Metrics and reports delivered
  - Response times per transaction type and the number of concurrent users
  - Response times per transaction type for different user profiles - private customers, small- , medium businesses
  - To obtain a measure for user perceived performance, we measured the time before the first HTTP data packet arrived. This was interesting for transactions such as bill display were a large amount of data was being transferred. The user himself can see the first page of the bill before the download has finished. This depends on the HTML data stream.
- The following parameters were measured on the web application server after it was identified early as the performance critical system in the architecture:
  - CPU Load
  - Virtual Memory Size

The performance results were about a factor of **50 worse than requirements**. The development team was unable to quickly determine what needed to be done to meet requirements. The project was therefore suspended and restarted with a new development team, project management and quality assurance brief.

Copyright by IBM 2000

Focusing on quality is key for e-business solutions

Summary and Critical Success Factors (1):

- Security issues can kill your e-business. Address these issues carefully in advance - use secure protocols, disclosures for the testers, etc..
- Non functional requirements: define then early in the project
- Use test cases based on business transactions
- Use an independent test team focused only on the requirements - testers who work also as developers do not usually have that "black box" view of the system
- E-business test tool skills: it is necessary to have experience of Web based problem management and browser simulator tools if you need to deploy them fast
- Quality cannot be tested into a product. Plan for quality from the beginning using, for example, IBM's Lifecycle Testing approach

Copyright by IBM 2000

Focusing on quality is key for e-business solutions

Summary and Critical Success Factors (2):

- Web content development and e-business solution application development are quite different. Application development requires a strong quality, requirements and project oriented approach
- Test the performance of your e-business solution under real world conditions (end to end tests, load the server with the right transaction mix, SSL) and in a near real world test environment
- Use Web based tools for problem management and test case documentation
- User profiles: always keep in mind that there are a lot of different types of users out on the Web who are accessing your e-business
- Plan and execute performance tests during all of the solution development phases

Copyright by IBM 2000

Any questions?

How to contact us:

- Rainer Pirker
- e-mail:Rainer_Pirker@at.ibm.com
- Phone: +43-1-1706-4163
- Fax: +43-1-1706-2393
- Web: www.geocities.com/ad_consultants

- Andreas Rudolf
- e-mail:Andreas_Rudolf@at.ibm.com
- Phone: +43-1-1706-4347
- Fax: +43-1-1706-2393

Copyright by IBM 2000

The need for quality – e-business performance testing

Rainer Pirker
Application Development Consultant
e-mail: Rainer_Pirker@at.ibm.com
IBM Global Services
Vienna/Austria

Andreas Rudolf
Application Development Consultant
e-mail: Andreas_Rudolf@at.ibm.com
IBM Global Services
Vienna/Austria

Abstract

Customer expectations are very high in e-business. Above all when they make first contact with your organization. The first impression is key, the user interface, the performance, the right response, right away. Get ready for interfacing with your customers at first click or call – you only have one chance to attract each customer with good usability and fast, reliable responses. One click, one chance, period.

In e-business it is essential to set high standards for the customer perceived usability, reliability and performance of your solution. With e-business solutions you can win with an innovative, high quality solution but just as quickly loose again if your system breaks down, has complex user interfaces or long response times – frustrated customers cancel their transactions, dragging down further the performance of your e-business solution. Your customer care centers get flooded with angry customer calls and e-mails and these customers are lost forever.

We will help you get the best out of your system - for the benefit of your customers. We will define the critical success factors and measurements that enable you to determine the quality of your e-business testing. You will learn how test teams can work independently at different locations with the help of leading edge tools. We will also take a look at the security issues affecting e-business testing.

Summary

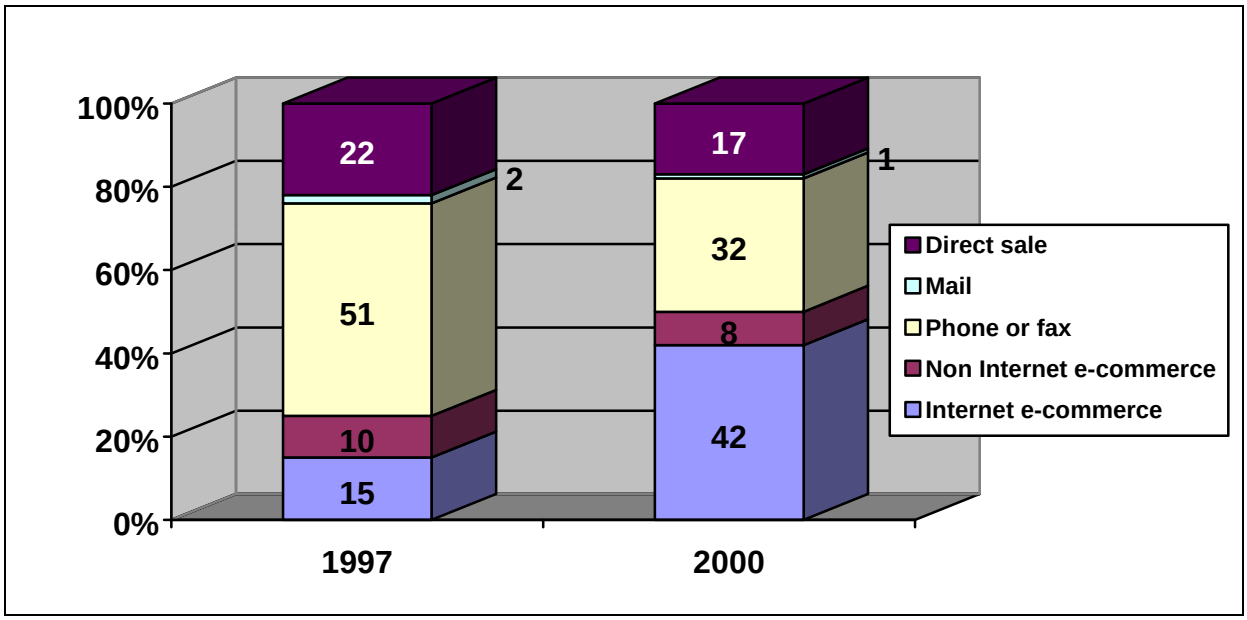
1 Performance aspects on the Web 2
2 Key quality aspects for e-business solutions 3
3 Complexity of e-business solutions 5
4 The performance testing project 6
4.1 Assessing the base 6
4.2 Defining the test strategy 7
4.3 Defining the test concept 7
4.4 Defining the test cases 9
4.5 Skills needed 9
4.6 Building up a test environment 10
4.7 Test Data Security 12
5 Current Project Status 12
6 Critical Success Factors 15
7 References 16

1 Performance aspects on the Web

One of the most dominant factors for the success of an Internet based Web service is the quality of service as perceived by the users of the service. The principle aspect for the users is the “responsiveness” or “good” performance of the service [2]. But what is good performance for a user? The so called “eight second rule” has become an oft-quoted standard. It is not really clear where this statement first gained credence, but now it is widely believed that if a Web page is not downloaded within these eight seconds, the users will bail out – the percentage of users who simply do not wait around for pages to load and instead go on to other sites is called the “bailout rate” for a Web site [13].

A study in the U.S. showed that the average Web backbone connection speed is around 5 Kbd/sec. The main body of Internet users are working with 28.8Kbd and 56Kbd modems. To be able to stay within the “8-second-rule” and not become a “load-time-frustrator”, the opening page of a Web service provider should not exceed 40 Kb for 28.8Kb modem owners. This study further showed that the reduction of an opening page to 34Kb lowered the bailout rate rate from 30% to 6-8% – just because of a one second load time reduction! If users are frustrated by a page, 30% of them will not stay on and become customers of the site. Taking into account the “8-second-rule”, the Web page load times and ISP (Internet service provider) failures, performance and availability weaknesses account for 15% of lost e-business [13].

Figure 1: Increasing strength of the e-commerce sales channel [1]



The increasing strength of e-commerce sales means that the performance, reliability and availability of Web services, perceived as quality of service by the users, are getting to be major economic factors.

Application performance and reliability = business performance [4]

Performance needs to be planned for both before and during the development life-cycle. The complexity of Web based applications is best expressed in download time. The performance which a user perceives

consists not only of the modem, processing speed but also of page size, graphical complexities, hardware, browser type, configurations, ISP and connection speed of the backbone provider. This complexity and the economic importance of e-commerce are creating a growing demand for adequate performance testing.

In figures 2 and 3 you can see some actual statistics about Internet usage in the USA and within the EU and the approximated revenue growth for e-business in Austria for the next years.

Figure 2: Internet distribution 2000

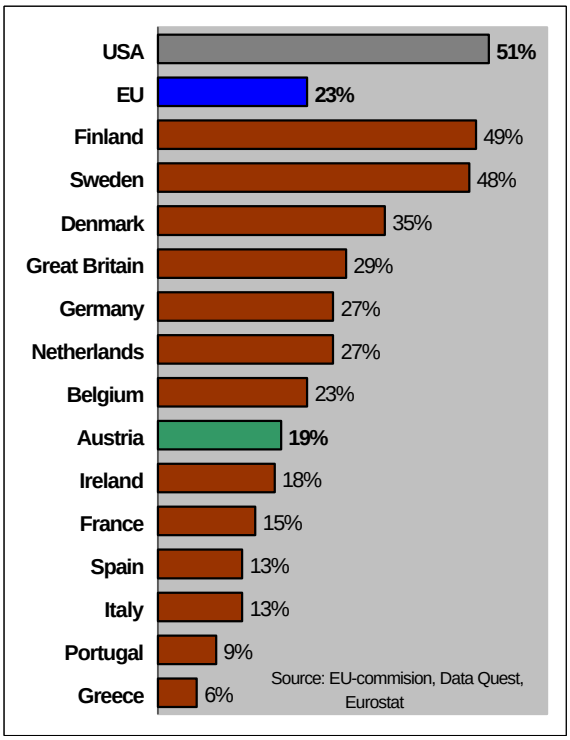
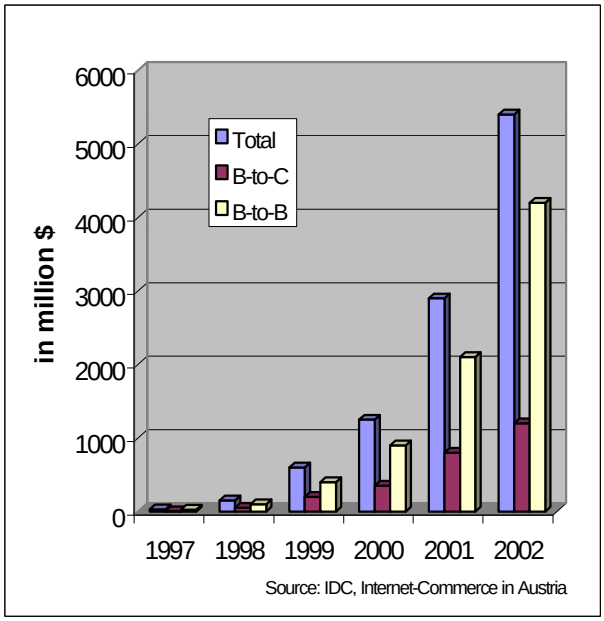


Figure 3: E-business revenue growth in Austria



2 Key quality aspects for e-business solutions

The common key quality aspects of e-business solutions are scalability/performance, security, usability, reliability. Measurements have to be developed for these quality of services for controlling and improving e-business solutions.

- Scalability/Performance: The main aspects of performance are response time and throughput. Response time can be measured at the server side and/or from the user's perspective, in which case it includes all various aspects of the different components influencing Web performance.

  Throughput is usually measured in requests or transaction per second and determines the rate at which the system can deliver work. Throughput is more important for a systems administrator than for an end user – they are concerned much more about the response time.

  E-business workloads are composed of different sessions [6]. Menascé defines a session by a sequence of transactions of different types such as browse, search, select, add to the shopping cart, and pay. There is a great variation of users working with the e-business solution. User

can therefore be classified into different user profile categories. Each user profile uses one specific set of sessions (transaction mix).

End to end performance in general should be measured from the end users point of view – many current solutions tools are net server centric and measure i.e. total requests made per hour, status of each server, boot and shutdown by domain, machine, group and server etc.

- Security: securely transmitted information through the Internet has been identified as a major success factor for Web solutions [10]. We differentiate between security for the end user and security for the company offering the e-business solution. The more secure an Internet operation seems to be for a user (security perception), the likelier they will return to this site and recommend it to other users. If only one security problem goes public about the Web site, the commercial impact to this company will be enormous. For high security operations the user accepts lower performance, i.e. e-banking or paying by credit card.

  The costs of entry into the e-business market is rather low, the risk of failure in this marketplace is potential very high. Internet attacks to companies offering e-business solutions can be structured in two main categories [2] – active or passive attacks on the Web. Passive attacks are for example wire tapping of data, IDs and/or passwords or analysis about the structure of the data stream. Active attacks can influence, spoof, or mutilate data and/or the communication within a company. In the worst case this can modify or delete data, disable the access to the site or delay information for users.

- Usability: some aspects for the usability of an e-business solution are: navigation, user interactions, help messages, computer literacy, printing.

  - Navigation: a Web site must support the user to easily find the information presented on this site. Therefore it improves the usability when the site supports a search facility, a site map and buttons for navigating on each page. The site should not contain any broken links.
  - User interactions: the possibility to interact with the system must be very easy for the end user i.e. “one click” shopping.
  - Help messages: especially users in business to customer (B2C) environment cannot be trained in the usual way like for company internal client-server application, therefore help messages for the e-business solution are important to support usability.
  - Computer literacy: for the user it is most important to find the information on a Web page, he is searching for – graphical design is not the crucial point for an Internet user. 79% of the Internet users run over the text on searching for valid information. Only 16% of surfers are reading an articel word by word [8].
  - Printing: user intend to print Web pages because reading a page on the screen is about 25% slower than reading a hardcopy of this text. If a Web page is designed with white text on a black background the print output will not be very readable.

- Reliability: Critical aspects of reliability of an e-business solution are the duration of down times [5] – duration indicated how long a problem persists – and the mean time between failures (MTBF).

These are technical quality of service criterias. However these criterias should be dedicated from to business measurements like revenue and profit – since these are the key business drivers for e-business. For these business measurements “revenue throughout” (dollar per second) is a useful metric [6].

3 Complexity of e-business solutions

There are two different e-business solution types on the Web [11]: companies that are developing a brand new e-business Web site with complete new functionality, and companies that are simply building Web functionality on top of their existing legacy applications and infrastructure. Companies of the first group are taking the biggest risk and therefore are most in need of testing. Companies in the second category also have to test their applications, but the underlying functionality and programs are already tested and the extent of the test has not to be as large as for the first group.

Superficial considered the typical e-business system is a three-tiered client-server environment – so Web testing is much like client-server testing – but with more potential tiers and greater load variability. Given the basic structure of a database server working with application or business servers, fronted by Web servers there are may be many other special purpose servers involved: firewalls, distributed objects, transactions, authentication, credit card verification and payment servers, backbone provider, ISPs etc. [3]. And with Web site, it is difficult to predict with certainty how many users will hit a site at any given time. There are many different browsers they can use, there is no limit how many people can access the site at once, there may be untrained users and even users who want to crack your site.

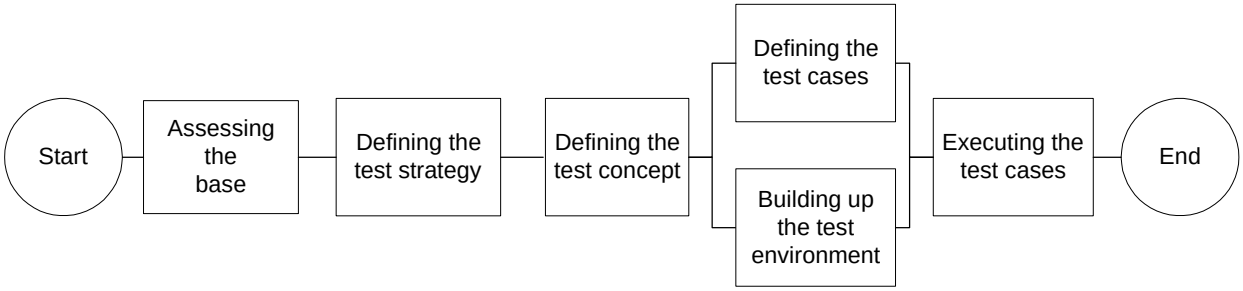
4 The performance testing project

Based on a real life testing project we give you the measurements, the methodology and tools used. Our project provides an electronic bill inquiry service for customers – one paper bill costs them 5 dollars per bill during the lifecycle, and electronic one costs them only a fraction. The customers can view or download their telecom bill over the Internet. IBM's goal was to quickly validate in 4 weeks if the solution provided by a software house meets the performance and stability requirements of the telecom provider.

We teamed with an IBM business partner to form an independent test team to help the customer quickly obtain the desired results.

Based on our experience in other testing projects (e-business, EURO, Y2k, New Application Development) we chose a proven approach as outlined below. You will see that the basic approach is straightforward and not in any way e-business specific.

Figure 4: e-business testing basic approach



4.1 Assessing the base

Before defining the project strategy, we checked existing concepts to see if they could be used as a base for our project. We followed IBM's Application Effectiveness Methodology to search for the best project strategy. We focused on the following to obtain an understanding for the software to be tested and the test environment required:

- Test cases based on business transactions
- User documentation
- Requirement, design and program documentation
- Available test environments
- Test tools (capture and replay, problem and test case management, test data manipulation)

To begin with we had no clear definitions for the performance and stability requirements. We discussed this problem with the customer and defined these requirements. The functional requirements were well defined in the functional specifications. From these performance and stability requirements and the functional specifications we were then able to derive the goals for the tests.

The customer did not have any tools for testing in this area so we brought in tools for Web browser simulation and problem tracking. An isolated test environment that included a Web server, database server and clients was available. Only on the mainframe side did we at first have to interface with the production environment.

4.2 Defining the test strategy

The customer wanted to quickly reach a decision as to whether his e-billing solution was ready to go into production. A very tight time schedule, about 4 weeks, meant that we had no chance to start with the traditional test approach, where we would first have done functional testing and then performance and stress testing. We were able to convince the customer however, that it is absolutely necessary to validate applications against their functional requirements and that this should be the second part of our test plan. Below you can see the outline for both the first and second part of out test plan.

Test plan outline:

1.Part

- Identify the requirements for performance and stress testing
- Define the performance test concept
- Define the performance test cases
- Build-up the test environment
- Customize the HTTP browser simulator for the e-billing solution specifics
- Execute the performance test cases

2.Part

- Define the function test concept
- Define function test cases
- Execute function test cases
- Collateral acceptance of performance tests for each new application driver delivered
- Execute performance and stress tests

4.3 Defining the test concept

In the test concept we defined the “What” and “How” questions. The highlights of this concept were checking functional and non functional requirements and the definition of test cases for measuring end to end performance and security.

For functional requirements the application and the application’s specification were available and sufficient. Non functional requirements such as performance, response time, transaction mix, reliability etc. were only partially available and had to be detailed by us. First we had to define or rather “identify” the customer requirements. We started by identifying the following:

- Transaction types to be tested:
  It was decided the following transaction types should be tested:
  - Logon
  - Show bill
  - Show detailed bill including ticket data
  - Bill comparison against former billing periods
  - Logoff

  The following transaction groups are not part of the concept:
  - Transaction which are only relevant for the call center, since our focus is on end to end tests for customers
  - Transactions, which change existing customer data, since it is planned to use the existing production data like registration and cancellation of the e-billing service.
- Transaction mix:
  Due to a lack of requirements we started with an equally distributed transaction mix
- Transaction rate:
  30 parallel transactions
- Number of users concurrently logged on:
  The system should handle about 1000 concurrently logged on users
- End user response time:
  Maximum of 3 seconds to display one screen of content to the user
- Availability of the system:
  7*24h, maintenance window every 3 months for 3 hours

For transactions with extremely varying data quantities we defined an average data size.

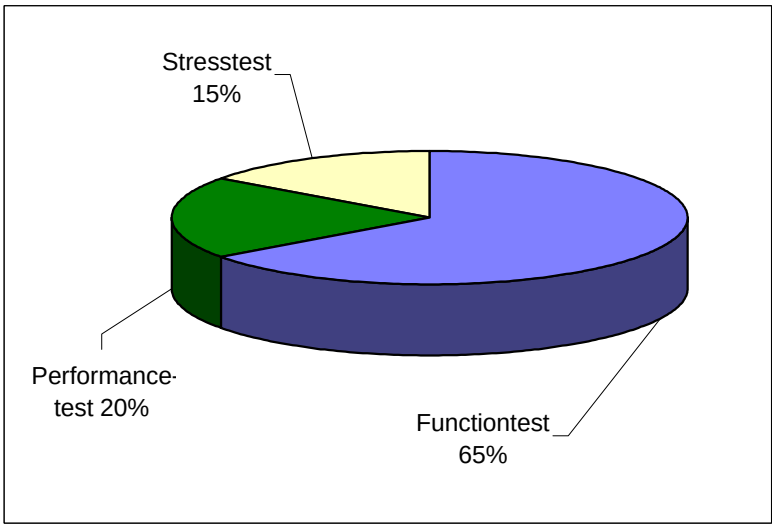
The following variations were considered for their performance impact.

Variation	**Transaction type**
Differentiation between private customers, small and medium companies	Logon, bill, detailed bill including ticket data, Bill comparison against former billing periods
Display of the bill in ATS or EURO (for the display in EURO the Web server needs a conversion)	Bill, detailed bill including ticket data, Bill comparison against former billing periods
Filter, Sorting	Detailed bill including ticket data, Bill comparison against former billing periods

We decided to go for back box testing of the application based on business transactions. We selected the most characteristic transactions that we would eventually need performance figures for.

Apart from performance and stress tests we had to execute a big part of the functional tests. This chart shows that our main test effort was planned to be spent on functional test of business transactions, 20% was planned on performance tests and 15% on stress tests.

Figure 5: Distribution of test effort



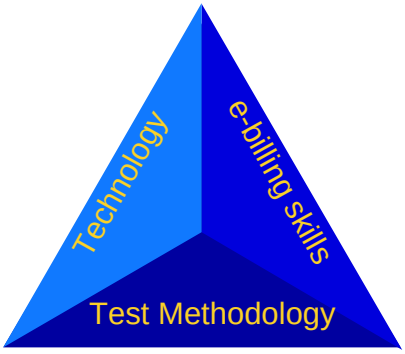
4.4 Defining the test cases

The test cases were defined for business transactions as described by their functional specification and by exploring the application itself. Short descriptions of the test cases were documented in MS Excel.

4.5 Skills needed

For building up our e-billing skills we had the functional specifications and the application itself, for special technology skills like capturing, parsing and replaying HTTP we teamed with an IBM business partner.

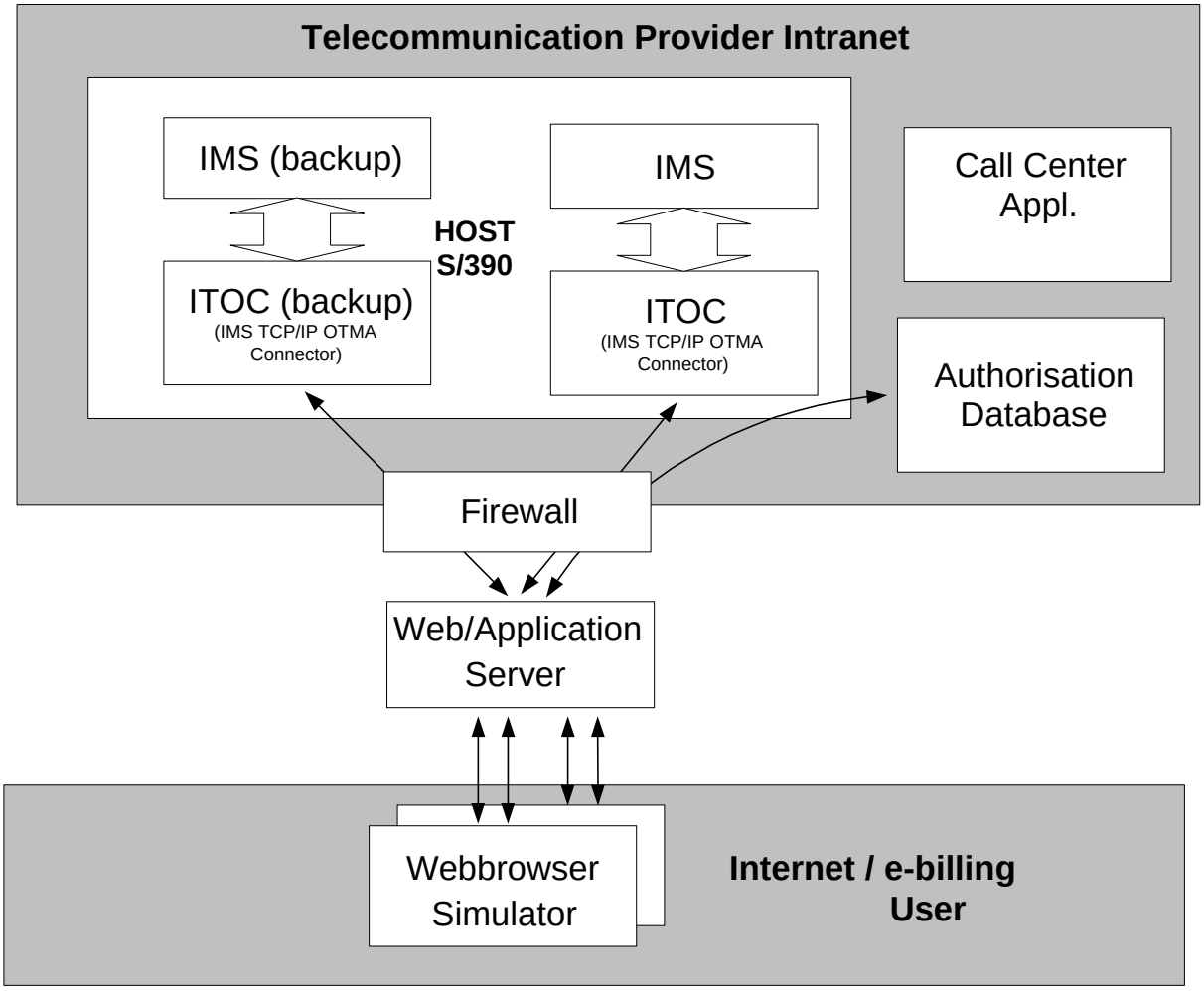
Figure 6: Skills triangle



4.6 Building up a test environment

We had a production like environment as shown in this picture. Only the host connection had to be shared with the production environment.

Figure 7: Test Environment

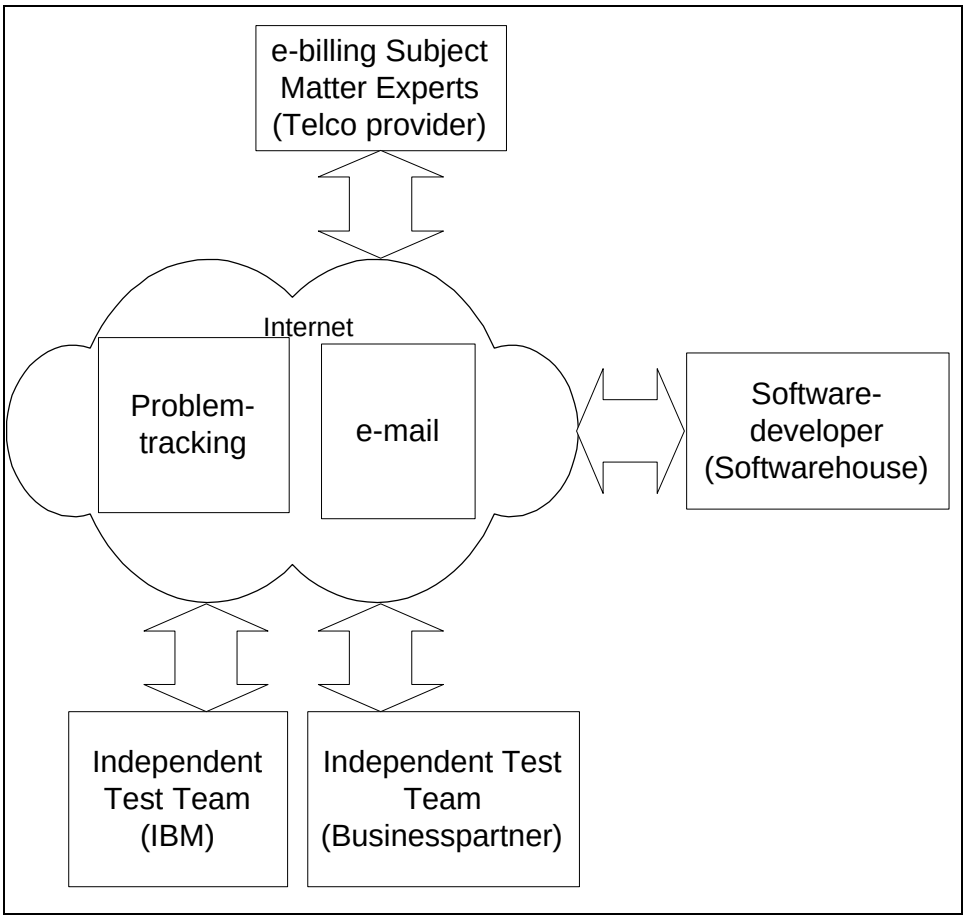


As test tools we used a Web based problem management and browser simulation tool.

4.6.1 Problem management

Since the project team was dispersed across different locations, we need a problem management tool which could be operated over the Internet. With TestTrack Web 1.2 from Seapine Software we choose an easy to configure and easy to use problem management tool. Testers can open, verify and close, developers can answer problem records via the Web.

Figure 8: Problemtracking in different locations



4.6.2 Browser simulator

The test driver runs on LINUX and simulates, as the name says, the HTTP interface of a Web browser by using GET and POST HTTP commands. It is a generic driver, configurable to support without reprogramming changes in the application.

<u>The simulator measures:</u>

- Time for open and connect of a TCP/IP socket
- Time for sending a request
- Time for receiving the first part of the response
- Time for receiving the complete response

Furthermore the simulator delivers the data size of the complete response. For the user the time required for the first screen of information and the display of the complete response is important.

We had to customize the HTTP browser simulator. With a capture/replay facility it was possible to capture performance test scenarios, customize them and then replay them. The customization facility allowed us to exchange URLs, user ids and passwords via configuration files.

4.7 Test Data Security

In our project we used production data as test data.

We want to discuss here shortly the security implications of production test data when dealing with an independent external test team. With production data you give the independent test team access to your confidential company data. When you have an isolated environment, where the application under test and the data is only available inside of your company you can handle security issues via a non-disclosure agreement with the independent test team. However in an e-business project this data is often available on the Internet, since access to the data is handled with a Web browser. Your company data is than available to anyone who has a user identification / password or who can sniff itout overthe net if you are using unsecured HTTP. We have seen examples of extended functionality, like call center access, being tested unsecured over the Internet.

Here are some basic guidelines for your test data:

- Use synthetically created test data or anonymized test data whenever possible
- Use SSL/HTTPS for your sensitive data
- Do not test internal functions (i.e. the call center interface) over the Internet

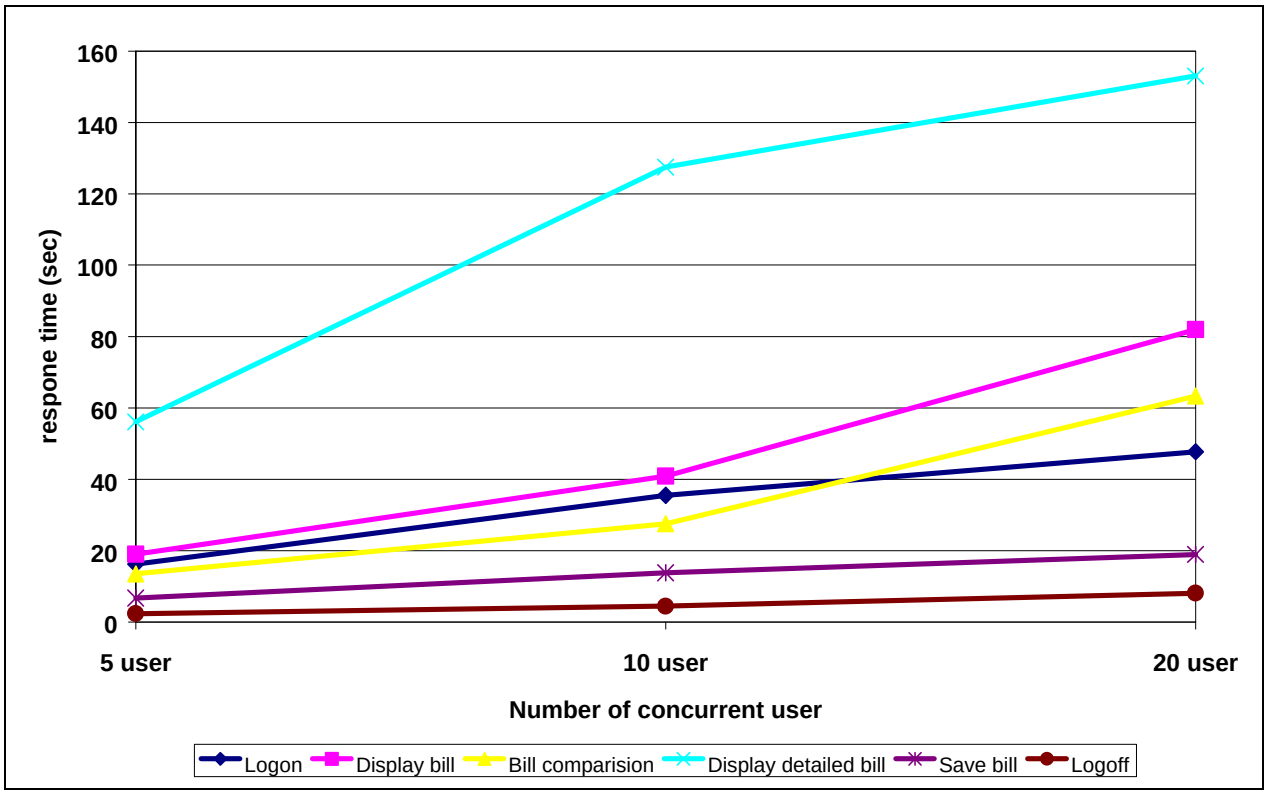
5 Current Project Status

We executed the defined test plan within a three week timeframe focusing on performance aspects and accomplishing the following:

- Performance test concept was written and reviewed
- Performance test cases were defined
- Test environment was built up
- Test scripts for the browser simulator were recorded and adapted
- Problem management tool was selected and deployed including training

After three week we were are to run our first performance test. The performance test simulated a fast user of the e-billing service and included waiting time between transactions. In the first run were not able to collect enough statistical data owing to problems with the test environment. After these were fixed we recorded the response times shown in the graph below for different simulated numbers of concurrent users.

Figure 9: Response times of the different e-billing transactions



The results were unfortunately for removed from the requirements and also far removed from the eight seconds rule. The goal was now for the development team to improve these figures by a factor 50-100. Critical for this scenario was also that the response time increased linearly above 5 concurrent users. This was the first performance measurement for this application every made. Neither the whole application nor parts of this application had been tested in this way before.

The test team prepared and presented to the project management and the development team the following metrics:

- Response time per transaction type dependent on the number of concurrent users
- Response time per transaction dependent on the user profile (private customer, small- , medium businesses)
- To get an approximation for user perceived performance, we also measured when the first HTTP data packet arrived. This was of special interest for transactions such as bill display were a large amount of data is transferred. It depends on the HTTP data stream whether the user will see the first page of the bill data immediately it has been transferred and before the download has been completed.

Additionally the following parameters were measured at the Web server, since it was identified early as the performance critical system in the architecture:

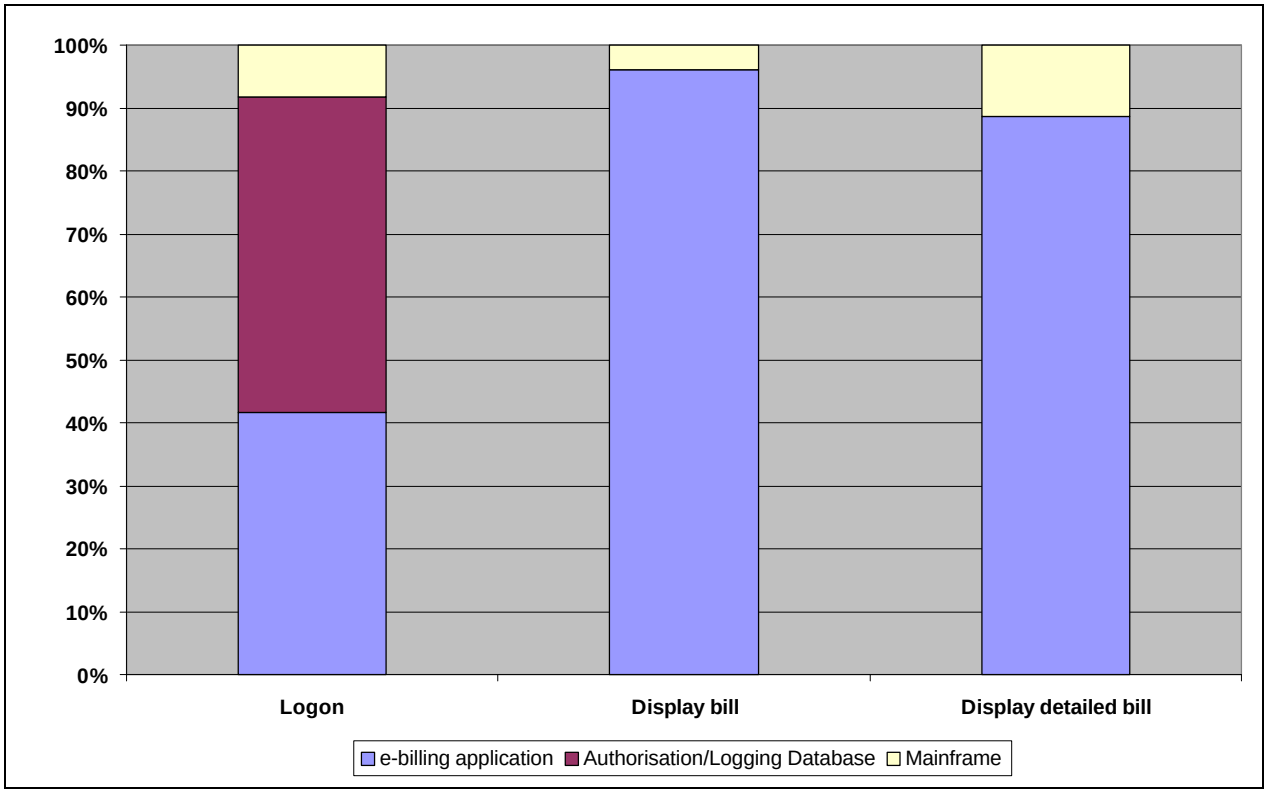
- CPU Load
- Virtual Memory Size

The following actions were now taken by the development team:

- reconfiguration and adaptation of server parameters (reload time of java servlets, number of concurrent threads)
- full usage of the JAVA JIT features
- performance traces

After profiling the application, the performance traces showed (see figure below) that most of the time was consumed by the e-billing application itself. The request processing time on the mainframe were just 5-12% of the overall time. These mainframe transactions had not yet been optimized so these percentages would be even lower in the final version.

Figure 10: Response time distribution of different e-billing transactions measured in the application server



The development team was unfortunately not able to show how the required performance improvements could me achieved within a month. A guess from our side: it was an architectural problem involving the ineffective handling of the data conversion between the mainframe, the application internal format (XML) and the browser representation (HTML). The project is being restarted with a different software house responsible for the development.

For the test team this was a success story. We reached our given target of validating the application and the environment against the requirements in the given timeframe.

6 Critical Success Factors

The test team has shown how with a clear roadmap, the right skills and the right tools your can attack e-business testing in a critical situation.

- *Security can be a killer:*
  Within testing an e-business application, security of the system and data has to be addressed carefully in advance.
- *Non functional requirements:*
  It is key to define the non functional requirements early in the project – best would be the customer has already defined it.
- *Using test cases based on business transactions:*
  Base the definition of test cases on well defined business transactions. The well written functional specifications and access to the subject matter experts enabled us to get fast to results. E-business testers must simulate user behavior and create test scenarios that match reality.
- *Independent test team:*
  The independent team orientated strictly at the requirements – testers which work also as developers do not usually have the "black box" view of the system.
- *E-business test tool skills:*
  It is necessary to have knowledge about e-business test tool skills like Web based problem management and browser simulator and the ability to deploy it fast..
- *Real World test environment:*
  Test your e-business solutions under real world conditions.
- *Application was far from ready for performance testing:*
  When performance tests started this e-business solution was far from ready to be performance tested.
- *Quality cannot be tested into a product:*
  Plan for quality from the beginning. For example use IBM full Lifecycle Testing approach.
- *Automate performance tests as much as possible:*
  Use the right tools i.e. browser simulators for capture and replay of performance test scripts.
- *User profiles:*
  Keep always in mind that there are a lot of different user types out in the Web who are using your e-business solution.
- *Establish metrics right from the beginning:*
  We established metrics right from the beginning. We estimated effort and for performance and stability we defined and measured the relevant figures (samples). Further, each problem found during testing was classified and weighted by severity. Without having metrics established it is difficult, if not impossible, to improve the testing process over time.

7 References

[1] Baresi, C., and Bazzana, G., and Rumi, G., RADIUM – Applying RAD to innovative ERP/ E-commerce projects, *in Proc. Software Quality Week Europe*, November 1999

[2] Conti, Marco, and Gregori, Enrico, and Panzieri, Fabio, Load Distributation among Replicated Web Servers: A QoS-based Approach, *in Proc. The 2nd Workshop on Internet Server Performance*, ACM SIGMETRICS, May 1999

[3] Gerrard, Paul, E-Commerce, Risks and Testing, *in Systeme Evolutif*, http://www.evolutif.co.uk/ September 1999

[4] Kanaan, Elie, E-Commerce: Where Application Testing means Staying in Business – Taking Care of E-Business..., *in Proc. Software Quality Week Europe*, November 1999

[5] Keynote Systems, Inc., Measuring and Improving Your E-Commerce Web Site Performance with Keynote Perspective, *White Paper*, http://www.keynote.com/

[6] Menascé, D. A., and Almeida, Virgilio A. F., and Fonseca, Rodrigo, and Mendes, Marco A., Resource Management Policies for E-commerce Servers, *in Proc. The 2nd Workshop on Internet Server Performance*, ACM SIGMETRICS, May 1999

[7] Menascé, D. A., and Almeida, Virgilio A. F., and Dowdy, Larry W., Capacity Planning and Performance Modeling: from Mainframes to Client-Server Systems, *Prentice Hall, Englewood Cliffs*, 1994

[8] Nielsen, Jakob, *in Proc. Internetworld*, August 1999

[9] Rabin, Steven, E-Commerce Functional Testing and Performance Benchmarking – Methodology and Criteria, *in Proc. Software Quality Week Europe*, November 1999

[10] Soliman, Fawzy, Managing the E-Business Operations, *in Proc. Software Quality Week Europe*, November 1999

[11] Taylor, Sandy, in True Test of The Web by Alan Earls, *in Proc. Informationweek*, http://www.informationweek.com/, January 1999

[12] Telemation Netzwerke, Mit Sicherheit ins Intranet, *White Paper*, http://www.telemation.de/

[13] Zona Research, Inc., The Economic Impacts of Unacceptable Web-Site Download Speeds, *White Paper*, April 1999, http://www.zonaresearch.com/

QW2000 Vendor Technical Paper 6V1

Mr. Ted Burnett
(Watchfire)

Why Web QA Is Essential in all Phases of the Development Process

BACK TO QW2000 PROGRAM

Key Points

- Will explore the benefits of automated website testing
- The downside of a poor website
- Web sites must be tested often

BACK TO QW2000 PROGRAM

QW2000 Paper 6T1

Alan Myrvold
(Entrust Technologies Limited)

Feeling Tcl-ish? Applying Tcl to Real Test Tasks

BACK TO QW2000 PROGRAM

Tcl is a scripting language that can be used to tackle real test tasks. This talk will show simple and advanced ways to apply Tcl to testing software.

Key Points

- Tcl can be used to make new test tools
- Tcl can be used to test Web applications via HTTP
- Tcl can be used to express a state-machine model and generate tests

Abstract...

Tcl can be used to make new test tools
If you have a C/C++ API to exercise product functionality, Tcl can create scripts that automate functional and stress testing.

Tcl can be used to test Web applications via HTTP
Need to test a web page and CGI or Servlet? Tcl can bypass the browser to send HTTP requests to the web page for reliable testing. This can also be used for multi-user testing.

Tcl can be used to express a state-machine model and generate tests
This one is a more complex Tcl program. With a state-machine model of a portion of the system under test, and code for each event in the model, the script can find unreachable states and generate tests to cover the model.

More information on TCL

For more information on TCL, visit the Scriptics web site, or the Tcl Advocacy page for a comparison of TCL to Perl, Python, Java Script and Visual Basic, and a history of TCL.

About the Author

Alan Myrvold is the product verification manager for Entrust/PKI, and has the joy of leading one of the best test teams in the software industry.

Entrust Technologies, http://www.entrust.com, is a global leader in solutions that bring trust to

e-business. Entrust is committed to securing e-business transactions and communications over wireless networks, intranets, extranets and the Internet.

BACK TO QW2000 PROGRAM

Feeling Tcl-ish?
Applying Tcl to Real Test Tasks

Alan Myrvold
Entrust Technologies Limited
Ottawa, Canada

Entrust

Tcl / Tk is ...

- a scripting language
- easy to learn
- extensible / embeddable
- free

Entrust

Simple script

- Suppose you need to summarize a test log file like this:

```
Test#1: PASSED
Test#2: PASSED
Test#3: NOT RUN
-- because data file not present
Test#4: FAILED
Test#5: FAILED
Test#6: PASSED
```


Simple script

```
set fTest [open "test.log" r]
while {[gets $fTest sLine] >= 0}  {
    if [regexp {^Test.*: *(.*)} $sLine sMatch sStatus] {
            if [info exists total($sStatus)] {
                    incr total($sStatus)
            } else {
                    set total($sStatus) 1
            }
    }
}
close $fTest

puts "SUMMARY"
puts "-------"
foreach sStatus [lsort [array names total]] {
    puts "   $sStatus: $total($sStatus)"
}
```

Simple script

- The script produces this output:

```
SUMMARY
-------
  FAILED: 2
  NOT RUN: 1
  PASSED: 3
```


Adding commands to Tcl

- Tcl is extensible
- New commands can be added
- Scripts can take advantage of the power of Tcl's string manipulation and regular expressions, and Tk's GUIs
- Multiple chunks of functionality can be glued together

Testing web servlets + cgi's

- Use *package require http*
- No browser involved … server component of application can be tested
- Functional tests, scalability, multi-user
- Still need to test with browsers

Testing web servlets + cgi's

```
proc GetStockPrice {ticker} {
    package require http
    set q [::http::formatQuery s $ticker d v1]
    set tkn [::http::geturl http://finance.yahoo.com/q -query $q]
    ::http::wait $tkn
    set data [::http::data $tkn]
    return [ExtractPrice $data]
}

foreach tick {ENTU MSFT CORL COGN} {
    set price [GetStockPrice $tick]
    puts "$tick $price"
}
```

Testing web servlets + cgi's

```
proc ExtractPrice {data} {
    regsub -all {</?su.>} $data {} data
    set s [regexp {<td nowrap><b>([^<]+)</b>} $data m price]
    regsub { } $price {+1.0*} price
    return [expr $price]
}
```


Adding a GUI with Tk

- You may want to put together a simple GUI to make a test package easy to run

Adding a GUI with Tk

```
frame .fApp
label .lApp -text "Application directory" -width 20 -anchor w
entry .eApp -width 20 -textvariable eApp
pack .lApp .eApp -side left -in .fApp
frame .fLog
label .lLog -text "Log file" -width 20 -anchor w
entry .eLog -width 20 -textvariable eLog
pack .lLog .eLog -side left -in .fLog
frame .fTest
radiobutton .rSanity -text "Sanity" -variable vRun -value Sanity
radiobutton .rFull -text "Full" -variable vRun -value Full
pack .rSanity .rFull -side left -in .fTest
button .bGo -text "Execute Tests" -command ExecuteTests
pack .fApp .fLog .fTest .bGo
```


State / event model

- A few pages of code allows this declaration

```
Event START  collect_12_apples HAVE_APPLES
Event HAVE_APPLES pay PAID_FOR_APPLES
Event PAID_FOR_APPLES exit_store DONE_SHOPPING
```

to be analyzed and a table of states and events created, unreachable states found, and tests to cover the transitions.

State / event model

- The test generated:

All states are reachable

Tests that cover all possible transitions

1 START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES exit_store DONE_SHOPPING

- That one was simple … how about:

Event START collect_12_apples HAVE_APPLES
Event HAVE_APPLES pay PAID_FOR_APPLES
Event PAID_FOR_APPLES exit_store DONE_SHOPPING

Event START exit_store DONE_SHOPPING
Event HAVE_APPLES collect_12_apples HAVE_APPLES
Event PAID_FOR_APPLES collect_12_apples HAVE_APPLES

State / event model

All states are reachable

Tests that cover all possible transitions

1 START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES exit_store DONE_SHOPPING

2 START exit_store DONE_SHOPPING

3 START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES collect_12_apples HAVE_APPLES collect_12_apples HAVE_APPLES

State / event model

Or even this one:

Event START collect_12_apples HAVE_APPLES
Event HAVE_APPLES pay PAID_FOR_APPLES
Event PAID_FOR_APPLES exit_store DONE_SHOPPING

Event START exit_store DONE_SHOPPING
Event HAVE_APPLES collect_12_apples HAVE_APPLES
Event PAID_FOR_APPLES collect_12_apples HAVE_APPLES

Event HAVE_APPLES exit_store STOLE_APPLES
Event START pay GAVE_MONEY_AWAY
Event PAID_FOR_APPLES pay GAVE_MONEY_AWAY

Event GAVE_MONEY_AWAY exit_store DONE_SHOPPING
Event STOLE_APPLES pay PAID_FOR_APPLES

Event HAVE_LOTTERY_TICKET got_luck VERY_RICH
Event VERY_RICH buy_store PAID_FOR_APPLES

State / event model

The following states are unreachable: HAVE_LOTTERY_TICKET VERY_RICH

Tests that cover all possible transitions

1 START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES exit_store DONE_SHOPPING

2 START exit_store DONE_SHOPPING

3 START pay GAVE_MONEY_AWAY exit_store DONE_SHOPPING

4 START collect_12_apples HAVE_APPLES exit_store STOLE_APPLES pay PAID_FOR_APPLES collect_12_apples HAVE_APPLES collect_12_apples HAVE_APPLES

5 START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES pay GAVE_MONEY_AWAY

Learning more...

- http://www.scriptics.com

From the script, you can see that Tcl supports defining variables on the fly, regular expressions, arrays, and basic file I/O. In this example, the script would print:

```
SUMMARY
-------
   FAILED: 2
   NOT RUN: 1
   PASSED: 3
```

Hooking up to an API to make new test tools

Suppose you have some functionality to test, and it is exposed in the form of a C/C++ API, for example:

```
// Here is a function that needs to be tested.  Returns 0 on success.
int Sched_MakeAppointment(char* szName, char* szYYYYMMDDwhen);

// And a function to convert a non-zero return to a meaningful string
int Sched_ErrorString(int nError,char** pszError);

// And free the string allocates
void Sched_FreeString(char* szError);
```

In the real world, you might be delivering a GUI or Web application to the client, and maybe a toolkit to interface to it too, and there will be many, many more functions. You could write a C/C++ program to test this, but every time you add a test case you will need to recompile.

What sort of tests should you try? Long names, short names, special characters in names, valid dates, invalid dates, dates in the past, dates far in the future, 2 digit years, NULL pointers, and more.

To add a command to Tcl that calls this function, we just need to write a bit of wrapper code:

```
int Tsched_MakeAppointment_CmdProc(
        ClientData clientData,
        Tcl_Interp *interp,
        int argc,
        char *argv[])
{
        // check arg count: must be 2 args (argc is 3)
        if (argc != 3) {
                char* szError = "Syntax:\nsched_make_appointment name date";
                Tcl_Obj *tResult = Tcl_NewStringObj(szError,-1);
                Tcl_SetObjResult(interp,tResult);
                return TCL_ERROR;
        }
        int rc = Sched_MakeAppointment(argv[1], argv[2]);
        if (rc) {
                char* szError = NULL;
                Sched_ErrorString(rc,&szError);
                Tcl_Obj *tResult = Tcl_NewStringObj(szError,-1);
                Sched_FreeString(szError);
                Tcl_SetObjResult(interp,tResult);
                return TCL_ERROR;
        }
```

```
        return TCL_OK;
}


extern "C" TSCHED_API int Tsched_Init(Tcl_Interp *interp)
{
        Tcl_InitStubs(interp,"8.0",0);
        Tcl_CreateCommand(interp, "sched_make_appointment",
             Tsched_MakeAppointment_CmdProc, NULL, NULL);
        return TCL_OK;
}
```

And now the command is in Tcl:

```
E:\tcl83\bin>tclsh83
% load tsched
% sched*
Syntax is:
        sched_make_appointment name date
% sched_make_appointment test 405
Bad date
% sched_make_appointment test 20000405
%
```

Why bother doing this? Tcl can be a productive programming environment, with built−in string and regular expression handling. Also, Tcl can be used as the glue that allows many components to be brought together.

Testing web applications with HTTP

A web application has two sides: the client and the server. As applications grow more complex, both the client and server get more complicated to test.

There are web testing tools available, but Tcl can provide a cheap and easy way to test server functionality.

Here's a real example that extracts stock prices from Yahoo's finance web site.

```
# no error checking, but this shows sending information
# to a web site and extracting a piece.
proc GetStockPrice {ticker} {
        package require http
        set q [::http::formatQuery s $ticker d v1]
        set tkn [::http::geturl http://finance.yahoo.com/q −query $q]
        ::http::wait $tkn
        set data [::http::data $tkn]
        return [ExtractPrice $data]
}

proc ExtractPrice {data} {
        regsub −all {</?su.>} $data {} data
        set s [regexp {<td nowrap><b>([^<]+)</b>} $data m price]
        regsub { } $price {+1.0*} price
        return [expr $price]
}

foreach tick {ENTU MSFT CORL COGN} {
        set price [GetStockPrice $tick]
        puts "$tick $price"
}
```

Testing a state model

A state model is one way to look at software. As a tester, the process of thinking about how to model a program with states can often uncover more flaws than actually trying each transition.

A state model is a collection of states, and events. Each event moves us from one state to another. With a simple model, the destination state can be uniquely determined given the current state and the event ... no history is required.

For example, consider going to a store to buy some apples. Here is a simple model with 4 states and 3 events. Note that states are written in uppercase and events in lowercase. The 4 states are: START, HAVE_APPLES, PAID_FOR_APPLES, and DONE_SHOPPING. The 3 events are collect_12_apples, pay, and exit_store.

With the Tcl scripts shown in the appendix, the model is just:

```
Event START  collect_12_apples HAVE_APPLES
Event HAVE_APPLES pay PAID_FOR_APPLES
Event PAID_FOR_APPLES exit_store DONE_SHOPPING
Analyze_events state_report.html test_concise.tst test_full.tst
```

The script generates three HTML tables.

	START	HAVE_APPLES	PAID_FOR_APPLES	DONE_SHOPPING
collect_12_apples	HAVE_APPLES			
pay		PAID_FOR_APPLES		
exit_store			DONE_SHOPPING	

	Tests that cover all possible transitions
1	START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES exit_store DONE_SHOPPING

	Tests that cover all possible transitions, allowing destructive verification
1	START collect_12_apples Verify(HAVE_APPLES)
2	START collect_12_apples HAVE_APPLES pay Verify(PAID_FOR_APPLES)
3	START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES exit_store Verify(DONE_SHOPPING)

After some thought, we might add some transitions to this model:

```
Event START exit_store DONE_SHOPPING
Event HAVE_APPLES collect_12_apples HAVE_APPLES
```

```
Event PAID_FOR_APPLES collect_12_apples HAVE_APPLES
```

And generate these tables.

	START	HAVE_APPLES	PAID_FOR_APPLES	DONE_SHOPPING
collect_12_apples	HAVE_APPLES	HAVE_APPLES	HAVE_APPLES	
pay		PAID_FOR_APPLES		
exit_store	DONE_SHOPPING		DONE_SHOPPING	

	Tests that cover all possible transitions
1	START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES exit_store DONE_SHOPPING
2	START exit_store DONE_SHOPPING
3	START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES collect_12_apples HAVE_APPLES collect_12_apples HAVE_APPLES

	Tests that cover all possible transitions, allowing destructive verification
1	START collect_12_apples Verify(HAVE_APPLES)
2	START exit_store Verify(DONE_SHOPPING)
3	START collect_12_apples HAVE_APPLES pay Verify(PAID_FOR_APPLES)
4	START collect_12_apples HAVE_APPLES collect_12_apples Verify(HAVE_APPLES)
5	START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES exit_store Verify(DONE_SHOPPING)
6	START collect_12_apples HAVE_APPLES pay PAID_FOR_APPLES collect_12_apples Verify(HAVE_APPLES)

Thinking a bit more about the model, we might add these transitions and states.

```
Event HAVE_APPLES exit_store ERROR(STOLE_APPLES)
Event START pay ERROR(GAVE_MONEY_AWAY)
Event PAID_FOR_APPLES pay ERROR(GAVE_MONEY_AWAY)

Event GAVE_MONEY_AWAY exit_store DONE_SHOPPING
Event STOLE_APPLES pay PAID_FOR_APPLES
```

And finally, to prove that unreachable states are detected by this script, we can add these transitions and states.

```
Event HAVE_LOTTERY_TICKET got_luck VERY_RICH
Event VERY_RICH buy_store PAID_FOR_APPLES
```

Adding in a TK GUI

Another advantage of Tcl is how easy it is to create a GUI using Tk.

You may have a test package that needs a few configuration details, such as how to find the software under test, what tests to run, and where the output should be placed.

All this can be done by creating a form with Tk.

```
# Defaults for the Tk GUI
set eApp {c:\test.exe}
set eLog {c:\test.log}
set vRun Sanity

# Example of a Tk GUI
frame .fApp
label .lApp −text "Application directory" −width 20 −anchor w
entry .eApp −width 20 −textvariable eApp
pack .lApp .eApp −side left −in .fApp
frame .fLog
label .lLog −text "Log file" −width 20 −anchor w
entry .eLog −width 20 −textvariable eLog
pack .lLog .eLog −side left −in .fLog
frame .fTest
radiobutton .rSanity −text "Sanity" −variable vRun −value Sanity
radiobutton .rFull −text "Full" −variable vRun −value Full
pack .rSanity .rFull −side left −in .fTest
button .bGo −text "Execute Tests" −command ExecuteTests
pack .fApp .fLog .fTest .bGo

# Code executed on the button click
proc ExecuteTests {} {
        global eApp eLog vRun
        # execute tests here
}
```

Learning more

There's lots more to discover. Visit Scriptic's web site at http://www.scriptics.com, or drop down to your local bookstore to buy one of many books on Tcl, such as Tcl and the Tk Toolkit, by John Ousterhout.

Appendix: Tcl code for the state model

```
# Routines for handling the specification of
# and verification of state machines

# list of states and a list of event names
set glbState_List [list]
set glbEvent_List [list]
set glbHTMLEventList {}

# add a state to the list of states, if it isn't there already
proc Add_to_unique_list {list_name element_value} {
        global $list_name
        if {[lsearch −exact [subst $$list_name] $element_value] == −1} {
                lappend $list_name $element_value
        }
}

# declare an event
proc Event {from_state event_name to_state} {
        global glbEventForwardsList glbEventStateForwardsList glbStateBackwardsList \
                glbHTMLEventList glbEventStateErrorList

        # An ERROR is a special case ... the to_state is the same
```

```
        # as the from, but an ERROR is generated
        if {[regexp {ERROR\((.*)\)} $to_state match err_state]} {
                set to_state $err_state
                set is_error "ERROR"
        } else {
                set is_error "SUCCESS"
        }

        # enforce style ... events lower case and states upper
        if {![regexp {^[A−Z_]+$} $from_state] || ![regexp {^[A−Z_]+$} $to_state]} {
                error "States must be upper case"
        }
        if {![regexp {^[a−z0−9_]+$} $event_name]} {
                error "Events must be lower case"
        }

        Add_to_unique_list glbState_List $from_state
        Add_to_unique_list glbState_List $to_state
        Add_to_unique_list glbEvent_List $event_name

        lappend glbStateBackwardsList($to_state) $from_state
        lappend glbEventForwardsList($from_state) $event_name

        if {[info exists glbEventStateForwardsList($from_state,$event_name)]} {
                error "Duplicate State / event"
        }
        set glbEventStateForwardsList($from_state,$event_name) $to_state
        set glbEventStateErrorList($from_state,$event_name) $is_error

        if {![info exists glbStateBackwardsList($from_state)]} {
                set glbStateBackwardsList($from_state) [list]
        }
        if {![info exists glbEventForwardsList($to_state)]} {
                set glbEventForwardsList($to_state) [list]
        }

        # Add to list for HTML output
        set glbHTMLEventList "$glbHTMLEventList<br>Event $from_state \
                $event_name $to_state"
}

# Mark a given state as reachable.  Recursively.
# Also compute depths of all nodes reachable from the given node
# as 1 + current depth
proc Mark_as_reachable {state_name} {
        global glbReachability glbEventForwardsList glbEventStateForwardsList \
                glbStateDepth

        # if already marked, we're done
        if {$glbReachability($state_name) == "REACHABLE"} {
                return
        }

        # mark as reachable
        set glbReachability($state_name) REACHABLE

        # record depths for next level
        set next_depth [expr 1 + $glbStateDepth($state_name)]
        foreach fwd_event $glbEventForwardsList($state_name) {
                set to_state $glbEventStateForwardsList($state_name,$fwd_event)
                if {$glbStateDepth($to_state) > $next_depth} {
                        # found a shorter path to the state
                        set glbStateDepth($to_state) $next_depth
                }
        }
```

```
        # mark all the states we can go to as reachable
        # this is separate from the depths loop on purpose,
        # to optimize depth assignment
        foreach fwd_event $glbEventForwardsList($state_name) {
                set to_state $glbEventStateForwardsList($state_name,$fwd_event)
                Mark_as_reachable $to_state
        }
}

# Determine reachability and depth (from START) of states
proc Determine_reachability {} {
        global glbState_List glbReachability glbStateDepth

        # mark all as unreachable and deep
        foreach state $glbState_List {
                set glbReachability($state) UNREACHABLE
                set glbStateDepth($state) 99999
        }

        # can reach the start
        set glbStateDepth(START) 0
        Mark_as_reachable START

        # count how many are unreachable
        set unreachable_count 0
        foreach state $glbState_List {
                if {$glbReachability($state) == "UNREACHABLE"} {
                        incr unreachable_count
                }
        }
        return $unreachable_count
}

# Find a transition to a reachable state that has not been covered
proc Find_uncovered_transition {} {
        global glbState_List glbEventForwardsList glbEventStateForwardsList \
                glbReachability glbCovered

        foreach from_state $glbState_List {
                foreach fwd_event $glbEventForwardsList($from_state) {
                        set to_state $glbEventStateForwardsList($from_state,$fwd_event)
                        if {$glbReachability($to_state) == "REACHABLE" &&
                                $glbCovered($from_state,$fwd_event) == "NO"} {
                                return [list $from_state $fwd_event]
                        }
                }
        }

        return [list]
}

# Find an uncovered transition backwards, or,
# if not found, find a transition to a node with
# the least distance from start
proc Find_previous_transition {state bConcise} {
        global glbStateBackwardsList glbEventForwardsList glbEventStateForwardsList \
                glbCovered glbReachability glbStateDepth

        set rc [list]
        set minDepth 99999
        foreach from_state $glbStateBackwardsList($state) {
                if {$glbReachability($from_state) != "REACHABLE"} continue
                foreach fwd_event $glbEventForwardsList($from_state) {
                        set to_state $glbEventStateForwardsList($from_state,$fwd_event)
                        if {$state != $to_state} continue
                        if {$bConcise && $glbCovered($from_state,$fwd_event) == "NO"} {
```

```
                                set rc [list $from_state $fwd_event]
                                return $rc
                        } else {
                                if {$glbStateDepth($from_state) < $minDepth} {
                                        set minDepth $glbStateDepth($from_state)
                                        set rc [list $from_state $fwd_event]
                                }
                        }
                }
        }
        return $rc
}

# Find an uncovered transition forwards, or,
# if not found, return an empty list
proc Find_next_transition {state} {
        global glbEventForwardsList glbCovered
        foreach fwd_event $glbEventForwardsList($state) {
                if {$glbCovered($state,$fwd_event) == "NO"} {
                        return [list $state $fwd_event]
                }
        }
        return [list]
}

# Arrive at tests that cover all possible STATE/event pairs
proc Design_tests {bConcise} {
        global glbState_List glbEventForwardsList glbEventStateForwardsList \
                glbReachability glbCovered glbTestlist
        # Mark all transitions as uncovered, except those from
        # unreachable states
        foreach state $glbState_List {
                foreach fwd_event $glbEventForwardsList($state) {
                        if {$glbReachability($state) == "REACHABLE"} {
                                set glbCovered($state,$fwd_event) NO
                        } else {
                                set glbCovered($state,$fwd_event) YES
                        }
                }
        }

        # start with no tests
        set glbTestlist [list]

        # Find an uncovered transition to a reachable state
        set uncovered [Find_uncovered_transition]
        while {[llength $uncovered] > 0} {
                set test_sequence $uncovered
                set state [lindex $uncovered 0]
                set fwd_event [lindex $uncovered 1]
                set glbCovered($state,$fwd_event) YES
                set to_state $glbEventStateForwardsList($state,$fwd_event)

                # Work backwards to start, getting the shortest path to START state
                while {$state != "START"} {
                        set prev_transition [Find_previous_transition $state $bConcise]

                        # add it to test
                        set state [lindex $prev_transition 0]
                        set fwd_event [lindex $prev_transition 1]
                        if {$bConcise} {
                                # only mark it as covered if we are trying to
                                # produce fewer tests
                                set glbCovered($state,$fwd_event) YES
                        }
                        set test_sequence [linsert $test_sequence 0 $state $fwd_event]
```

```
                }

                # Work forwards, only hitting uncovered transitions
                # but only if we are trying to produce fewer tests
                if {$bConcise} {
                        set next_step [Find_next_transition $to_state]
                        while {[llength $next_step] > 0} {
                                set state [lindex $next_step 0]
                                set fwd_event [lindex $next_step 1]
                                set glbCovered($state,$fwd_event) YES
                                set to_state $glbEventStateForwardsList($state,$fwd_event)
                                set next_step [Find_next_transition $to_state]
                                lappend test_sequence $state $fwd_event
                        }
                }

                # pop on last state
                if {$bConcise} {
                        lappend test_sequence $to_state
                } else {
                        lappend test_sequence [format "Verify(%s)" $to_state]
                }

                # Add to test list
                lappend glbTestlist $test_sequence

                # find the next
                set uncovered [Find_uncovered_transition]
        }
}

# Do all the work to analyze the model and
# create an HTML page to document the results
# and a TCL file for the code.
proc Analyze_events {html_file test_file1 test_file2} {
        global glbState_List glbReachability glbEvent_List \
                glbEventStateForwardsList glbTestlist glbHTMLEventList \
                glbEventStateErrorList

        set fHTML [open $html_file w]
        puts $fHTML {<html><title>State chart</title><body \
                bgcolor="#f8f8f8"><b>State chart</b><br>}

        # Show input
        puts $fHTML "$glbHTMLEventList<p>"
        # Check reachability
        set unreachable_count [Determine_reachability]
        # show state chart
        puts $fHTML {<table border=3><tr><td>}
        foreach state $glbState_List {
                if {$glbReachability($state) == "REACHABLE"} {
                        puts $fHTML "<td>$state"
                } else {
                        puts $fHTML "<td><font color=red>$state</font>"
                }
        }
        puts $fHTML {</tr>}
        foreach event $glbEvent_List {
                puts $fHTML "<tr><td>$event"
                foreach state $glbState_List {
                        set to_state {}
                        if {[info exists glbEventStateForwardsList($state,$event)]} {
                                set to_state $glbEventStateForwardsList($state,$event)
                                if {$glbEventStateErrorList($state,$event) == "ERROR"} {
                                        set to_state "<font \
                                                color=red>!$to_state</font>"
```

```
                                }
                        }
                        puts $fHTML "<td>$to_state"
                }
                puts $fHTML {</tr>}
        }
        puts $fHTML {</table>}
        # Show reachability info
        if {$unreachable_count > 0} {
                set sReach "The following states are unreachable:"
                foreach state $glbState_List {
                        if {$glbReachability($state) == "UNREACHABLE"} {
                                set sReach "$sReach $state"
                        }
                }
                puts $fHTML "<p><b>$sReach</b>"
                puts $sReach
        } else {
                puts $fHTML "<p>All states are reachable"
        }
        foreach bConcise [list 1 0] {
                if {$bConcise == 1} {
                        set fTEST [open $test_file1 w]
                } else {
                        set fTEST [open $test_file2 w]
                }
                # UNCOMMENT to create preamble in testfile
                # puts $fTEST [Start_testfile]
                Design_tests $bConcise
                if {$bConcise} {
                        puts $fHTML "<p><Table border=3><tr> \
                                <td><td>Tests that cover all possible \
                                transitions</tr>"
                } else {
                        puts $fHTML "<p><Table border=3><tr> \
                                <td><td>Tests that cover all possible \
                                transitions, allowing destructive verification</tr>"
                }
                set nTest 0
                foreach test_sequence $glbTestlist {
                        # UNCOMMENT TO GENERATE TEST
                        # puts $fTEST [Start_test $test_sequence]
                        incr nTest
                        puts $fHTML "<tr><td>$nTest<td> \
                                $test_sequence</tr>"
                        set test_steps [split $test_sequence]
                        set n_item 0
                        foreach test_item $test_steps {
                                incr n_item
                                if {[expr $n_item % 2] ==  0} {
                                        set event $test_item
                                        set is_error $glbEventStateErrorList($state,$event)
                                        # UNCOMMENT TO GENERATE TEST EVENT
                                        # puts $fTEST [Event_$event $state $is_error]
                                } else {
                                        set state $test_item
                                }
                        }
                }
                puts $fHTML  {</table><p>}
                close $fTEST
        }
}
```

QW2000 Paper 6A1

Dr. Jean Hartmann & Mr. Claudio Imoberdorf (Siemens Corporate Research)

Functional Testing of Distributed, Component-Based Software (6A1)

BACK TO QW2000 PROGRAM

Key Points

- Structured test design based on UML state charts.
- Tool support for test generation and test execution.
- Methods suitable for testing individual components or sets of integrated components.

Presentation Abstract

Increasing numbers of software developers are using the Unified Modeling Language (UML) and associated visual modeling tools as a basis for the design and implementation of their distributed, component-based applications. Once developed, however, test cases must be designed, generated and executed to validate the components. This is especially important for unit and integration testing.

At Siemens Corporate Research, we are addressing this issue by integrating our test generation and test execution technology with commercial UML modeling tools such as Rational Rose; the goal being a *design-based* testing environment.

In order to generate test cases automatically, developers first define the dynamic behavior of their components via UML Statechart Diagrams. These views then need to be annotated with additional, test-specific information such as coverage criteria, data variations and interconnected, in the case of multiple components.

With the help of our test generation technology, test cases are then systematically derived from the annotated UML StateChart Diagrams and executed using our test execution environment, which was developed specifically for interfacing to components based on COM/DCOM and CORBA/IDL middleware.

Providing such a design-based testing environment ensures major benefits for developers:

> With minimal additional effort, developers can reuse their component designs as test specifications. In the case of code changes, these test specifications can be updated and used as a basis for automatically generating a new set of regression tests.
>
> Developers no longer need to manually implement custom test drivers for components, which can be especially tedious and error-prone in the case of distributed components. Executable

test drivers are automatically generated directly from the test specifications.

The environment can be applied to individual as well as a collection of components, making it suitable for use during unit and integration testing.

In future, we see the delivery of software components being accompanied with a standardized test specification, possibly based on UML StateChart Diagrams. This will be necessary to ensure compliance of the component as it is integrated into a larger software system.

About the Author

Jean Hartmann is a project manager at Siemens Corporate Research responsible for software testing technology. His research interests focus on new techniques and tools for testing components, graphical user interfaces, and internet-based systems. He received the Ph.D. degree in computer science from the University of Durham, UK, in 1993 where is thesis topic emphasized improved regression testing techniques.

Claudio Imoberdorf is a Member of the Technical Staff at Siemens Corporate Research. He has seven years experience in the area of software design, software developmemnt, and component technologies. Prior to joining Siements he was a lead designer on a component-based building automation system at Siemens Building Technology, Switzerland.

BACK TO QW2000 PROGRAM

Functional Testing of Distributed, Component-Based Systems

Jean Hartmann and Claudio Imoberdorf
Siemens Corporate Research
Princeton NJ 08540

Tel: ++1 609 734 3361
Fax: ++1 609 734 6565
Email: jhartmann@scr.siemens.com

Objectives

- Growing Importance of Domain-specific Frameworks
- Modeling of Software Components and their Interfaces
  - Using a standardized design notation => UML
  - The model must express the component interactions
- Using the Model for Test Generation and Execution
  - Focusing on unit and integration testing (black-box)
  - Automating the test generation and execution steps
  - Providing test execution support for middleware-based components

Modeling Components

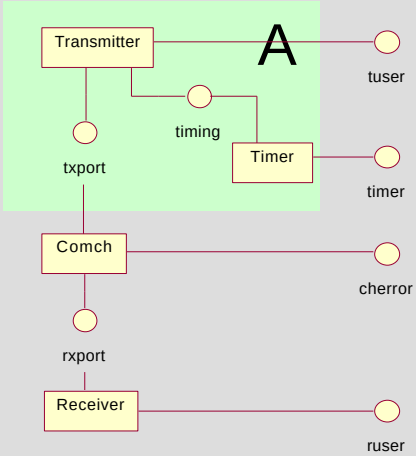
- Use UML Statecharts to model each component (object)
- Statecharts should reflect the normal and erroneous behavior of the component
- To facilitate component interaction, we extend UML with a new transition labeling scheme
- For integration testing, users define the collection of components to be tested (*subsystem definition)*
- Assumptions:
  - Point-to-point communication semantics between components rather than a shared event model
  - Communications are synchronous (blocking)
  - Implementation is deterministic

Steps Taken During Model Composition

1. Normalizing the UML-based Models
   - Importing the Rational Rose repository
   - Noting the subsystem definition, if any
   - Resolving transitions with multiple events
2. Composing the Global Behavioral Model
   - Applying an incremental composition and reduction algorithm
   - Determining a composition order (based on subsystem definition)

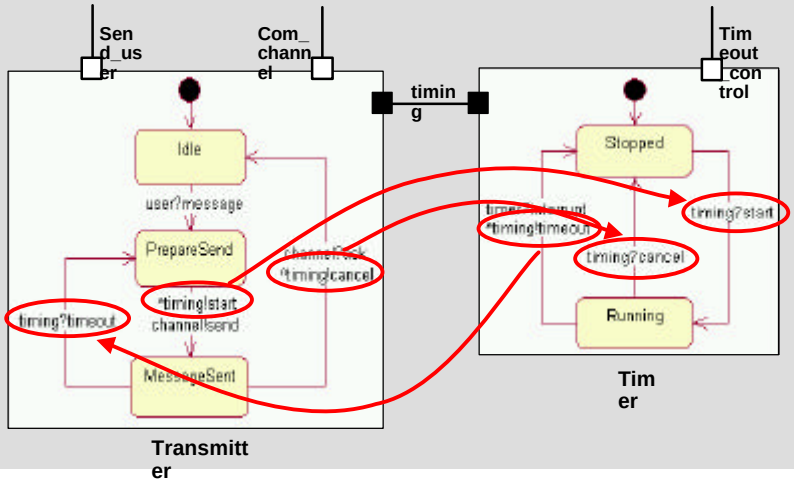
- Scalability
  - Algorithm complexity is better than exponential

Example: A Communication Protocol

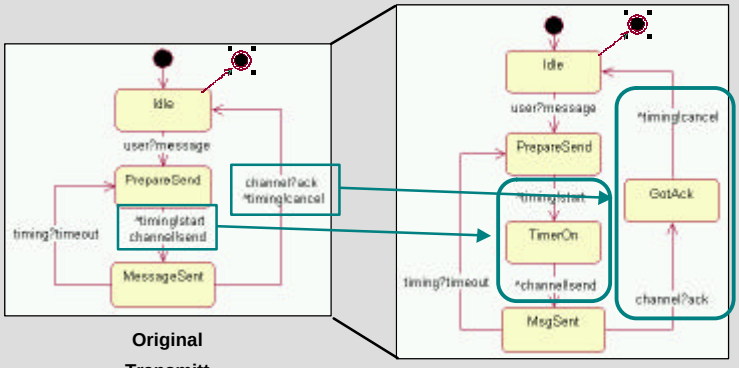


- Focus on integration testing
- Generate test cases to validate component interaction
- Consider subsystem A
- External interfaces:
  - **tuser**
  - **timer**
  - **txport**
- Internal Interfaces:
  - **timing**

Matching Send and Receive Events Between Components



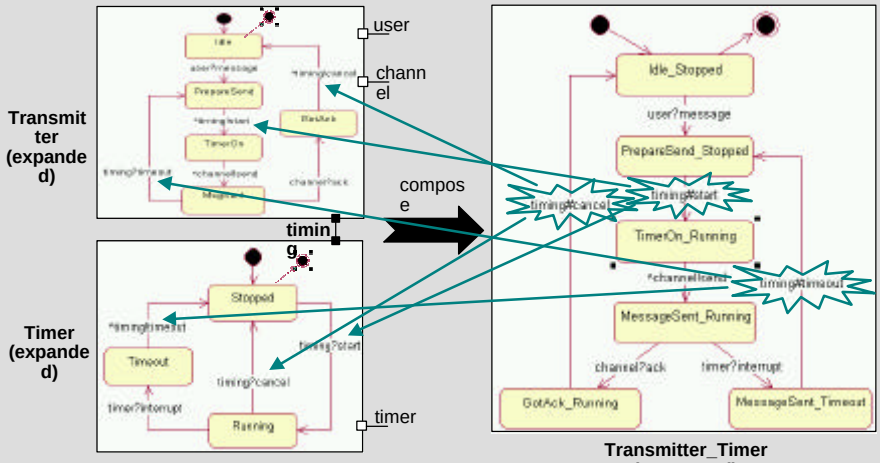
Normalizing the UML-based Models...



Original Transmitter

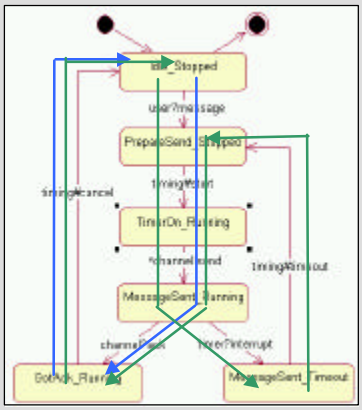
Transmitter (expanded)

Composing a Global Behavioral Model...



Transmitter_Timer (composed)

Test Case Generation



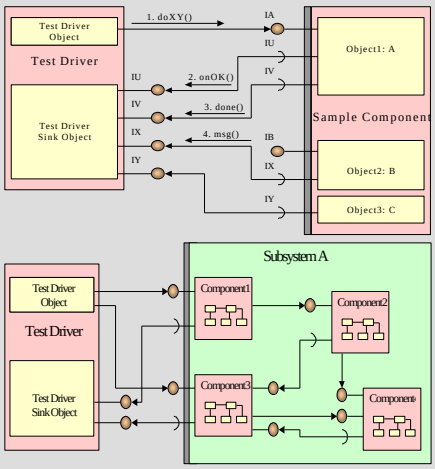
Transmitter_Timer (composed)

```
TEST CASE #1
 * IN  user.message();
 * OUT channel.send
 * IN  channel.ack();
 * IN  user.message();
 * OUT channel.send
 * IN  channel.ack();
TEST CASE #2
 * IN  user.message();
 * OUT channel.send
 * IN  timer.interrupt();
 * OUT channel.send
 * IN  channel.ack();
 * IN  user.message();
 * OUT channel.send
 * IN  channel.ack();
```

Test Cases for Subcomponent

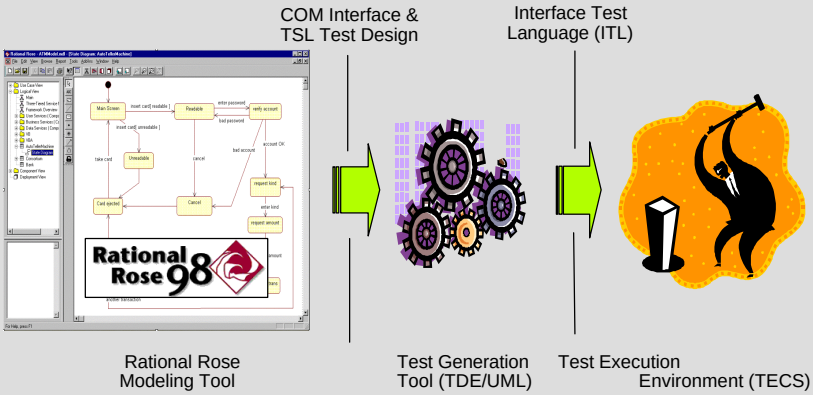
For integration testing: all transitions between components are covered

Test Case Execution

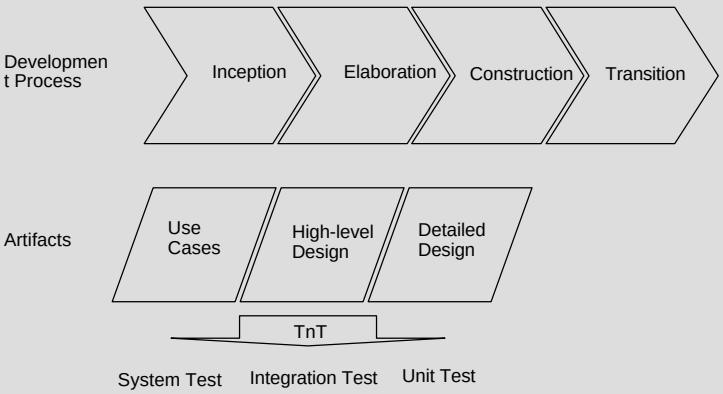


- Executable test driver is created from the generated test cases
- We have specific support for reactive components
- This is achieved through an event pattern matching mechanism
- Supports automated regression testing during unit and integration testing

Implementation : T*n*T = TDE/UML + TECS



Applying T*n*T in the Software Lifecycle



Future Research and Development

- Modeling
  - Support additional UML model elements (e.g. nested states)
  - Examine the issues related to asynchronous communication between components
  - Modeling real-time aspects (e.g. timing constraints)
  - Develop a philosophy for an optimal composition order
- Test Generation
  - Exploit the use of data variations within test generation for integration testing purposes
- Test Execution
  - Focus on a distributed test execution environment

Conclusions

- We provide a UML-based test generation and execution environment using Statecharts
- We focus on unit and integration testing for components
- We support the testing of middleware-based components
- We are starting to apply it within Siemens
- We are continuing to refine and improve T*n*T

QW2000 Paper 6W1

Mr. Anand Sundaram
(RSW Software, Inc.)

Managing E-Business Quality in Internet Time

BACK TO QW2000 PROGRAM

Key Points

- Application scalability has become one of the top concerns most often cited by IT managers. This presentation addresses how to alleviate scalability risk.
- To avoid "scalability shock" load and performance testing should be an integral part of the process of designing, buiilding, and maintaining the web application.
- One must continuously monitor application performance to identify slowdowns and performance degradations before users have a bad experience.

Presentation Abstract

Presentation abstract to be supplied.

About the Author

Speaker bio to be supplied.

BACK TO QW2000 PROGRAM

Managing e-Business Quality in Internet Time

Anand Sundaram, RSW Software

RSW Software
e-Test for e-Business

Introduction

- Fast paced development for internet applications
- E-commerce systems are in a constant state of change
- Typically, many QA activities are skipped due to time constraints
- "Process" is seen as overhead and ignored

No time to do it over, must do it right the first time

RSW Software
e-Test for e-Business

Scope of this Presentation

- Overview of solution
  - Present all aspects of e-Business Quality
  - Prioritize to reflect high ROI for Internet Time'
  - Strong focus on scalability testing
    - As it provides the highest return on investment
    - Represents highest risk if not addressed early
- Intended audience
  - Development managers
  - QA Engineers/managers
  - Operations Engineers

RSW Software
e-Test for e-Business

Project Management and Staffing

- *Managing the development process and environment*
- *Staffing issues*
  - QA departments are typically understaffed and inexperienced
  - Key process responsibilities are often not assigned to individuals
  - Required roles for success of QA
- *Getting QA involved early*
  - Participation in requirements analysis
  - Formal Test Planning
  - Test goals
  - Test approach
  - Test environment creation and maintenance

RSW Software
e-Test for e-Business

Requirements Analysis

- *Benefits of formal system requirements*
  - Capture customer needs and business objectives
  - Keep project aligned with business objectives
  - Ensure the right product is developed
  - Enable efficient and effective testing
- *Prioritizing and costing the requirements*
  - Decide features/functionality to be implemented first
  - Allows for efficient decision making for changes and enhancements

RSW Software
e-Test for e-Business

Requirements Management

- *Benefits of requirements management*
  - Provides a means to rapidly respond to changing needs and objectives while preserving quality
  - Enables efficient impact analysis and defect resolution
  - Enables efficient maintenance of code and test cases
  - Improve organizational communication
- *Requirements management strategy for e-commerce development environments*
  - Link system requirements to business objectives and test cases
  - Use RM to efficiently track and manage change in the development environment

RSW Software
e-Test for e-Business

Architecture Validation and Component Testing

- *Prototyping vs. evolutionary development*
  - Verifying the feasibility of proposed system architecture (Get tools involved early -> must be easy to use)
- Code/Unit Test
  - *Component testing and validation (COM Objects, Java Beans, etc.)*
  - *Configuration Management*
  - Controlling the development, staging, and production environments
    - Full builds vs. individual file updates

RSW Software
e-Test for e-Business

Functional Testing

- *Risk Assessment – prioritization of testing activities*
- *Linking test cases to requirements*
- *Automation of functional testing for dynamic web sites*
  - Determining the scope of automation
  - Efficient creation and maintenance of test scripts

RSW Software
e-Test for e-Business

Performance Testing

- *Planning*
  - Determine user and load profile
  - Determine transactions and scenarios
  - Determine test environment/resource requirements
    - Hardware
    - External tools
    - People
  - System level vs. component level testing
    - Create scripts that isolate system components (web server, application server, WAN link, etc.) to aid in bottleneck ID
    - Create scripts that measure complete business transactions
  - Fail-over testing

RSW Software
e-Test for e-Business

Performance Testing

- *Development*
  - Scripts must be easy to create and maintain due to rapidly release time frames
  - Scripts must be verified in a single and multi-user environment prior to test execution
- *Execution*
  - Execution best practices
  - Test environment
  - Hardware/network monitoring during execution
- *Analysis*
  - Evaluating test results
  - Bottleneck identification

RSW Software
e-Test for e-Business

Regression Testing

- *Strategy for regression testing dynamic systems*
  - Choosing what to include in a regression test (both manual and automated)
- Using requirements management to efficiently maintain regression test suites
- Performance and scalability regression testing
  - Early Benchmarking
  - Cross Benchmark Comparison

RSW Software
e-Test for e-Business

Performance Tuning and Staging

- Staging Environment for tuning application
- Scale of Staging Environment
- Goals
  - Fastest Response Time
  - Best Throughput
  - Best Fail over
- Re-use of load testing scenarios
- Expertise with development environment

RSW Software
e-Test for e-Business

Problem Tracking and Resolution

- Defect Tracking and Resolution
  - *Benefits of formal defect tracking and resolution workflow*
  - *Defect workflow strategy in an e-commerce development environment*
  - Efficient review of reported defects
  - The importance of requirements management in defect resolution
- Efficient impact analysis
- Efficient regression testing

RSW Software
e-Test for e-Business

Post-Deployment Monitoring

- Post Deployment Monitoring
  - *Creating a monitoring strategy*
  - Determining transactions to monitor
  - Determining performance / system statistics of various system tiers to monitor
  - Monitoring from local and/or remote sites
- *Making effective use of monitoring data*
  - Analyzing the data
  - Notifying the appropriate resources of detected problems

RSW Software
e-Test for e-Business

Q&A

- www.rswsoftware.com
- Anand@rswsoftware.com
- 781-993-8509

RSW Software
e-Test for e-Business

QW2000 Paper 6M1

Ms. Lisa Crispin
(TRIP.com)

Guerilla Tool Selection

BACK TO QW2000 PROGRAM

Key Points

- Where to get information about tools
- How to minimize risk in tool selection
- How to best implement a new tool

Presentation Abstract

It's a jungle out there. So many potential problems. So many tools to help avert them. So little time. There are many tool evaluation methodologies around, but who has the time to go through them? Here are some tips to help you slash through the tangle of sales pitches to select tools that work for you. This paper reflects not only my own experience but also that of our development team in selecting and implementing a variety of state-of-the-art tools. This presentation is geared toward organizations building Web and e-commerce applications.

About the Author

I have 18 years experience in the industry with the last 9 in Testing and Quality Assurance. I began my career as a programmer with the University of Texas Data Processing Division, where I helped develop the Online Catalog and circulation systems. My QA experience before my current job was with database, 4GL and client/server software on UNIX, NT and Windows at medium to large software companies “ Software AG (now SAGA), Unidata (now Ardent Software).

I started my current job at TRIP.com in March of 1998 with little knowledge of the Web and no experience working in a startup environment. I was the first test engineer “ no testing had been done previously even though the site had been online more than a year. The challenge of building quality into Web applications while meeting extremely tight development cycles has taught me much. We have grown from a test department of one (me) to seven test engineers, and I have been promoted to 'Quality Boss.' We test state-of-the-art, first-of-their-kind applications such as our intelliTRIP product which uses software from BEA Weblogic, StoryServer, and On Display to query partner airline Web sites for best fares. In addition, we play a key role in finding ways to build quality into applications and prevent errors before coding. To this end, I have been instrumental in implementing a configuration management tool as well as unit testing and memory/performance analysis tools. I work closely with Marketing and Product Management on process definition, formatting of specification documents, and

planning.

Our latest testing challenge has been to move our Web sites to a new hardware and software configuration with two mirrored data centers running load-balanced clustered UNIX servers. TRIP.com has shown its commitment to quality by providing us with a test environment that exactly matches that of production: the same hardware, the same database, and a separate but equal network, so that we can effectively load test.

I have given successful presentations at both local and international user conferences to audiences of up to 60 people. I have many years experience training both technical and end users.

BACK TO QW2000 PROGRAM

Guerilla Tool Selection

Lisa Crispin
Quality Boss, TRIP.com

1

Guerilla Tool Selection

Lessons to be Learned:

- Beginning the search for tools
- Minimizing risk
- Evaluating tools quickly
- Justifying your decision
- Implementing successfully

2

Guerilla Tool Selection

Standard Tool Selection Process

- Detailed requirements definition
- 2 or more months just to select tool
- Do it if you can!

3

Guerilla Tool Selection

Guerilla selection defined:

- You need the tool now!
- You're the only tester
- Can't afford ' shelfware'
- Fast-paced environment (Internet)

4

Guerilla Tool Selection

Hacking your way through the tool jungle

Define your criteria for evaluation…

- **Features**
- **Ease of use**
- **Training requirements**
- **Cost**
- **Supported environments**
- **Tech support**
- **Vendor commitment**

5

Guerilla Tool Selection

Hacking your way through the tool jungle

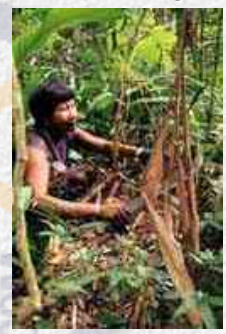
Research resources on the Internet:

- http://www.softwareqatest.com/index.html
- http://www.crl.com/~zallar/testing.html#Kerry
- http://www.iac.honeywell.com/Pub/Tech/CM/CMTools.html#TeamWare
- http://www.methods-tools.com/tools/testing.html

6

Guerilla Tool Selection

Hacking your way through the tool jungle



Ask questions

- Coworkers
- Vendors
- Job candidates
- Newsgroups

Find survey results – does one tool stand out?

7

Guerilla Tool Selection

Scoping likely targets

Dig through the sales material

Is it just advertising, or is concrete technical information provided?

8

Guerilla Tool Selection

Scoping likely targets

Evaluate the vendor

- Is it easy to get a price quote?
- Can salespeople answer technical questions?
- Is this tool their specialty?

9

Guerilla Tool Selection

Scoping likely targets

Get references

Companies you know and respect

Other vendors you are happy with

10

Guerilla Tool Selection

Scoping likely targets

Recruit experts

- Developers, information systems, testers
- Quiz references
- Evaluate tools

11

Guerilla Tool Selection

Avoiding Traps – Evaluating Tools

Install an evaluation copy

- Does the install go smoothly?
- How good is tech support?
- What's the ramp-up time for basics?

12

Guerilla Tool Selection

Avoiding Traps – Evaluating Tools

Calculate costs

- Purchase
- Training
- Implementation

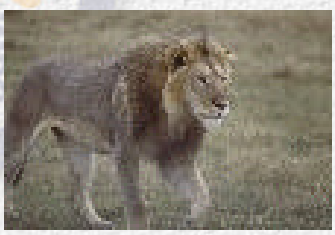
13

Guerilla Tool Selection

Avoiding Traps – Evaluating Tools

Evaluate Risks

- Is it safer to go with a big name?
- Look past bells and whistles
- Low price lowers risk
- Future flexibility



14

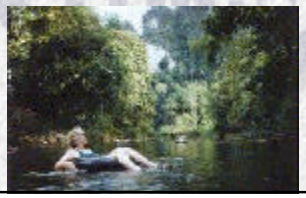
Guerilla Tool Selection

Avoiding Traps – Evaluating Tools

Justify your decision

- Pros and cons – tool types and tools
- Will it take you where you want to go?
- Is the risk acceptable?

Why buy the Titanic if an inner tube will do?



15

Guerilla Tool Selection

Successful Implementation

Identify a champion – examples:

- Customer support manager for defect tracking tool
- Developer for white-box test tool
- Someone who can carry the project!

16

Guerilla Tool Selection

Successful Implementation

Form a team

- Tool champion
- Experts who helped in selection

17

Guerilla Tool Selection

Successful Implementation

Get input

- All key users
- Facilitates buy-in
- Don't lose momentum

18

Guerilla Tool Selection

Successful Implementation

Inform users

- How tool contributes to success
- Mutual benefits

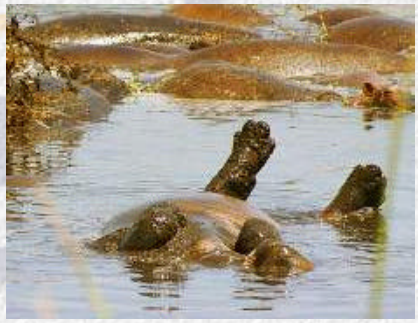
19

Guerilla Tool Selection

Successful Implementation

Warning:

- Implementation takes longer than selection!
- Drive the momentum until it's done.
- THEN you can relax!



20

Guerilla Tool Selection

Lisa Crispin
Quality Boss
TRIP.com

It's a jungle out there!

So many potential problems! So many tools to choose among in your battle to avert those problems! So little time! Tool evaluation methodologies abound. But who has the time for a long, drawn-out process? Here are some tips to help you slash through the tangle of sales pitches so that you can select tools that work for you. This paper reflects the experiences of TRIP.com's QA and Development teams in selecting and implementing a variety of state-of-the-art tools.

Lessons to be Learned:

- Where to start in the search for tools
- How to minimize risk in tool selection
- How to evaluate tools quickly
- What to think about when justifying your decision
- How to successfully implement a tool

Guerilla Tool Selection Defined

Much has been written and presented on the best practices for selecting tools, such as for automated testing. When I worked for midsized and large database software companies, we often spent a month just defining and prioritizing our requirements. Then, we spent even more time examining tool functionality, performing pretrial demos, and completing thirty-day trials on multiple candidates. After all of this, we felt assured that we were choosing appropriate tools.

On two occasions since I have been employed by TRIP.com, I have needed to buy and implement tools and in "Web time." I'll use a couple of examples to illustrate our "guerilla tool selection" technique:

- As the first (and at the time, only) test engineer for TRIP.com, I desperately needed both a tracking tool and an automated test tool. It took me only a month to get both, and two years later we're still happy with both.
- As our development organization grew, we experienced severe problems that could have been prevented by proper configuration management (CM). We needed a new CM tool immediately. We pinned down our selection in a matter of days. Implementation was tougher— more on that later! Again, the tool we selected is serving us well.

If you have enough infrastructure and time, by all means use best practices and take your time to select your tools. If your risk of not having any tool is greater than the risk of selecting a less than optimal tool, and you don't have a lot of resources to spend on selection— go guerilla! In hindsight, I don't believe we would have changed our selections if we'd had more time and people to spend on the selection process. When you're facing down a dangerous beast, you can make the right split-second decision!

Here's the procedure I've used to quickly hunt down tools that have worked well for us.

Disclaimer – the tools I use in my examples work well for TRIP.com. You must choose tools that work for YOU. Also, the prices I quote in my examples may no longer be current. I provide them just to support our decision process.

I. Hacking Your Way Through the Tool Jungle

Where to start? When it comes to tools that aid software development and testing, you can't see the trees for the rainforest! In my first days at TRIP.com, I needed to select an automated test tool for the Web but

knew nothing about Web testing. That was limiting! When we needed a new CM tool, my CM experience was not extensive, and I didn't have any idea which tools to look at.

- **Define** your requirements. You need some criteria for evaluating the tools. Don't spend a month on it, but think about the salient points. You will refine this as you go through your search and learn more about the methodologies and available tools. Some criteria I used for my final decision on an automated test tool include the following:
    - Is the tool easy to learn? How much training is required?
    - How much does it cost?
    - Can scripts run unattended? Are unexpected results handled gracefully?
    - Is the scripting language easy to learn and robust?
    - Is the tool GUI or HTTP?
    - How well does the tool emulate actual users?
    - Does the tool include capture/playback, link checking, load testing?
    - What browsers are supported?
    - How good is the tech support?
    - How committed is the vendor to Web-testing tools (as opposed to client-server or other types)

  Other criteria were not important to me. For example, I didn't care if the tool had an integrated defect-tracking tool because I already had one. I decided that I could live without an integrated test plan repository. These criteria might be essential for a different organization. Consider what you want and record them in a living document that changes as you learn.
- **Research** resources on the Web by using an engine such as AltaVista. The following URLs were invaluable to me for locating both automated test and CM tools:
  •**http://www.softwareqatest.com/index.html**
  •**http://www.crl.com/~zallar/testing.html#Kerry**
  •**http://www.iac.honeywell.com/Pub/Tech/CM/CMTools.html#TeamWare**
  •**http://www.methods-tools.com/tools/testing.html**
- **Ask questions** of coworkers and people you know in similar organizations. Vendors you already use can be a good source of information – if they're developing software, they need a lot of the same tools you do. I've even gotten input from developer and test candidates while I'm interviewing them. When we were researching CM tools, it was easy to ask "Have you had any experience with CM tools? Which ones do you like? Why?" Even if you don't hire the candidate, you could get a helpful tip!

II. Scoping Likely Targets

If you're in a hurry, you need a narrow field to choose from. How to quickly cut the most likely beasts out of the herd of vendors? These tactics worked for me.

- **Find survey results** – Does one tool stand out? When I first started searching for CM tools (which at the time we needed 'yesterday') I had only worked with one Unix-based tool and was concerned that it was overkill for our needs. In my searches on the Web, I found a document of survey results (http://www.iac.honeywell.com/Pub/Tech/CM/CMSurveyRes.html) in which one tool, Perforce, consistently received high ratings. That prompted me to research the vendor further.
- **Dig through the sales material** – Website or printed – is it just advertising or is concrete technical information provided? For example – does their Website offer white papers and feature-by-feature comparisons with competitor products? These documents can help your search – they can educate you about features you may not have considered and turn you on to more potential vendors. My experience is that vendors who want to educate you – even if it might lead you to another product – have a good understanding of the problem and offer a well-thought-out solution. When I was Web-impaired and started looking for Web testing tools, it was helpful to find a comparison of GUI versus HTTP methodologies – I didn't even know what HTTP was at the time!
- **Evaluate the vendor –** You're not just buying software – you're establishing a long-term relationship with a company on which you will depend. Consider these criteria:
    - Is it easy to get a price quote? Personally, I prefer to work with vendors who are straightforward rather than the ones who behave like car dealers. I like to see the prices stated clearly on the Website. I've found those vendors to be straightforward in all my dealings with them. This is important when you have a problem!

- Can the salespeople answer technical questions? I like a company that hires salespeople with some technical knowledge and trains them – it gives me more confidence that there is technical merit in their product and that their tech support people also know what they're doing.
- The size and way of doing business of the company influences me as well. Working for a Web startup, I have preferred to do business with other companies who either have Web-based products or do their business mainly on the Web; companies who are young and dynamic and growing, just like TRIP.com. For a large, mature corporation, a vendor with similar organization may be a better fit.
- Is the particular tool you need a specialty of the vendor? When I first started at TRIP.com, some GUI test tool vendors had kluged together a 'Web testing tool' but these were not robust.

- **Get references –** Try to find companies you know and respect to vouch for tools they use. One source can be other vendors whose products have satisfied you. When I was looking into Perforce for CM, I noticed on their Web site that BEA Weblogic is their customer. Because we use BEA Weblogic's tools and like their quality, we called our contacts to get access to their developers and grilled them about Perforce. Their answers influenced our buying decision. This provided a quick way to get reliable information— perfect for the guerilla tool selector.
- **Recruit experts** in your company to help quiz references and evaluate the tools. These might be developers, testers, information systems staff— people who have experience and/or a vested interest in the tool you are purchasing. I snagged a couple of senior developers who were experienced in CM to help evaluate CM tools. They're the main users of the tool and I also couldn't implement the tool without them— more on that later. After I narrowed the search down, they were happy to interview references.

III. Avoiding Traps – Evaluating Your Selections

A few likely products have caught your eye. Now you need to figure out fast if one will work for you. All the paths through the jungle look both scary and inviting— which ones have well-disguised traps and which are safe?

- **Install an evaluation copy.** Does the install go smoothly? How good is the support if you have problems or questions? What's the ramp-up time to learn basic functionality? If you're the lone tester in your organization, you don't have time to go to class, you can't afford install hassles and you'd better be getting top-notch customer support. If the vendor doesn't care about you when you are running a trial, they sure won't pay any attention to you once they have your money. When I worked in a large test organization, we selected a couple of test tools that required weeks of training and these tools worked out very well. When I came to TRIP.com as a test department of one, I went with a tool I could figure out and start using for real testing in just a few hours.
- **Calculate the cost, including training and implementation.** Do you get what you pay for? I don't think this conventional wisdom applies in the software business. Some companies have to price their products high to pay for R&D and/or marketing costs. If a few geniuses came up with a good product and don't spend much on marketing, they may offer their tool at a very attractive price. Don't forget to estimate implementation cost: if it takes five people three months to implement the tool, it's not cheap at any price (although it still could be worthwhile). The test tool we have used successfully for two years, WebART, costs a ridiculous-sounding $825 a seat, including unlimited simulated users for load testing. I didn't need to buy any up-front training to use it and implementation was a breeze. Unless price is a criterion due to a limited budget, don't let it cloud your evaluation— judge each product on its merits.
- **Evaluate risks.** Faced with those forks in the trail, you need to consider many factors.
  - Behemoths like IBM and Microsoft have a huge competitive advantage. Nobody will criticize you for choosing a product that millions of other people use— even if the products aren't a good fit for you! If you want to be safe from criticism, go with the big name. If you want a tool that really works for you and doesn't turn into shelfware, choose the tool you need. It doesn't matter if nobody has ever heard of the vendor if you can satisfy yourself that they are reliable. People look at me like I have two heads when they ask me what test tool I use and I reply "WebART" – nobody's ever heard of it. It's produced by OCLC Inc., a non-profit, which provides services to libraries. I worked with OCLC's products back when I was helping develop an online library

catalog, which gave me a level of comfort in choosing them. I probably would have gone with their tool even if I wasnt familiar with them, based on what I learned in my evaluation.

- Look past the bells and whistles. Pretty packaging doesnt make a tool useful. Stick to the bottom line: does the tool report the information you need?
- Low cost obviously lowers risk— an important although not essential factor in guerilla tool selection. By keeping costs low, you avoid the whole Vietnam syndrome: "we've put so much money into this we cant stop now." The CM tool we use, Perforce, cost $600 a seat. We didnt have much time to evaluate it, but we decided that if purchasing it turned out to be a mistake, at least it wasnt an expensive mistake. Our defect tracking system, TeamTrack, was only $400 a seat, and also offered a limited license at about half the cost that worked for many of our users. We didnt buy any of these products solely based on price, but we felt more comfortable about the amount of risk involved than if we had been buying tools costing $15,000, $50,000 or more.
- Low cost also offers future flexibility. When my test team grew, and I felt I needed some help making better use of the tool to automate our testing, I had plenty of money in the budget to pay for WebART training and consulting. If our needs should change, or if some new fabulous test tool comes out that takes our breath away, WebART is inexpensive enough that we could keep using the tests already developed there and not feel bad about buying another tool for future test automation.

- **List Pros and Cons.** OK, this is a no-brainer, but dont discount the importance of making a list of pros and cons of different **types** of tools. For example, before selecting a Web-testing tool, I had to decide if an HTTP or a GUI tool would fit our needs better. Once Id narrowed down the features I wanted, it was easier to compare the advantages and disadvantages of various tools.
- **Justify your decision.** Will the tool take you where you want to go? The Titanic was luxurious, but kayaks work really well for the Inuit. After all, maybe all you need is a raft. Are you comfortable with the level of risk associated with your choice? I almost didnt buy WebART because I felt a little foolish choosing such an inexpensive tool. Yet when I listed the pros and cons associated with each tool I had evaluated, it came out on top.

IV. Rallying the Troops – Tips for Successful Implementation

Implementation is by far the most challenging aspect of acquiring a new tool. If you used my guerilla tactics, youre clearly in a hurry. Dont slow down now. As Tarzan rallied the animals of the jungle to combat his foes, you must rally support among potential users of the tool.

- **Identify** a champion for the tool. For example, a customer support manager could take ownership of a defect-tracking system; a developer might be motivated to implement a CM or white-box testing tool. Choose someone with a vested interest in the tool and who can carry the project forward. If you cant identify someone capable, you may have to look to a new hire to contribute the expertise. One of our developers installed our CM tool, did the basic setup and admin so that other teams could start using it, but we couldnt fully implement until we hired a QA engineer who handles CM as part of his daily tasks.
- **Form a team** with a champion to finalize the implementation plan. This team should include the experts who helped you in the selection process. For example, the architect, analyst, and lead developers could plan implementation of a visual modeling tool.
- **Get input** from all key users on the best way to implement the tool. This will facilitate buy-in. I organized meetings of each development team along with the product champion to decide how to best use the CM tool. A couple of rounds of these were needed to keep the momentum going.
- **Inform** users of what the tool will do for them and for the company. Let them know how the tool will make them look good. They need to see the benefits of their using the tool and helping you get others to use it.
- **Drive the momentum** until everyone who should be using the tool accepts it as a part of his or her daily jobs. If you made a good choice, soon they wont be able to imagine life without it and you can relax and enjoy the beauty of the rainforest!

QW2000 QuickStart 6Q

Mr. Tobias G. Mayer
(eValid, Inc.)

WebSite Testing

BACK TO QW2000 PROGRAM

Key Points

- Websites are very similar to ordinary pieces of application software, but different in many important ways.
- Effective testing of websites is best accomplished with a "Test Enabled Web Browser", particularly if Object Mode operation is required.
- Different ways of testing websites will be explored.

Presentation Abstract

- Testing a website BEFORE it is uploaded
- Testing a website AFTER it is uploaded
- Website Testing Strategies

About the Speaker

Tobias Mayer is a senior software engineer at Software Research, Inc. He is reponsible for the main design and implementation of the "eValid" Web Test engine. Tobias has a (UK) BSc from South Bank University, London. He is a member of, and OO Metrics consultant to, the Center for Systems & Software Engineering (CSSE) at South Bank University. Tobias has presented and published a number of papers on OO metrics, including papers at IEEE 'TOOLS' 1999 and British Computer Society 'SQM' 1999.

Tobias had a number of years' experience of facilitaing group work/team building seminars before moving into the software development field. The QW 2000 workshop is intended to be an interactive experience, rather than a lecture. Participants will be facilitated to share ideas, explore pertinent issues, and learn from one another.

BACK TO QW2000 PROGRAM

Tobias Mayer
eValid, Inc.
email: mayer@soft.com

WebSite Testing

QuickStart, 6Q

Summary of presentation

1. Testing a site prior to uploading
   (~15 mins)
2. Testing a site after uploading
   (~25 mins)
3. Testing Strategies …
   (~35 mins)
4. Questions/Comments...
   (~5-15 mins)

Running time = 80-90 minutes

Stage 1: Before the upload

- What can be tested?
  - » HTML
  - » Links
  - » Applets
  - » Scripts
  - » Images
- Other Considerations...

Stage 1: Before the upload

- What can be tested?
  - **HTML**
    - » Does it meet company/self-imposed formatting?
    - » Use of WebLint, HTMLTidy, etc.
    - » Spell Checkers
    - » Can HTML be tested like regular source code?

Stage 1: Before the upload

- What can be tested?
  - **Scripts**
    - » Again, do they meet company/self-imposed formatting?
    - » Syntax...
    - » Logic...

Stage 1: Before the upload

- What can be tested?
  - **Links**
    - » do they go where they are supposed to go?
    - » Can we actually test this prior to upload?
      - Own site -v- some other site...
    - » What kind of error messages can we get?

Stage 1: Before the upload

- What can be tested?
  - **Applets** (and other embedded executable objects)
    - » To what extent does an object interact with the HTML?
    - » Do objects execute correctly -
      - As independent entities
      - In relation to the Web site
    - » Executable objects should be fully tested prior to their inclusion on the site.

Stage 1: Before the upload

- What can be tested?
  - **Images**
    - » Do all images (gifs, jpegs, etc) exist at the correct locations?
    - » Does each image have the Width & Height values set?
    - » Check image sizes (i.e. byte size)
      - Is an image going to seriously affect the download time?
      - What methods can be used to reduce an image's size?

Stage 1: Before the upload

- Other Considerations…
  - Where will the site be positioned?
    - » Internet
    - » Intranet
    - » Local, browser-based app?
  - Other considerations …?

Stage 2: After the upload

- User Interactions
- Static / Dynamic Pages
- Test the site on a regular basis
- Use Automated testing

Stage 2: After the upload

- User Interactions
  - What will the User try to do?
  - Can the developer anticipate ***everything?***
    - » Every possible user interaction?
    - » Are users always smart???
  - How can the developer deal with "user stupidity?"

Stage 2: After the upload

- Static / Dynamic Pages
  - What is the difference?
  - Which is harder to test. Why?
  - Can dynamic pages actually be tested?
    - » How. What can be validated?
    - » How often?
    - » What can we learn?

Stage 2: After the upload

- Test the site on a regular basis
  - Regression Testing
  - What else is there?

Stage 2: After the upload

- Use Automated testing
  - Manual Testing:
    - » How many testers will be employed?
    - » How useful are they?
    - » What can they do that a robot-tester cannot do?

Stage 3: Testing Strategies

- Server-Side Testing
- Client-Side Testing
  - Browser-Level Testing
  - Operating System-Level Testing
- What else is there…?

Stage 3: Testing Strategies

- Server-Side Testing
  - Load Testing
    - » How many hits per (sec/millisec?)
    - » How many KB delivered per (sec/millisec?)
  - Site Validation
    - » Can it be done at this level?

Stage 3: Testing Strategies

- Client-Side Testing
  - How many times can you hit the server?
    - » What does this tell you?
  - Validations:
    - Visible Text
    - Table Cells
    - Images,
    - etc...

Stage 3: Testing Strategies

- Client-Side Testing
  - Static Sites
    - » Nothing changes from day to day unless the developer explicitly changes it.
  - Dynamic Sites
    - » Stuff changes all the time. Once created the developer has little, or no, control...

Stage 3: Testing Strategies

- Client-Side Testing
  - What else can be tested from the client side?
  - Is Client-Side Testing the best strategy? If not, why not?

Stage 4: Questions/Comments

- Anything to ask?
- Anything to add?

QW2000 Vendor Technical Paper 6V2

Michael Aivazis
(ParaSoft Corporation)

Methods for Effective Unit Testing

Effective C/C++ Unit Testing

An overview

Dr. Michael Aivazis
Director of Technology
***Para**Soft* Corporation
mga@parasoft.com

Why Modify Your Software Development Process

- Current industry trends are forcing companies to turn out applications in 2-3 months
- If you don't have an effective development process, you will not survive

Goals of This Presentation

- Demonstrate the necessity of configuring a development process that effectively prevents errors
- Describe what such a development process should look like
- Discuss how tools fit into this development process
  - *Describe what commercial and public domain tools help you perform each step or assess readiness to progress,*
  - *Explain how to effectively configure public domain tools*

3

Error Prevention

- Errors are not inevitable
- Errors occur because of bad coding and development practices
- Examples in C++ and Java
- Preventing errors results in a higher-quality product than introducing errors, then trying to find and remove them

4

Example 1

IBM Study

"Projects that aim from the beginning at achieving the shortest possible schedules regardless of quality considerations tend to have fairly high frequencies of both schedule and cost overruns. Software projects that aim initially at achieving the highest possible levels of quality and reliability tend to have the best schedule adherence records, the highest productivity, and even the best marketplace success."*

* Jones, Capers. Programming Productivity. New York: McGraw-Hill, 1986.

5

Example 2

- W. Edwards Deming: Use statistical quality control to build quality into product by implementing a process that prevented errors, rather than by focusing on the product itself (Total Quality Control)
- In the 1970s, American auto manufacturers' failure to focus on process and error prevention prevented them from improving product quality
  - *They assumed defects were inevitable and did not try to stop causing defects*
  - *Instead, they tried to remove existing defects*
  - *Manufacturers failed to significantly reduce the number of product defects*

6

Development Process

- Customers
- Development teams
- Methodology
- Practices
- Tools

Methodologies

- Waterfall
- Spiral
- RAD
- eXtreme programming

Practices

- Certain practices are universally acknowledged as beneficial, regardless of development methodology
- Code construction
  - *Coding standards, code reviews, unit and application testing*
  - *Debugging support: firewalls, debugging information and debugging methods*
- Source monitoring
  - *Source control, automated and regular builds, bug tracking, automatic error detection, metrics*

ParaSoft

Coding standards

- Standard code formatting
- Standard coding constructs
  - *beyond syntactically permissible*
  - *standardized practices*
- Promote:
  - *Smaller chance of introducing defects*
  - *Smaller chance that defect will go unnoticed during code review*
  - *Enhanced communication through the code*
  - *Increased efficiency*

ParaSoft

Source control

- Central repository for the entire source base
- Provide a record of the evolution of the source base
  - *who, when, why*
- Lessen the risk of change
  - *recovering older, more stable version*
  - *cheaper to try different approaches*
- Facilitate group access to a common source base
  - *Network, remote access*
- Source tagging
  - *Named collection of sources*

11

Automated builds

- Minimize the overhead in assembling the application pieces
- Identify certain classes of interface errors
- Promote frequent builds of the entire application
- Provide the basis for automated regular testing at the application level

12

Unit testing

- Write the test *before* the code!
- Use the tests to document the behavior of the object
- Guaranteed maintenance of the test suite along with the code
- Thoroughly test the entire publicly accessible interface
- Test the boundary conditions:
  - *Out of range arguments, typos, NULL objects, numbers that are too big or too small*

13

Unit testing - continued

- Provides
  - *Increased confidence that the unit is performing as expected*
  - *Continuous assessment of the quality of the code*
  - *Early detection of incompatible changes*
- Helps minimize the risk of change by providing a means of verifying correctness and detecting fault

14

Application testing

- Functional tests document the expected application behavior
- Necessary for assessing
  - *Overall application behavior*
  - *Unit integration*
  - *Performance*
  - *Resource requirements*

15

Regression tests

- Non-trivial applications have non-local defects
- Repairing a defect sometimes disturbs a sizeable portion of the system
- Regression tests
  - *provide a warning system against non-local defects*
  - *Shield against poor understanding of application behavior by the maintainer*
- A defect is not repaired until a test case that detects its presence is placed in the regression test suite

16

Additional testing

- In-house testing
- Beta cycles
- Release candidates

17

Bug tracking

- Ensure appropriate people are notified of problems
- Facilitate bug reporting
- Correlate bugs to source versions
- Facilitate tracking of individual problems
- Defect counts and defect rates are fundamental quality metrics

18

Code reviews

- Promote collective ownership of the code
- Distributes the knowledge about the implementation details to the team
- Allow more experienced developers to share know-how
- But, most importantly, is a very effective way to detect defects

19

Debugging support

- Add logic for pre-conditions and post-conditions
- Retain the code fragments used to exercise the code during debugging sessions
- Write routines specifically for use during debugging:
  - *Expensive data structure validations*
  - *Multi-object interrelations*
  - *Detailed examination of memory owned by an object*

20

Automatic error detection

- Simplest way to detect defects
- Works well with automatic application builds and test suites
- Some tools can generate test cases automatically

Coverage analysis

- Measures the fraction of the application code that has been exercised
- Uncovers certain types of defects
- Provides a metric of the thoroughness of test suites

Unit Testing Tools

Types of tools available

- Integrated
- Task-specific
- Perform black-box, white-box, and/or regression testing
- Various degrees of automation

23

Unit Testing Tools

What to look for:

- *Ease of use*
- *As much automation as possible*
  - *Builds harness automatically*
  - *Creates test cases automatically*
  - *Automatically generates stubs*
- *Customizable*
- *Variety of tests performed*
- *Coverage of automatic tests*

24

Conclusions

- Survival requires a development process that promotes shipping reliable applications as rapidly as possible
- Tools – both freely available and commercial– are critical to maintaining such a development process
- As the confidence of tool vendors in Linux increases, more specialized development tools should become available

Methods for Effective C/C++ Unit Testing

by Michael Aivazis
ParaSoft Corporation

Unit testing--testing at the class or object level--is a much esteemed yet rarely practiced testing technique. Although most developers and managers have heard that unit testing can result in robust code and reduced debugging time, they often shy away from the technique because of the time and effort required. These developers think of class testing as a sort of "golden rule"--a form of testing they would perform in an ideal world, if only they had the time. However, there are practical methods of performing unit testing so that the benefits vastly outweigh the initial investment of resources.

Unit testing uncovers many errors that would be difficult--if not impossible--to find and fix if testing were delayed until integration. The basic principle here is simple; unit testing makes error detection easier because it pulls each class out of the application and tests it separately from the rest of the application. Isolating each class brings you closer to each error, which increases your odds of finding the errors.

When you compare unit testing to application-level testing, consider this metaphor: Finding an error with application-level testing is like standing at a pool table with all the balls in the triangle, and having to use a cue ball to hit the ball in the middle of the triangle into a particular pocket with one stroke. Finding an error with unit testing, on the other hand, would be like trying to hit one loose billiard ball into a particular pocket with a single stroke--a far simpler task.

The first integral part of unit testing is to create scaffolding for the class you want to test. First and foremost, the scaffolding should be an error-free class that runs your class. Make sure your scaffolding is error-free, or you'll run the risk of causing additional errors in your class. You should also take care to create scaffolding that thoroughly exercises your class and uncovers as many errors as possible.

To help illustrate the process of creating scaffolding, let's look at a coding example. Suppose we want to test a class that calculates the balance of a bank account. The `BankAccount` class below contains a method `void deposit`, which increases the balance, a method `void withdraw`, which decreases the balance, a method `int balance`, which returns the current balance, and the constructor `BankAccount`. We also use the class `BankException`, which allows us to process exceptions:

```
#include "BankException.h"
class BankAccount
{
public:
BankAccount() : _balance(0) {}
int balance() { return _balance; }

void deposit (int amount) {
if (amount < 0) {
throw BankException("invalid deposit amount");
}
_balance += amount;
return;
```

```
}
void withdraw (int amount) {
if (amount < 0) {
throw BankException("invalid withdraw amount");
}
if (_balance - amount < 0) {
throw BankException("maximum withdraw amount is exceeded");
}
_balance -= amount;
return;
}
private:
int _balance;
};
class BankException {
private:
const char *_message;
public:
BankException(const char *message) : _message(message) {}
const char *message() { return _message; }
};
```

We want this class to be able to deposit and withdraw money and to calculate the proper balance. Our first step in unit testing will be to build test scaffolding around the class. Our first step in scaffolding is to write a class that contains the method `test`. We'll use `test` as the place from which we call different methods of the class we're testing.

In this case, we will call our scaffolding class `BankAccountTest`. As you can see below, we are testing to see if our class actually handles deposits and withdrawals correctly. We use our scaffolding class to make two deposits and one withdrawal, and then we check the balance:

```
#include "BankAccount.h"
#include <iostream>
using namespace std;
class BankAccountTest {
public:

static void test() {
BankAccount account;
account.deposit(20);
account.deposit(30);
account.withdraw(7);
if (account.balance() == 43) {
cout << "Test passed" << endl;
} else {
cout << "Test failed" << endl;
}
}
};
int main()
{
BankAccountTest::test();
return 0;
}
```

After running this test, our balance comes up correctly as $43 and we read the message "Test passed". Here we have taken a common-sense approach to testing; we have tested the simplest

functionality first and will now test how the class handles special cases, such as a balance of $0 or a negative balance.

But we also need to make sure that our class will perform properly under unusual circumstances. Bank accounts must be able to handle situations in which someone withdraws the entire balance or tries to withdraw more money than they have in the account. The scaffolding class below will test how our `BankAccount` class handles these situations:

```
#include "BankAccount.h"
#include <iostream>
using namespace std;
class BankAccountTest {
public:
static void test() {
BankAccount account;
account.deposit(20);
account.deposit(30);
account.withdraw(7);
account.withdraw(43);
if (account.balance() == 0) {
cout << "Your balance is 0" << endl;
} else {
cout << "Error!" << endl;
}

try {
cout << "withdrawing $3" << endl;
account.withdraw(3);
cout << "return from withdraw" << endl;
}
catch(BankException) {
cout << "catching BankException" << endl;
}
cout << "done " << endl;
}
};
int main()
{
BankAccountTest::test();
return 0;
}
```

In the code above, we brought our balance up to $50, withdrew $7 and $43 to bring our balance back down to 0, and then instructed the class to print one of two messages. We then tried to withdraw another $3, but the class caught an exception. Our printouts tell us that our class can handle a balance of 0 and that it will prevent us from withdrawing more money than we have available:

```
Your balance is 0
withdrawing $3
catching BankException
done
```

Consider the next example, which contains a `LinkedList`. This is a simple `LinkedList` that is only linked in one direction. This `LinkedList` contains the variable `value`, which stores the value of the LinkedList, and the variable `next`, which represents the next element. Each element

in the list has a container that contains a value, and a link to the next element. You can see that we also create and print the `LinkedList`, and that we have created a method called `test` that is sensitive to one element in the `LinkedList`:

```
class LinkedList
{
public:
LinkedList(int value, LinkedList * next):
_value(value), _next(next)
{}
static void print(LinkedList * list) {
while (list) {
std::cout << list->_value << std::endl;
list = list->_next;
}

return;
}
static void test(LinkedList * list) {
if (!list->_next) {
return;
}
if (!list->_next->_next) {
return;
}
if (list->_value == 21342
&& list->_next->_value == 8765
&& list->_next->_next->_value == -1000) {
int * null = 0;
std::cout << *null << std::endl;
}
return;
}
private:
int _value;
LinkedList * _next;
};
```

To test our class, we must create and print the `LinkedList`, then call the `test`. We can jump from one container to the next, in much the same manner as one would jump from one car to the next on a train, and we have specified that each element must be linked to the element that is one integer greater (represented by `i+1` in the code):

```
#include <iostream>
#include "LinkedList.h"
int main()
{
// Create
LinkedList * list = new LinkedList(0, 0);
for (int i = 0; i < 10; ++i) {
list = new LinkedList(i+1, list);
}
// Test
LinkedList::test(list);
LinkedList::print(list);
return 0;
}
```

When we run this test case, we get the following output:

```
10
9
8
7
6
5
4
3
2
1
0
```

We expected to receive this output because we have linked our integers in ascending order. Our scaffolding does not throw an exception. So we're done...right? Wrong. We still haven't had a chance to get into our code because one of our values is never what it is supposed to be. It just so happens that there are hidden errors in our class.

These hidden errors raise the question, "How do we know when we're finished with unit testing?" The goal in unit testing is 100% coverage. We need to dig into each method, find the exceptions, repair them, and then start the process over again. Only when each class comes up clean will we be finished.

To that end, we must test our `LinkedList` again. We need to have a very specific set of elements in the `LinkedList` in order to find a mistake, and we need to create a set of `LinkedList`s that contain these elements. We have intentionally written the following example in a peculiar manner to illustrate the difficulty of finding the error here. To find the hidden error, we need to write some new scaffolding that will look for the specific error we want to find:

```
#include <iostream>
#include "LinkedList.h"
int main()
{
// Create
LinkedList * list;
list = new LinkedList(0, 0);
list = new LinkedList(-1000, list);
list = new LinkedList(8765, list);
list = new LinkedList(21342, list);
// Test
LinkedList::test(list);
LinkedList::print(list);
return 0;
}
```

There was a piece of code we hadn't tested, but by writing the correct piece of scaffolding above, we finally got the exception we were looking for. Actually, you can see just by glancing at the original code that we call for an element [7], but there is no 7th element. If we had never bothered to execute this segment of the code, we would have seen the error turn up later in the larger application, where it would have been more costly to fix.

Now that we have performed basic testing and special case testing on our classes, we can move on to the next step in unit testing. If the class under test references unavailable or inaccessible external resources (such as other files, CORBA objects, or databases), we must create stubs so

that we can test interactions between the class and the external resource. Stubs should mimic the action of the actual resource that is being referenced; the most effective stubs thoroughly test the interaction without causing any extraneous tests to be performed.

It would be impossible to overstate the importance of specialized test cases in unit testing. To thoroughly test all aspects of the class, a test suite should be composed of three types of test cases: black-box, white-box, and regression. Black-box test cases should test functionality; these test cases should be based on the specification and should be written as the code is being developed. White-box test cases should test construction and robustness; to do so, they should achieve the fullest possible coverage of the methods and provide a wide variety of inputs. Regression test cases test that modifications correct previous errors and do not introduce new errors. Every time you modify your code, you should run your code against your regression test suite to make sure that your code has not "regressed" into previous errors. When you find an error, you can write a test case against it and add it to your suite. As you develop your application, your regression testing suite will grow. Previously-run white-box and black-box test cases can be re-used as regression test cases.

Incidentally, there is no limit on regression testing. The more you perform it, the greater the benefits for your code. You can even use a script to automatically pull up a class, compile it, and run it against your test suite. You'll get a clear idea of what is going on with the code, and you don't even need to be in the office when the testing occurs. As soon as you have created a harness, stubs, and test cases, you can run the test cases and then flag exceptions and failed test cases. You should correct errors and then run the class through the same test suite to verify that modifications corrected existing problems and did not introduce new problems.

Fortunately for developers, there are now ways to automate many aspects of unit testing. Developers can use scripts, cron jobs, Windows scheduling programs, and unit testing tools to automate aspects of unit testing, and, by doing so, gain a practical way to incorporate unit testing into even the most compressed development schedules.

Michael Aivazis, Ph.D., is Director of Technology at ParaSoft. You can reach him at mga@parasoft.com.

QW2000 Paper 6T2

Ms. Elisabeth Hendrickson
(Aveo Inc.)

Quality in an Application Service Provider (ASP) Environment

BACK TO QW2000 PROGRAM

Key Points

- The difference between developing software at an ASP and developing software in a more traditional independent software vendor.
- How to identify and achieve the key quality criteria for an ASP environment.
- How to focus software development efforts, including testing, in an ASP environment.

Presentation Abstract

In the last year, the term "ASP," or "Application Service Provider," has hit buzzword status. A variety of companies have announced that they are now Application Service Providers—meaning that instead of selling software, they sell access to servers running that software.

The challenges inherent in developing software in an ASP environment are different from developing software for sale:

- The primary "customer" of the software is in-house: in an ASP, the people who deploy and maintain the system are internal; the people who access the system are external.
- The software is not the business—it supports the business.
- Behind the scenes chaos is acceptable as long as the people who use the service perceive it as reliable and steady.
- It is the quality of the service that matters. The quality of the software supports—but is not synonymous with—the quality of the service.

This talk focuses on software quality in an ASP environment, how the quality factors for software in an ASP environment differ from software quality at a more traditional software vendor, and how to achieve that quality.

About the Author

Elisabeth Hendrickson is the Director of Quality Engineering at Aveo Inc., an Application Service Provider. Aveo Inc. offers Attune, a pre-emptive technical support service whose mission is to help companies communicate the right information to the right customer at the right time. Prior to joining

Aveo, Elisabeth was the founder of Quality Tree Consulting, a software quality assurance consulting firm. As a consultant, Elisabeth provided services to Application Service Providers as well as more traditional independent software vendors.

BACK TO QW2000 PROGRAM

Quality in an ASP Environment

It Really is Different from Developing Software for the Enterprise

What's Different About an ASP

- The software is not the business— it supports the business.
- Simplifications: control over server configurations, internal people are the customers.
- Greater complexity: need to support multiple customers, scalability, greater visibility into state of service, better measurability, two different kinds of testing for client v. server.

Critical Quality Factors

- Reliability and speed of the service to the user
- Scalability of the service
- Availability of data to the customer
- Quality of any client software, including web sites, that a user or customer must use to get to the service

Clients v. Servers

- Don't hold your client and server to the same quality criteria.
  - Hold the server to high standards for flexibility of implementation, scalability, and performance.
  - Hold the client to high standards for reliability, compatibility, and fault tolerance.

Designing for Quality: the Server

- Is the service real time or asynchronous?
- How can multiple customers co-exist on a single server?
- How will the software deployment scale as demand for the service for a single customer grows?
- How will your administrators be able to tell when the server software is not responding?
- How will the administrators deploy and administer the software?

Designing for Quality: the Client

Think Steel Brick: Self Contained, Compact, & Unbreakable

- **Small**: downloads fast, takes up little space
- **Fault Tolerant**: can connect to the server no matter what
- **Reliable**: does not crash or do anything else to appear unstable to the user
- **Plays Well With Others**: does not interfere with any other programs on the system including Internet download managers, etc.

Testing the Results: the Server

- Test your software in the same configuration it will be deployed
- Work with your internal customers to define use cases for testing
- Increase your focus on performance, load, and stress tests.

Testing the Results: the Client

- Test the client software as though it is mission critical—because it is.
- Test with a variety of hardware and software configurations.
- Become expert at compatibility testing.
- Emphasize destructive testing techniques.

Develop Expertise in Other Areas

- Identifying environmental problems that could result in perceived lack of quality in the service. Example: troubleshooting network problems
- Administrating the infrastructure for your service. Example: developers and testers should understand the basics of setting up and administrating a server farm.

Tools

- PQDI or Ghost for imaging the machines
- WCAT for load testing the server
- Custom stress tools to emulate real connections to your server or lots of activity on your client

Lessons Learned

- If you change your business model, your requirements have probably changed as well.
- Perception is reality— and the Internet can warp perceptions.
- Internal chaos is OK— as long as the customer and user see a polished image.
- Inefficiencies are OK— for a while.
- Your service will be judged harshly by customers and users alike— and you won't know in advance on what criteria, so be prepared to react quickly.

Quality in an Application Service Provider Environment

by Elisabeth Hendrickson

March 31, 2000

In 1998 I was consulting for a traditional enterprise software development company when the CEO called an all hands meeting. Although such meetings normally would have been only for employees, big changes were on the horizon, and I was asked to attend.

"We're no longer a software company. Now we're an Application Service Provider," the CEO proclaimed. We all looked at each other wondering what that meant.

The CEO continued, "The executive staff struggled with this decision. After much soul searching, we've concluded that this is the direction we must take. Sadly, not all of us could sign up for the new plan. I am sorry to announce the resignation of our VP of Engineering."

Most of us were shocked. A few people who had been warned about the meeting exchanged knowing looks. The CEO continued, "As sad as it is to see our VP of Engineering depart, I think that it's better that he leave than that he stay without being fully committed to our vision. And so, as tough as this is to say, I encourage all of you to consider this carefully. Anyone who doesn't feel they can sign up for the new deal should consider whether or not they should be here."

These were brash words in one of the worst hiring markets ever. To invite your staff to leave bordered on insanity. Yet the CEO didn't want anyone to think that the next day would be business as usual. As far as he was concerned, everything had changed. Sure, the engineers would keep coding. But now they were not building enterprise software. They were building a service.

What's Different at an Application Service Provider (ASP)?

For an independent software vendor, the software is the product. The company ships the software— usually in a fancy box with documentation, some flyers, and a registration card— to the customer, who opens the box, takes out the distribution media, and swears a few times as he or she installs the software on his or her system. In contrast, ASPs sell access to the software, and not the software itself.

The ASP business model is gaining popularity because it simplifies adoption of new software:

- ASPs typically charge their customers for the service on a subscription basis. For the customer, that means that they don't have to pay a large sum up front to acquire software licenses. It also means that if the new software does not work out, they can cancel their subscription— they are not stuck with software they will not use.
- The ASP hosts the server machines and handles the installation, administration, and maintenance of the software. For the customer it means that they don't have to assign a system administrator to maintaining the new software.

The term "ASP" covers a wide range of businesses:

- Companies like FaceTime and eGain provide access to software to facilitate real time and asynchronous communication (instant messaging and email management).
- Companies like Aveo provide a mechanism for distributing targeted communications to your customers.
- Companies like Exodus provide co-location hosting services, Internet access, and server monitoring.

A typical ASP has server-side software that directly supports the service and client-side software that allows users to access the service. The client-side software may be a web-based application or a standalone application that resides on the client's machine.

ASPs still need software. So why does anything need to change for the software development and testing groups?

The company where the CEO announced the change from enterprise software development to service provider discovered that the change in the business model meant a change in priorities. Suddenly having a nice server-side installer was not as important as being able to support multiple clients on a single server.

The change in priorities reflects a change in the requirements for the system being built. The customers' needs have changed, and the system must change accordingly. So it's worth spending a little time talking about the customer.

Who is the ASP's Customer?

For the purposes of this paper it is worth making a distinction between the ASP's customers and the end users. These two stakeholders may have slightly different criteria on which they judge the quality or value of the service.

The customer is the company paying for the service from the ASP. The customer needs to be assured that the service will stay up. The customer also may need access to data on the system— but not directly. They will need to run reports remotely, perhaps through web pages. The customer usually also uses client software to access the service.

The end user is the person using the service, usually at the behest of the customers. They use the client software provided by the ASP.

Defining Quality in an ASP

When buying software, the customer's first impression of the quality of the software rests on the quality of the packaging, the readability of the documentation, and the ease of installation. The customer's continued perception of the quality of the software rests on the software's reliability and capability: does the software do what the user bought it to do, does it do it reliably, and does it do it without interfering with anything else the customer is trying to do on the computer.

In an application service provider environment, everything changes.

If the customer receives software to install, it is not server-side software; it's a client that enables the customer to get to the server. This client software is judged on many of the same quality attributes as "traditional software." However, it is only a small piece of the puzzle.

The customer's continued perception of the quality of the service relies on the availability and reliability of the service, not the server software. In an application service provider environment, the software is not the product— the service is. The software enables the service. But the quality of the software is not as important as the quality of the service.

"Quality" in an ASP is therefore:

- Reliability and speed of the service to the end user
- Scalability of the service
- Availability of data to the customer
- Quality of any client software that a user or customer must use to get to the service

Ease of installation and maintenance of the server software is no longer a key factor in the perception of the quality of the service. If you have to use duct tape and bailing wire to get the software onto the server, that's OK— as long as the service stays up. Ease of installation becomes much more of a cost factor than a quality factor: the harder it is to install the server software, the longer it takes, the more it costs to do the installation, the more the service costs to provide. But the customer and the end user are not directly affected by the system administrator's installation woes.

That doesn't mean that ASPs can completely ignore installation issues. Particularly in Silicon Valley, hiring good system administrators can be very expensive. It is not cost effective to make these people spend hours trying to get the server software up and running on each new installation. However, particularly when an ASP is dealing with relatively few servers, inefficiencies in the process are acceptable.

The ASP also has another advantage: since the ASP controls the servers, the ASP also controls the server configurations. Companies that sell server software must support a variety of server configurations—different OSes and patch levels, different back-end databases, different HTTP servers. The ASP can choose just one server configuration to support.

Designing the Server Software for Quality in an ASP

At one enterprise-software-company-turned-ASP, we thought, "We're still building the same software. But now we're just selling access to it rather than selling the software package itself." We missed some key requirements changes:

- The original design allowed for only one customer's software to run on each machine. The new ASP business model called for a large number of customers, each bringing in a relatively small number of end users. Our server deployment scheme was now very inefficient.
- The tools that enabled administrators to add data to the servers worked well for small amounts of data but were hopelessly poor at managing large amounts of data. The user interface should have been entirely re-vamped given the new business model.
- Because we originally envisioned the server software being deployed behind a firewall, we did not design the server software to work well through firewalls. We initially had some trouble with users who could not get to the server because their firewall blocked our traffic.
- We did not take into account possible different ways in which customers would want to do business. When we were an enterprise software company, the problem would not have existed: the customer would have set up the server their way. In the new ASP business model, users connect to a shared server. We had to devise a method of providing the appearance of a customized service on a shared server very quickly.

These experiences have taught me that there are four key areas where we discovered that we had to change our original designs:

- How can multiple customers co-exist on a single server? If your software requires a single server for each customer, the cost of serving each customer goes up.
- How will the software deployment scale as demand for the service for a single customer grows? Design your server software for scalability: make sure it can be deployed in single server or multiple server configurations.
- How will your administrators be able to tell when the server software is not responding? Design your server software for monitoring so you know immediately when the service is out. On NT, you can do this by writing errors to the NT Event Log— there are administration packages such as SiteScope that will look for critical errors in the log and page the administrators.
- How will the administrators deploy and administer the software? Take advantage of the fact that the person who will be installing and supporting the software sits just a few cubicles away. Involve the internal customers early in the design process.

Perhaps the most important lesson I learned was that when you make a fundamental change to your business model, you must revisit your requirements.

Server Software Testing in an ASP

Originally, when we were an enterprise software company, we'd projected that the amount of data on the server at any given time would be relatively small. Part of the new business model included adding a great deal more data to the server. We failed to test the server software under these new conditions.

Unfortunately, when we put the server software into production we discovered severe performance degradation when large numbers of users tried to get large amounts of data.

To work around the performance problems, the servers had to be rebooted on a regular basis— about once an hour. Fortunately, because we were not running a real time service, the end users never noticed the difference. But our internal IT department was in an uproar.

The problems we observed might have become problems for our customers if we were selling the server software— or they might not have. It's hard to say if the conditions under which the server software would have been used in an enterprise environment would have triggered the underlying faults that plagued us those first few weeks after release.

We had one advantage: because we controlled the servers, we could roll out new software when we needed to without waiting for a customer's administrator to be available. We could also do a staged rollout— one server at a time— so we could see how the new software (much more thoroughly tested this time since we'd learned our lessons) behaved in production.

From this experience, I learned not to underestimate the importance or effort involved in performance, load, and stress tests. I learned that for our system there are two important variables: number of end users connecting and amount of data in the server. Both factors play an important role in the overall performance of our system.

In retrospect, this seems obvious. It's easy to say we did inadequate test analysis, and that we should have known we needed to do those kinds of performance, load, and stress tests. But what seems obvious in hindsight was anything but obvious to us when we were initially designing the load tests.

Because an ASP serves many customers, each with their own capacity requirements, server capacity tests are critical. As you are revisiting the requirements for the system, revisit the requirements for the tests. Perhaps you'll need to increase the amount of data on the server during the load tests. Perhaps you'll need to test the server by emulating several customers'end users so the emulated clients all get different data.

I've learned that as a general rule, it is a good idea to:

- Test your software in the same configuration it will be deployed— unlike testing software for sale, you can actually know precisely what that configuration will be.
- Work with your internal customers to define use cases for testing. This will ensure that the software is tested in a way that is representative of how it will actually be used.
- Spend a large percentage of your time designing and executing performance, load, and stress tests. Ensuring that the server software can handle the load— and knowing how to scale the server implementation for a heavy load— is as important as ensuring that it works as designed.

Ensuring Quality of the Client Software

Most of this paper has focused on ensuring the quality of the back end software. However, the client is the most obvious representation of the service to the end user. We learned that if you are an ASP distributing client software, you also have slightly different requirements than if you are an enterprise software company:

- The client must be lightweight: users downloading it from the Internet don't want to download 40Mb files and customers distributing it on their CDs don't have that kind of space available.
- The client must be absolutely rock solid: if users have problems with their machine, they will attribute those problems to your software whether or not your software was at fault. If your software really does cause problems, the end users will abandon the service rapidly.
- The client must be robust: no matter what, it must be able to reach back to the server— or you've lost the end user as surely as if they uninstalled your software.

The client may be a web application so no installation is required. Even if this is the case, the three principles above are still in effect. But now your test matrix is even larger: test with all the flavors of browsers on different kinds of machines including Macs as well as the different PC operating systems.

If you are an enterprise software company, the client software probably isn't considered mass-market software. Once you become an ASP, however, you're in the mass-market software business. You'll want to treat your client development effort like a mass-market software development effort: test across a very wide range of possible client machines, test for compatibility with other software on the system, and put a lot of effort into negative and destructive tests to ensure that the client software can withstand a lot of abuse.

One example of a compatibility problem we found: another software package inappropriately removed one of the system components our software relies on. We had to find ways to work around that problem, even though it was not our bug.

Influencing Customer Perception of Quality

As stated earlier, the quality of the software supports— but is not synonymous with— the quality of the service. There are other things that the ASP can do to improve the customers' perception of the quality.

One ASP discovered that the customer perceived the performance of the service to be unacceptably slow, not because the service itself was slow, but because the customers' connection to the Internet was low-bandwidth. The ASP dispatched a network administrator to the customer site to troubleshoot the problem. The low-bandwidth connection was identified as the problem, the customer got a higher bandwidth connection to the Internet, and their perception of the quality of the service went up tremendously. The ASP developed expertise in monitoring the entire network and not just their own software.

In another case, the service administrators knew that the server installer they were getting from the software development group was clunky and error-prone. So the administrators practiced installing the new server software for the last two months of software development. By the time the server software was ready to deploy, the administrators could have installed it in their sleep. The customers, braced for a little downtime as the servers were upgraded, were pleasantly surprised to discover that the transition went smoothly from start to finish.

Lessons Learned

If you change your business model, your requirements have probably changed as well. Whether you realize it or not, the customers who would have bought your software probably had different expectations from customers who are paying for access to the software. Your company almost certainly has different needs since it will be supporting multiple customers on the same set of machines.

Perception is reality— and the Internet can warp perceptions. In the case of the company who thought that the service was unacceptably slow, the ASP learned that the customers' perception of speed is only partly determined by the actual speed of the service. Fair or not, the customer gauges their assessment of the quality of the service on their overall experience. They don't attribute the cause of poor performance to limitations in their environment.

Internal chaos is OK— as long as the customer and user see a polished image. In the case of the first server rollout where the administrators had to reboot the servers every hour (they finally wrote a script to do that), the users perception of the quality of the service did not change.

Inefficiencies are OK— for a while If you are in start up mode, you can usually afford some inefficiency in your business practices. The number of customers is relatively small, so the inefficiencies don't become huge because of multiplicative effects. However, if the ASP accepts inefficiencies in the long term, costs will skyrocket.

Your service will be judged harshly by customers and users alike— and you won't know in advance on what criteria, so be prepared to react quickly. One ASP discovered that users were balking at downloading their rather large client software. So they devised a new distribution mechanism to enable the

client to slowly download in the background over time. Another ASP solved the problem by turning their standalone windows client into a Java applet on a web page. Both changes had to be made very quickly. Working on Internet time takes on a whole new meaning when you are an ASP.

Tools & Bibliography

SiteScope is a tool for monitoring the health of servers: http://www.freshwater.com

PQDI is drive imaging software from PowerQuest: http://www.powerquest.com.

Ghost is drive imaging software from Symantec: http://www.ghostsoft.com

WCAT is a tool from Microsoft for load testing web applications:
http://msdn.microsoft.com/workshop/server/toolbox/wcat.asp

Anderson, Mark D. "The Top 13 Mistakes in Load Testing Applications," *STQE Magazine*, October/November 1999.

Hendrickson, Kirk and Hendrickson, Elisabeth, "Quality and the Technology Lifecycle," *SM/ASM 99 Proceedings*

QW2000 Paper 6A2

Dr. Jerry Gao, Mr. Kamal Gupta & Ms. Shalini Gupta (San Jose State University)

Design for Testability of Software Components

BACK TO QW2000 PROGRAM

Presentation Abstract

We focus on design for testability of software components. We first discuss our views of component testability, including its factors and aspects.

Then, we introduce a new concept, known as testable beans, to help engineers to understand testable components in terms of supporting features, properties and capabilities for testing.

We believe that good reusable commercial components must be deployable, testable, and manageable.

About the Authors

Jerry Z. Gao, Ph.D., Assistant Professor, Department of Computer Information and Systems Engineering (CISE), San Jose State University, San Jose, California. Dr. Gao has written many technical papers on software engineering, object-oriented technology, and object-oriented testing. His current research interests include component engineering, software testing methodology and supporting environments, virtual software engineering environment over the Internet, and Internet computing. Before he joined San Jose State University, he worked for Fujitsu Network Communications System, Inc. as a manager of an R&D group in the software-engineering department. At Fujitsu, he was instrumental in developing an enterprise software production environment over the Internet. His team is the driving force to establish an enterprise-oriented software-engineering environment over the Internet in Fujitsu. He is a co-editor of the book, "Object-Oriented Software Testing", IEEE Computer Society Press, 1998.

Dr. Gao and his colleagues have established the first Internet technology Lab in SJSU. He has

proposed and taught several graduate courses on Internet computing, including Java Programming, Introduction to WWW, Design and Implementation of Web-based Application Systems and Tools, and Design and Implementation of E-Commerce Systems. Dr. Gao has been invited to offer a short course on the topic of Engineering Global Software Production on the Internet.

Kamal Gupta is a Project Leader working in the area of Software Engineering at one of the leading EDA companies, Synopsys Inc.. He carries M.S. in Computer Engineering with specialization in Software Engineering from San Jose State University, San Jose, California. Before doing M.S. he did his B.S. in computer Engineering from India. His current research interests include software component engineering, software testing methodology, distributed object programming and database management systems. He has more than ten years of industry experience in the area of software engineering and is currently working on the development of software tools.

Shalini Gupta is working for Fujitsu Network Communications System. Inc. as a Software Engineer. She carries M.S. in Computer Information and Systems Engineering (CISE) with specialization in Client/Server computing from San Jose State University, San Jose, California. Her current research interests include component engineering, software testing methodology and Client/Server computing. She is currently working in the area of network management tools.

BACK TO QW2000 PROGRAM

Design for Testability of Software Components

Jerry Gao, Ph.D., Kamal Gupta, Shalini Gupta

San Jose State University
One Washington Square
San Jose, CA 95192-0180
Email:gaojerry@email.sjsu.edu

Presentation Outline

- *Introduction*
- *Understanding Component Testability*
- *Evaluation of Component Testing Capability*
- *Mechanism to increase Component Testability*
- *What is a Testable Bean ?*
- *Why do we need Testable Beans?*
- *Design For Testable Beans*
- *Advantages & Issues*
- *Technology used and Status*
- *Conclusion*

Introduction

- *Challenges in Software Component Engineering :*
  - *What is Software Component Testability?*
  - *How to increase Testability during design phase?*
  - *How to construct Testable Software Components systematically?*
  - *How to facilitate Component testing in a systematic way?*
- *Our Focus:*
  - *Design for testability of software components*
- *A new concept: Testable Bean*
  - *Propose a new component architecture*
  - *Propose well-defined built-in test interface.*

Understanding Component Testability

- *Important indicator for component quality and reliability*
- *Two aspects:*
  - *a) Observability and Traceability*
  - *b) Controllability*
- *Our perspective of Testability:*
  - *indicates how well a component is structured to facilitate software testing*
- *Current issues on component testability --> poor testability due to:*
  - *No external tracking mechanisms and tracking interfaces*
  - *No built-in tests*
  - *No built-in test interface to support software testing*

Evaluation of Component Testing Capability

Level 1: Initial

- *Constructed with ad-hoc testing mechanism, test formats & functions*
- *More time in understanding behaviors, debugging & testing*

Level 2: Standardized

- *Built to support pre-defined testing mechanism, test format*
- *Reduces cost on debugging & testing*
- *Extra programming overhead*

Level 3: Systematic

- *Designed with a set of systematic testing mechanism*
- *Easy to monitor and test the components*
- *Reduced programming overhead*

Level 4: Customizable

- *Designed to facilitate the support of testing functions & customization*
- *Helps to set up testing for component based software*

Mechanism to increase component testability

Three Mechanisms to construct testable components:

- *Framework-based testing facility*
  - *well-defined framework (such as class library) to add test code*
  - *simple and flexible to use*
  - *needs component source code*
- *Built-in tests*
  - *needs well-defined built-in mechanism to add test code*
  - *high programming overhead during component development*
  - *no external support needed*
- *Automatic component wrapping for testing*
  - *component wrapped inside program for testing*
  - *low programming overhead*
  - *well-defined test framework to interact with test tools*

Testability enhancement mechanisms

Testing Perspectives	Framework-Based Testing Facility	Built-in Tests	Automatic Component Wrapping for Testing
Source Code	Needed	Needed	Not needed
Testing Code Separation	No	No	Yes
Development Overhead for Testing	High	High	Low
Complexity	Low	Very High	High
Usage Flexibility	High	Low	Low
Applicable Components	In-house components	In-house components	In-house components and COTS

What is a Testable Bean?

- *What is a testable bean?*
  - *should be deployable and executable*
  - *must be traceable by supporting basic component tracking capability*
  - *must provide a consistent, well-defined and built-in interfaces*
  - *standardized mechanism to include built-in test code*

- *Design criteria while constructing testable beans:*
  - *minimize programming overhead for developers*
  - *standardize the test interface for testable beans*
  - *prevent adding detailed testing functions inside beans*
  - *separate the functional code from built-in test code*
  - *should not include detailed tests inside a test bean*

Test Interface

- *Test interface supports three basic functions:*
  - *set up a test (test case and/or data)*
  - *exercise a test for a specified function*
  - *validate and report the test results*

- *Features of a component testing support environment:*
  - *a test management tool for updating/maintaining/recording tests*
  - *a component test bed to set-up and exercise test*
  - *a component test generation tool to generate test cases*

Why Testable Beans?

- *Why do we need testable beans?*
  - *reduce test costs and enhance software quality*
  - *minimize testing efforts.*

- *Objectives to design and construct testable beans*
  - *provide traceable, reusable, executable, deployable, testable beans*
  - *provide plug-in-and-test environment*
  - *standardized the test interface for testable beans*
  - *achieve the component test automation*
  - *construct software testable components with good architecture*

Challenges while Constructing Testable Beans

–Design and implementation in a systematic way?

–Define component architecture and test interface?

–Create Testing framework to support test automation?

–Minimize programming efforts and system overhead?

–Provide standardized interactions between Bean and its supporting environment?

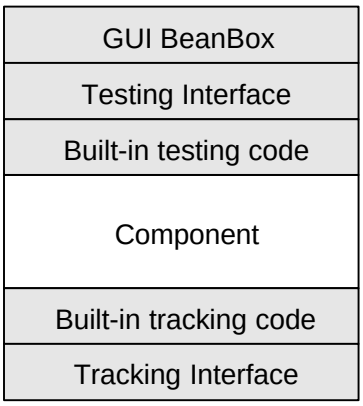
Design for Testable Beans

- *Our solution includes:*
  - *–a new architecture model*
  - *–a well-structured test interface*
  - *–a supporting test environment*
- *The Architecture model:*
  - *–GUI interface supporting user interaction*
  - *–Built-in test interface*
  - *–Built-in program layer to facilitate testing*
  - *–Built-in tracking interface for tracking capabilities*
  - *–Built-in tracking program layer to facilitate tracking*

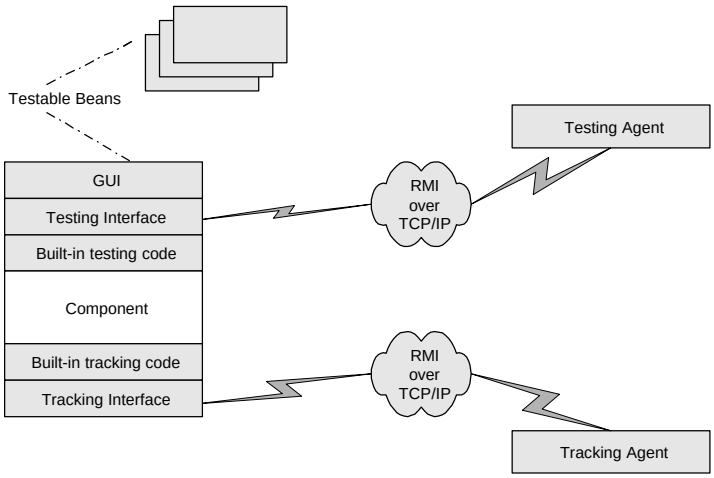
Design for Testable Beans (cont.)

- ***A well-structured test interface:***
  - ***includes three functions:***
    - ***setParameters, to set up the parameters for the functions***
    - ***runMethod, to execute a given method***
    - ***validateTestResult, to validate the test results***
- ***A supporting test environment:***
  - ***includes three parts:***
    - ***Testable Bean in a computer node***
    - ***A Test Agent (along with a testing package)***
    - ***A Tracking agent (along with a tracking package)***

Testable Bean Architecture



Testing and Support Environment



Transaction sequences

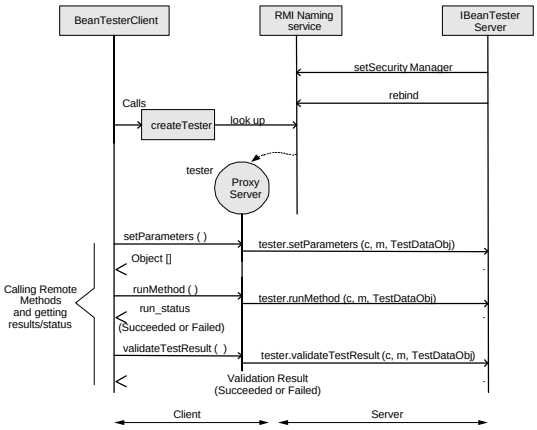


Figure: Transaction Sequence (RMI Client/Server Communication)

Interface between Component and Testing Agent

```
public interface IBeanTester
{ public Object[] setParameters(String className_val, String
                                methodName_val, TestData testData_val);
  public int runMethod(String className_val, String methodName_val,
                       TestData testData_val);
  public int validateTestResult(String className_val, String methodName_val,
                       TestData testData_val);
}

public class TestData
{ public TestData(Object[] argValueList_val,
                  Object[]expectedTestDataList_val)
  {Object[]   argValueList = argValueList_val;
     Object[]   expectedTestDataList = expectedTestDataList_val; }
}
```


Connection between the Client and Server

```
public class CreateBeanTester
{ ....
  public IBeanTester createTester(String serverName){
    try {
      IbeanTester testerObj = (IBeanTester)Naming.lookup(serverName); }
    catch (Exception e)
    {
      System.out.println("BeanTesterClient.createTester: an exception
                          occurred:");
      e.printStackTrace(); }
    return testerObj; }
}
```

Implementation of Testing Agent

```
public class BeanTesterServer
{
   public static void main(String args[]) {
      // Create and install the security manager
      System.setSecurityManager(new RMISecurityManager());
      try {
         // Create BeanTesterImpl
       BeanTesterImpl beanTestObj = new BeanTesterImpl("beanTestServer");}
      catch (Exception e) {
         System.out.println("BeanTesterServer.main: an exception occurred:"); } }
}
```


Testing code example

```
import beantester.tester;
… .              // other import package
public class BeanTesterClient
{    public static void main(String[] args)

// className, methodName, argValue_List, expectedTestData_List is supplied   by the user either
      through the GUI interface or at the command line

      boolean testable_flag = true; // Set up the testable_flag to true to enable the testing
      while(flag) {
         CreateBeanTester beanTesterObj = new CreateBeanTester();
         IBeanTester  tester = beanTesterObj.createTester("beanTestServer");
         TestData testDataObj = new TestData(className,methodName, argValue_List,
                                                                    expectedTestData _List);
         parameter_List = tester.setParameters(className,methodName, testDataObj);
         int run_status = tester.runMethod(className,methodName, testDataObj);
         int  validateResult = tester.validateTestResult(className,methodName, testDataObj);
         testable_flag = false;  }
 }
```

Advantages and Issues

•*Advantages:*

- *Reduces software test cost and enhances software quality*
- *Minimizes testing cost and efforts*
- *Provides a scaleable model*
- *Flexible to control testing capabilities*
- *Provides a well structured testing package*
- *Monitors external behavior of third-party components*

•*Issues:*

- *Software maintenance and support for component-based software*
- *Component behavior analysis and performance*

Technology used and Status

•*Technology Used:*

- *JDK 1.2 for Testing Client and Testing Agent*
- *RMI for the communication*

•*Implementation Status:*

- *Tool partially built but has successfully verified the suggested model.*

•*Future Work:*

– *Come up with a new component test-bed technology, applicable to diverse components developed in different languages and technologies and on different platforms.*

Conclusion

- ***Testability offers***
  - ***–significant insights that are useful during design, testing, and reliability assessment.***
  - ***–promise for quantitative improvement in software quality.***
  - ***–reduction in testing cost and efforts.***
- ***Introduced the new concept: Testable Beans***
- ***Defined the new architecture and test interface for Testable Beans.***

References

[1] Elaine J Weyuker, "Testing Component-Based Software: A Cautionary Tale", *IEEE Software*, September/October 1998.

[2] Jerry Z. Gao, "Testing Component-Based Software", STAR'99.

[3] Jerry Z. Gao, "Perspectives of Software Component Testability", submitted for publication in 1999.

[4] Voas, J. M. and Miller, Keith W.," Software Testability: The New Verification", *IEEE Software*, Vol. 12, No. 3: May 1995, pp. 17-28.

[5] Roy S. Freedman, "Testability of Software Components", *IEEE Transactions on Software Engineering*, Vol. 17, No. 6, June 1991.

[6] David S. Rosenblum, "Adequate Testing of Component-Based Software", Department of Information and Computer Science, University of California, Irvine, Technical Report UCI-ICS-97-34, Aug. 1997.

[7] William T. Councill, "Third-Party Testing and the Quality of Software Components", IEEE Software, Vol. 16, No. 4, pp. 55-57, July/August 1999.

[8] Yingxu Wang, Graham King, and Hakan Wickburg, "A Method for Built-in Tests in Component-based Software Maintenance", Proceedings of the Third European Conference on Software Maintenance and Reengineering, 1998.

Design for Testability of Software Components

Jerry Gao, Ph.D., Kamal Gupta, Shalini Gupta
San Jose State University
One Washington Square
San Jose, CA 95192-0180
Email: jerrygao@email.sjsu.edu

Abstract

Component engineering is gaining substantial interest in the software engineering community. A lot of research efforts have been devoted to the analysis and design methods for component-based software. However, only few papers address the testing of software components and component-based software. This paper focuses on *design for testability of software components.* It discusses component testability, its factors and aspects and solutions in supporting software components and component-based software. Moreover, the paper introduces the concept of *testable bean*, and proposes a new way to construct a testable bean based on a testable architecture and well-defined built-in interfaces.

Keywords: Component engineering, software component testing, software testing, software testability.

1. Introduction

As the software programs increases in size, it is very important to reduce high software cost and complexity while increasing reliability and controllability. With the advances in Internet technology, more distributed systems are being built to meet diverse application needs. Currently component engineering is gaining substantial interest in the software-engineering community. As more third-party software components are available in the commercial market, more software workshops have begun to use the component engineering approach to develop component-based programs for the distributed applications.

Although a lot of research efforts have been devoted to analysis methods and design strategies of component-based software, only a few papers address the testing of software components and component-based software [1][2][3][4][5][6][7][8][9]. To build high quality software, we need good quality components and cost-effective component testing methods [1]. In the practice of component engineering, we have encountered some new problems and challenges in testing of software components and component-based software [2]. One of them is how to build software components with good testability.*As pointed out by Jeffrey Voas [5], testability is an important indicator for component quality and reliability*. To solve this problem, we must have a clear understanding about component testability, and provide new architecture models and methods to help engineers to construct testable components. In addition, we need new component test tools and technologies to support the automation of a test process for software components.

In component engineering practice, we encountered several challenges concerning testing of software components. They are:

- What is software component testability?
- How to increase the testability of software components during component design?
- How to construct testable software components with a systematic approach?

- How to facilitate component testing in a systematic way?

In this paper, we focus on design for testability of software components. We believe component testability is a very important factor to increase component quality and reduce testing cost. Therefore, it is essential for component developers to construct deployable, executable, testable and manageable software components.

To help people understand how to construct testable software components, we introduce a new concept, called *testable beans.* To explain this concept, we propose a new component architecture and well-defined built-in test interfaces to enhance the component testability and facilitate component testing. In this paper, we focus on our discussion on testable beans by answering the following questions:

- What is a testable bean?
- Why do we need testable beans?
- What are the basic features, capability and properties of a testable bean?
- What are the architecture and interfaces of a testable bean?

In addition, our application and implementation experience indicates that the research result here is very useful and practical to develop a software test tool to automate a component testing process. Our application experience suggests that this approach is one of the cost-effective ways to build highly testable components. Moreover, standardized testable beans can help us to automate a component test process for all software components. Our result and case study suggests us that if we can standardize the architecture model, supporting test interfaces, and built-in testing mechanisms for software components, then we achieve component test automation from the following aspects:

- Generate testable components (or beans) in a systematic manner.
- Create a generic component test-bed to provide a plug-in-and-test environment for various software components.
- Create an independent component testing box with well-defined interfaces that interact with testing tools and test database to facilitate component testing in a more flexible way.
- Add built-in test codes inside a software component in a systematic way.

The paper is structured as follows. In Section 2, we first discuss our views of component testability, including its factors and related aspects. In Section 3, we introduce a new concept, known as *testable beans,* to help engineers understand testable components in terms of supporting features, properties and capabilities for software testing. Section 4 presents a systematic approach to construct testable beans and facilitate component testing. Finally, our conclusion remarks and future work are given in Section 5.

2. Understanding Component Testability

In the component engineering paradigm, one of the primary goals is to generate reusable software components as software products. The third-party engineers use the components as parts to build specific software systems according to the requirements given by customers. Therefore, the testability of a program highly depends on the testability of involved components and their integration.

Testability is an important indicator for component quality and reliability and has two aspects: a) *observability*, and b) *controllability*. *Controllability* of a program (or component) is an important property that indicates how easy it is to control a program (or component) on its inputs, operations, behaviors and outputs. *Observability* of a program (or component) is another critical property that indicates how easy it is to observe a program in terms of operational behaviors and outputs related to its inputs. From a customer's point of view, the testability of current software components is poor due to the following reasons.

- No external tracking mechanisms and tracking interfaces in software components for a client to monitor or observe external behaviors.
- No built-in controllable interfaces in software components to support the execution of unit tests, checking the test results, and reporting errors.
- No built-in tests or controllable test suites for software components to support black-box tests at the unit level.

Component Observability and Traceability

According to Roy S. Freedman [6], *software observability* indicates how easy it is to observe a program in terms of its operational behaviors, input parameters, and outputs. This implies that the design and definition of a component's interfaces (such as incoming and outgoing interfaces) affects its observability. In the practice of component engineering, we found that the component traceability is another very important factor that affects the component observability.

"Traceability" of a software component refers to the extent of its built-in capability of tracking the status of component attributes and component behavior. It has two folds: a) *behavior traceability*, and b) *trace controllability*. *Behavior traceability* refers to the degree to which a component facilitates the tracking of its internal and external behaviors. In the real world, component engineers have learned to check and monitor the internal and external behaviors of software components by adding a program tracking mechanism in software. In the real practice, engineers are not successful in delivering software components with good behavior traceability due to the following two reasons. First, in the development of software components, engineers used to pay much attention to track component internal behaviors than external behaviors. Therefore, component testers, integration engineers and customers have the difficulty in monitoring and checking the external behaviors of software components. Next, component developers are learning how to design and develop traceable components due to the lack of standardized component trace formats and tracking mechanism [3].

Trace controllability refers to the extent of the control capability in a component to facilitate the customization of its tracking functions. With trace controllability, engineers can control and set up various tracking functions such as turn-on and turn-off of any tracking functions and selections of trace formats and trace repositories. Although the current commercial components and most in-house components do not provide this capability, it is an ideal and cost-effective feature for testers and maintainers to support debugging, component integration, and system. Trace controllability is really useful and necessary for complicated components with customization functional features in a distributed environment.

Component Controllability

Controllability of a program (or component) is an important property that indicates how easy it is to control a program (or component) on its inputs/outputs, operations, and behaviors [6]. Component developers look at the *"controllability"* of a software component from three aspects: a) behavior control, b) feature customization, and c) installation and deployment. The first has something to do with the controllability of its behaviors and output data responding to its operations and input data. The next refers to the built-in capability of supporting customization and configuration of its internal functional features. The last refers to the control capability on component installation and deployment.

Test engineers and component customers are concerned about other factors of component controllability, which may directly affect the component testability. The trace controllability in software components is a typical example. Testers and customers expect components to provide a set of control functions in software components so that they can use them to monitor and check diverse component behaviors according to their needs. We believe component trace controllability is very useful in component debugging, component integration, and system testing. In addition, component testers and customers expect software component vendors to generate components with test controllability to support acceptance testing and unit testing in a standalone mode. Component test controllability refers to a component's capability of retrieving and exercising component tests. Unfortunately, most current component vendors do not provide components with this capability due to the lack of research results on how to design and develop testable components.

Evaluation of Component Testing Capability

It is important to evaluate the testing capabilities of a software component to be able to construct a good testable component. We have defined the following four levels for measuring and evaluating the testing capability of a software component.

Level 1: Initial – Software components, at this level, are constructed in a way to facilitate component and program testing by considering program tracking, built-in testing, interaction interfaces with test tools and test databases. However, component developers use an ad hoc approach to define program tracking format and mechanism, built-in testing methods, and built-in interfaces and interactions with testing tools. For software components at this level,

engineers usually encounter a lot of problems in understanding component behaviors, executing component tests, validating and debugging test results, and interacting software testing tools.

Level 2: Standardized – Software components, at this level, are built according to a pre-defined set of standards on test case formats, tracking message formats and mechanisms, built-in test techniques, and interaction interfaces with software test tools. These standards will help component developers to construct software based on the pre-defined standards to facilitate component testing and program testing. When developers follow these standards, they can generate testable components to reduce the testing cost and increase component testability. However, to reduce the programming overhead and testing overhead they need more guidelines and systematic way help them in component construction.

Level 3: Systematic – At this level, software components are designed with a set of systematic built-in mechanisms and interfaces to facilitate component testing and interactions with software testing tools, so that they can be easily executed, tested, and monitored in a systematic manner. With components at this level, component test automation can be achieved in a systematic way. The major advantage here is to reduce a lot of programming overhead on building testable components.

Level 4: Customizable – Components at this level consist more flexible built-in test mechanisms and testing interfaces are embedded inside to support the customization of tests, test formats, test function implementation and configuration. Building customizable software components for testing enhance the component flexibility to define, select, and configure built-in tests and testing functions, trace and test formats, as well as test tool interactions.

Mechanisms to Increase Component Testability

Since a component-based program is an integration of software components, its program testability depends on the testability of its components. Component observability is the fundamental base of component testability. We have discussed the component traceability and three methods to increase software component traceability in [2][3][4]. Similar to component traceability, we need to add some well-defined mechanisms to software components to enhance their testability. As we pointed out in [1], component engineers need guidelines and systematic mechanisms to construct testable components. This is very important for supporting and testing both in-house components and commercial components (COTS). Here we discuss three different systematic mechanisms to construct testable components by adding program codes and interfaces to support and facilitate component testing and program testing. Table 1 lists three basic approaches to add a consistent testing capability into software components to increase component testability.

Method 1: Framework-based testing facility – In this approach, a well-defined framework (such as a class library) is provided to component engineers to facilitate them to add program testing code into software components according to the application interfaces of the given framework. It usually is a testing facility program or a class library. Component engineers can use the facility to add their built-in testing code into components. This approach is simple and flexible to use. However, there are several drawbacks. Firstly, it requires a high programming

overhead. Secondly, it relies on engineers' willingness to add the testing code due to its high overhead in programming. Moreover, this approach assumes that component source code is available. Therefore, it is difficult to deal with commercial components (COTS) because usually they do not provide any source code to clients.

Testing Perspectives	Framework-Based Testing Facility	Built-in Tests	Automatic Component Wrapping for Testing
Source Code	Needed	Needed	Not needed
Testing Code Separation	No	No	Yes
Development Overhead for Testing	High	High	Low
Complexity	Low	Very High	High
Usage Flexibility	High	Low	Low
Applicable Components	In-house components	In-house components	In-house components and COTS

Table 1. Comparisons of Three Different Mechanisms for Testing

Method 2: Built-in tests – This approach requires component developers to add test code and tests inside a software component as its parts. Usually, this approach needs a well-defined component built-in mechanism and coding format. Developers must follow them to add tests and built-in test functions inside components. This causes a high programming overhead during component development. The major advantage of this approach is that tests are built-in inside components. Therefore, engineers can perform component tests without any external support from testing environment and test database. It is clear that this is a good idea to include component acceptance tests inside components like COTS. The major drawback of this approach is that components consume large system resources and become more complex because they include a lot of unrelated functional features. In addition, only limited types of tests can be included inside components.

Method 3: Automatic component wrapping for testing - This approach uses a systematic way to convert a software component into a testable component by wrapping it with the program code which facilitates software testing. Like the first method, this approach usually uses a well-defined test framework to interact with test tools or detailed testing functions. Compared with the first two methods, this approach has several advantages. First, its programming overhead is low because of automatic code wrapping. Next, it separates the code (which facilitates testing) from the original source code of a component. In addition, this method can be used for in-house reusable components and commercial components (COTS).

As shown in Table 1, these three approaches have different pros and cons. In real practice, we need to use them together to support different types of testing for a program and its components. To design and construct testable components, engineers need more guidelines on component architecture, testing interface, and supporting facilities.

3. Design for Testable Beans and Supporting Environment

What is a testable bean?

A testable bean is a testable software component that is not only deployable and executable, but also testable with the support of standardized components test facilities. Unlike normal components, testable beans have the following unique requirements and features.

- ***REQ #1:*** A testable bean should be deployable and executable. A JavaBean is a typical example.

- ***REQ #2:*** A testable bean must be traceable by supporting basic component tracking capability that enables a user to monitor and track its behaviors. As defined in [3], traceable components are ones constructed with a built-in tracking mechanism for monitoring various component behaviors in a systematic manner.

- ***REQ #3:*** A testable bean must provide a consistent, well-defined and built-in interface, called test interface, to support external interactions for software testing. Although different components include diverse functional interfaces, they must include a consistent test interface to support software testing. This is very important to automate component testing, and reduce test costs on environment setting and test driver construction.

- ***REQ #4:*** A testable bean must include some program code that facilitates software testing by interacting with external testing facilities or tools to support test set-up, test execution and test validation.

To construct cost-effective testable beans, it is very important for developers to understand the following design criteria:

- It is essential to minimize the programming overhead for developers.

- It is important to standardize the test interface for testable beans.

- It is suggested to prevent adding detailed testing functions inside beans.

- It is always a good idea to separate the functional code of a bean from the built-in code that facilitates the interactions between the bean and external testing facilities and tools.

- It is not good to include detailed tests inside a bean, otherwise, it is not easy to manage, reuse, and maintain tests in a flexible manner.

The test interface for a testable bean must support the basic three functions: a) set up a test (test case and/or test data) for testing a bean, b) exercise a test for a specific function of a bean, and c) validate the test result of a test with the given expected test data and report it.

Like traditional software testing, we can include the following features in a component testing environment:

- A test management tool, which allows testers to create, update, and maintain component tests. In addition, this tool also records the testing results from component testing.

- A component test bed, which uses component test drivers to set-up and exercise various test cases for a component.

- A component test generation tool, which helps testers to automatically generate component test cases and test data.

There are three questions regarding the design of testable beans. The first question is how to design and define a common and consistent architecture and test interface for all testable beans. The next question is how to generate testable beans in a systematic way. The final question is how to control and minimize program overheads and resources for supporting tests of testable beans.

Why do we need testable beans?

Increasing software component testability is a key to reduce software test costs and enhance software quality. Jeffery Voas, M. Miller, W. Keith in [5] define an interesting approach to measure and evaluate software testability of a program from the software reliability perspective. Their approach can be useful to check the testability of software components to help testers to find out the quality of the components. In this paper, we use the term 'testability' to indicate how well a component is structured to facilitate software testing. The major purpose of constructing testable components is to minimize testing cost and efforts.

The objectives to design and construct testable beans are summarized below.

- To provide components that are traceable, reusable, executable, deployable, and testable.

- To minimize the testing efforts by providing a plug-in-and-test environment to support component testing and component integration.

- To standardize the test interface for components so that various test tools and facilities can be integrated and used together.

- To find a systematic approach to achieve the goal of the test automation for software components.

- To provide a new way to construct software testable components with well-defined design architecture and testing interface.

To construct testable beans, we encounter the following challenges.

- How to design and implement testable beans in a systematic way?

- What is the component architecture and test interface for testable beans?

- How to provide a testing support framework (or facility) and a component test environment to achieve component test automation?

- How to minimize the programming efforts and system overhead during the construction and execution of testable beans?

- How to provide standardized interactions between a test bean and its supporting environment?

Here we present a solution based on the concept of testable beans to solve the above problems. We focus on component architecture and the test interface for testable beans. The solution includes a) a new architecture model for construct testable beans, b) a well-structured test interface for testable beans, and c) a testing environment for supporting component tests.

The Architecture Model for Testable Beans

As mentioned before, intuitively, a testable bean is a software component, which is designed to facilitate component testing. Figure 1 shows the architecture model for a testable bean. A testable bean includes the following extra parts other than its original functional parts.

- A GUI interface which is an extension of BeanBox supporting component user interactions for testing operations.

- A built-in test interface that supports the external interactions for testing.

- A built-in program layer that facilitates the interactions between the testable bean and external testing facilities and supporting tools to complete various testing capabilities.

- A built-in tracking interface which supports user interactions to configure and select various tracking capability and features. The details are reported in [4].

- A built-in tracking program layer that facilitates the interactions between a testable bean and external tracking facilities and supporting tools to complete program tracking functions. The details are reported in [4].

This component architecture has several distinct features. First, it considers the needs of essential testing support for a software component, such as tracking and testing interfaces. Second, it separates the testing and tracking functional code from normal functional parts of a component. This makes it easy to add or remove testing and tracking code inside a component. In addition, it promotes independent interfaces to support testing and maintenance of software components.

Test Interface of A Testable Bean

As we pointed out in [1], one of the major challenges in component test automation is the fact that various components are constructed with diverse interfaces and technologies. To deal with this issue, we have come up with an idea to design a consistent interface for components to support test interactions between components and external tools and component users.

Figure 1 shows a generic test interface (IBeanTester) for a testable bean in Java. It includes three functions:

- Object[ ] setParameters(....), which sets up a test case and test data to exercise a given method of a Java class.
- int runMethod (....), which executes a given method of a Java class with the set-up test data.
- int validate testResult(....), which check the test results for an execution class method.

The major purpose of the interface is to define and regulate a test interface to component users and test tools for exercising various functions in a software testable component (testable bean). It plays a very important role to standardize the test interface between testable beans and component testers or external test tools. Following this test interface, component developers and test tool vendors can provide their detailed implementation to support component testing.

Testing Environment for Testable Beans

To support the component testing for various testable beans, we developed a simple client-server prototype. Figure 3 shows the structure of a client-server environment for testing testable beans. This environment includes three parts: a) testable beans, b) a test agent, and c) a tracking agent.

```
public interface IBeanTester
{
   public Object[] setParameters(String className, String methodName, TestData testData_val);

   public int runMethod(String className, String methodName, TestData testData_val);

   public int validateTestResult(String className, String methodName, TestData testData_val)

}

public class TestData
{
   public TestData(Object[] argValueList_val, Object[]expectedTestDataList_val)
   {
      Object[]   argValueList = argValueList_val;                  // Actual parameter values
      Object[]   expectedTestDataList = expectedTestDataList_val; // Expected data
   }
}
```

Figure 1. Interface between the Component and the Testing Agent

A *Test Agent* is a multithreaded process. Each thread controls, supports, and monitors component tests for a testable bean in the environment. A test agent includes the following functional parts: a) a component test facility, which provides the detailed implementations of test functions defined in the test interface, b), a synchronous communication interface between testable beans and a test agent, c) administration and configuration GUI, and d) a linkage interface to a component test tool that generates, manages, and maintain component tests for testable beans.

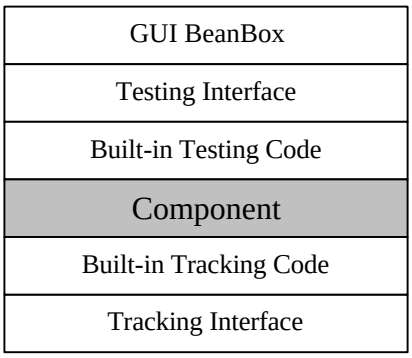


Figure 2. Testable Bean Architecture

In fact, the test agent can be located in the same machine as a component-based program. To deal with distributed component-based programs, we intentionally set up a client-server test environment to try our solution. We use Java RMI as a means to support the synchronous communications between a test agent and testable beans to support the test operations. Figure 4 shows the communication sequence between a client (a testable bean) and a server (test agent). First, the BeanTestAgent registers with the RMI Naming Service. After the registration, any testable bean constructed with the test interface can be supported on the network to complete the detailed test functions based on the component testing facility implemented in a test agent. The detailed test functions listed here are:

- Set up test parameters and test cases for a component function, for example *setParameter(…).*

- Invoke a component's function with a given test case, for example, *runMethod(….).*

- Validate test results from a test, for example, *validateTestResult(….).*

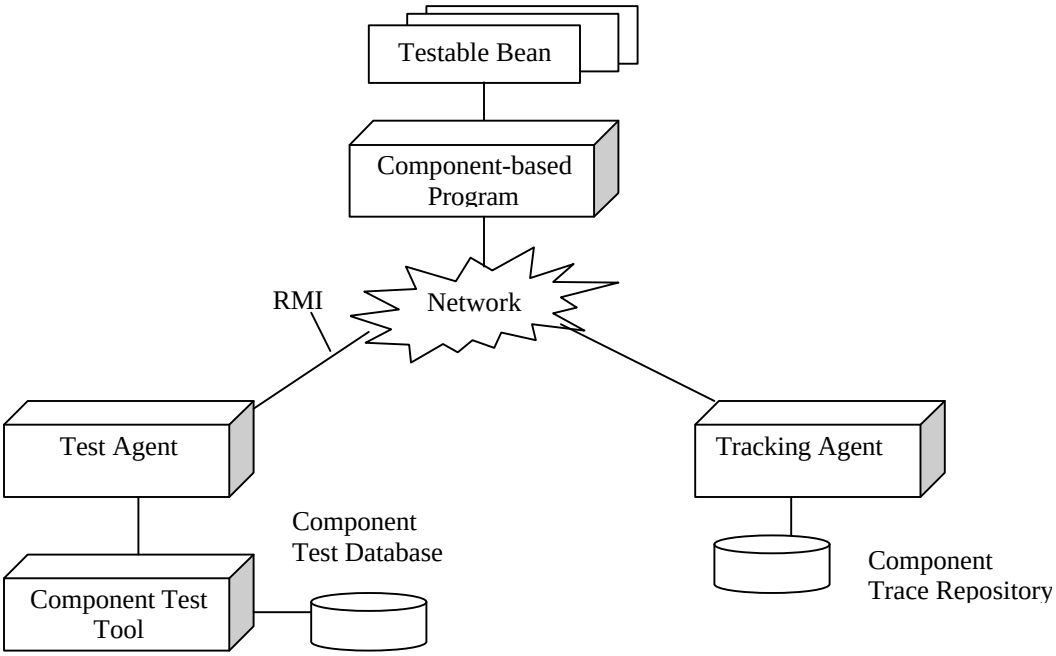


Figure 3. Component Testing Environment

The idea here is influenced by the concept of the remote procedure call. We implement the detailed testing support function as a component test facility of a test agent. They can be shared and used by all testable components (testable beans) on the network.

A *tracking agent* is a multithreading process that controls, records, and monitors diverse component behaviors in a component-based program. A *tracking agent* consists of the following three functional parts [4].

- *Tracking Listener* is a multi-thread program that listens and receives all types of tracking events through trace message queues, and dispatches them to *tracking processor.*

- *Tracking Processor* generates program traces according to a given trace event based on its trace type, trace message and data, and stores them in the proper trace repository.

- *Tracking Configuration* provides a graphic user interface to allow a user to discover and configure various tracking features for each traceable component.

Due to the limited scope of this paper, we only focus on test agent and testable beans. The more detailed information about Tracking Agent can be found in [4].

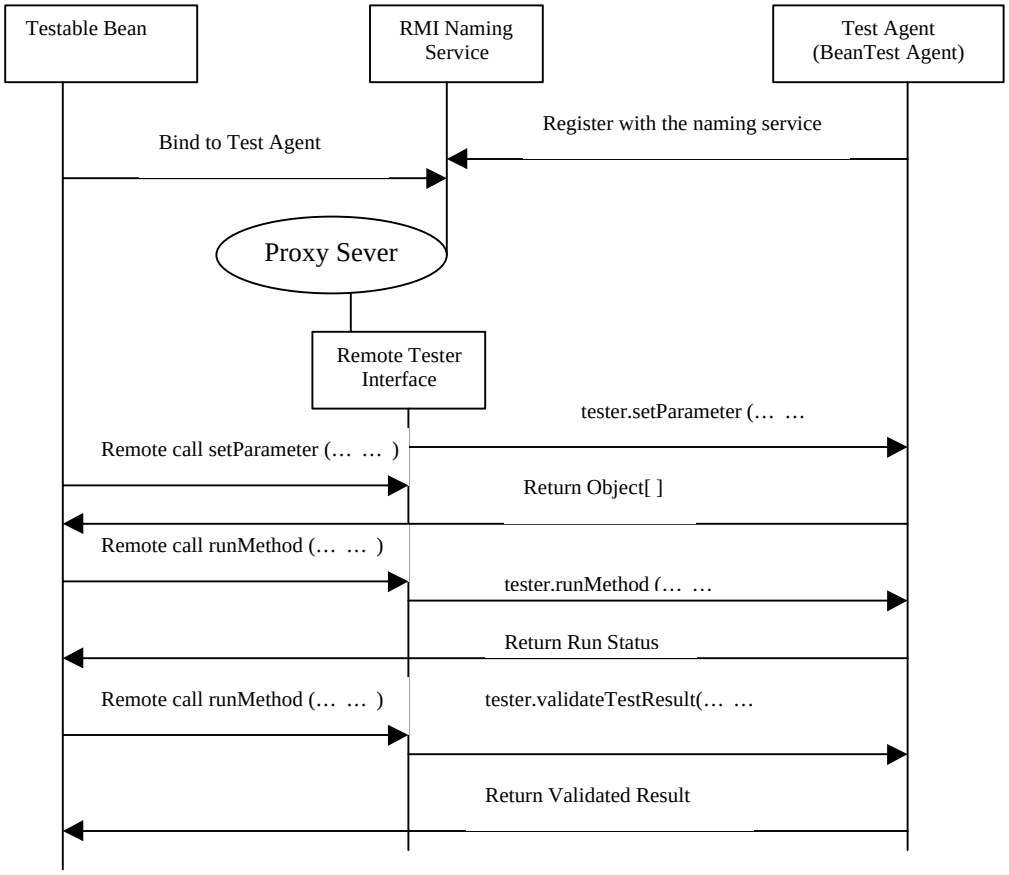


Figure 4. Transaction Sequence between a Test Agent and a Testable Bean

```
public class BeanTestAgent
{
   public static void  main(String args[])
   {
      // Create and install the security manager
      System.setSecurityManager(new RMISecurityManager());
      try
      {
         // Create the communication object to support the interactions between a test agent and
         // testable beans or test bean drivers
         BeanTesterImpl beanTestObj = new BeanTesterImpl("beanTestAgent");
      }
      catch (Exception e)
      {
         //Exception handling ...
      }
    }
}
```

Figure 5. Implementation of the Testing Agent

```
// import packages ...
public class BeanTesterImpl extends UnicastRemoteObject implements IbeanTester {
   BeanTest beanTestObj = new BeanTest();
   public BeanTesterImpl(String s) throws java.rmi.RemoteException {
      super();
      String name = s;
      try {
         Naming.rebind(name, this);   }
      catch (Exception e) {
         // Exception handling ...}
   }
public Object[] setParameters(String className_val, String methodName_val, TestData testData_val)   throws RemoteException
   { Object[] argValueList = beanTestObj.setParameters(className_val, methodName_val, testData_val);
    return argValueList;
   }
public int runMethod(String className_val, String methodName_val, TestData testData_val)throws NoSuchMethodException,
   SecurityException, java.rmi.RemoteException
   { int runStatus = beanTestObj.runMethod(className_val, methodName_val, testData_val);
      return runStatus;
   }
public int validateTestResult(String className_val, String methodName_val, TestData testData_val) throws java.rmi.RemoteException
  {
   int validateRes = beanTestObj.validateTestResult(className_val, methodName_val, testData_val);
   return validateRes;
  }
}
```

Figure 6. Implementation of the Communication Support

Implementation of Test Agent

We have implemented a simple prototype to try our idea. We use Java JDK 1.2 to construct our testable beans, and Java RMI technology to implement the communications between testable beans and the test agent. As we discussed before, our test agent consists of four parts: a) a communication interface, b) component test facility, c) administration and configuration graphic interface, and d) test tool linkage.

Here we only explain the first two parts. The other two parts will be reported in the future. We used Java and RMI technology to implement a test agent that can be used to support Java-based testable beans in a client-server environment. Figures 5 highlights the implementation idea for a simple test agent, where a communication interface object, called BeanTesterImpl, is constructed to support the communications between testable beans and a test agent. Figure 6 displays a sample of the communication interface object, which consists of binding function, and testing interaction operations.

```
//import packages ...
public class BeanTest implements IBeanTest
{
  public BeanTest(){}

   public Object[] setParameters(String className_val, String methodName_val, TestData testData_val)
   {// Detailed  implementation ...}

   public int runMethod(String className_val, String methodName_val, TestData testData_val)
   {// Detailed  implementation ...}

  public int validateTestResult(String className_val, String methodName_val, TestData testData_val)
  {// Detailed  implementation ...}

  // Types of objects supported
  public Class getClass(Object obj)  // method used in runMethod
  {// Used the Java Reflection  to determine the type of  the object such as "character", "Byte"," Short ","Integer ","Long ",
   // "Double"," Float ","Boolean " and the User defined type}
  }
}
```

Figure 7. Implementation of the Test Facility of Test Agent

The detailed implementation of component test facility is highlighted in Figure 7. The displayed test facility includes three basic functions, such as *setParameter(….), runMethod(…), and validateTestResult(…),* for component test operations. In addition, *getClass(Object obj)* is included to support *runMethod(…)* to determine the types of objects based on Java Reflection function.

4. Application Examples

Component developers and component testers can use the presented solution and test environment in the different ways:

- Add the built-in test code into a component to make it testable using the provided test interface and test facility.
- Use a systematic way to add the pre-defined test code into a component to make it testable based on the given test interface.
- Create a separate component test driver for a component using the provided test interface and test facility.
- Find a systematic way to generate test drivers for a component based on the given test interface.

Figure 8 shows two different ways to construct a testable component for a component.

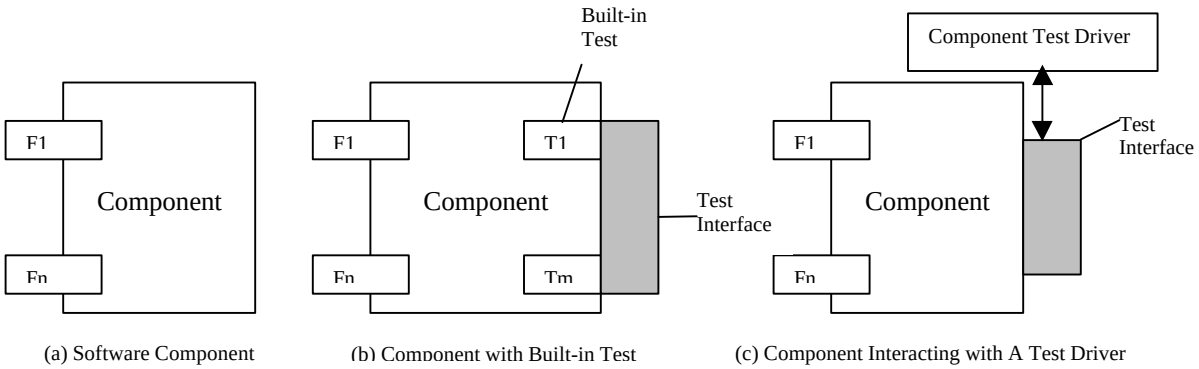


Figure 8. A Component and its Testable Components

A Component Test Driver:

A simple example presented in Figure 9 highlights the basic idea on how to generate a component test driver. This component driver uses the presented component test interface and the detailed component test facility provided in a test agent. It includes the following steps:

- Binding to a test agent through the RMI naming service.
- Get a test case from a test repository, or a graphic user interface
- Set up parameters through the remote test interface *(setParameters(…)).* It allows us to setup the parameters for exercising and validating different functions of a component.

- Invoke a component method using *runMethod(…).RunMethod(…)*uses Java "Reflection" to obtain the information about the classes and members of the given component. This is the technique that the JavaBeans "introspection" mechanism uses to determine the properties, events, and methods that are supported by a bean, for example. Reflection can also be used to manipulate objects in Java. We use the "Field" class to query and set the values of fields, the "Method" class to invoke methods, and "Constructor" class to create new objects.

- Validate the test results from a test case using *validateResult(…).* This method compares the execution result of the specified function in the given component with the given expected data. A validation status is returned to indicate if the test case has been passed or not. A test record can be generated and stored in a test management tool.

```
// import packages
public class BeanTestDriver
{   public static void main(String[] args) throws NoSuchMethodException, RemoteException
  {    String className, methodName;
       Object[] argValue_List, expectedTestData_List, parameter_List;

      // Bind to a test agent
     try
     {   // TestAgentName is given  through a  command line
         String TestAgentName = args[0];
         IBeanTester tester = (IBeanTester)Naming.lookup(TestAgentName);
     }
     catch (Exception e)
     {
        // Exception handling ...
     }
       // Program  code to get a component name (say a class name and method name).
       // Program code to retrieve a test case and test data  from a test repository or a tester
      // Programming code to set up argValue_List, and expectedTestData_list.

        //   Testing Code in a test driver
        TestData testDataObj = new TestData(argValue_List, expectedTestData_List);
        // Calling the setParameters
        parameter_List = tester.setParameters(className,methodName, testDataObj);
        //executing the method
        try   {
           int run_status = tester.runMethod(className,methodName, testDataObj);
        }
        catch(NoSuchMethodException ex)  { // Exception handling code …  }
        catch(RemoteException ex) {  // Exception handling code …    }
        int validateResult = tester.validateTestResult(className,methodName, testDataObj);            // Performing the validation
  }
}
```

Figure 9. An Example of a Component Test Driver

A for loop can be added in the example to support the execution of more test cases in a test repository. It is clear that the current example only can be applicable for function testing of software components.

```
// import packages
public class SampleComponent  // Component containing the built-in testing code
{
    public int sumMethod(int a, int b, int c) {
      int sum;
      sum = a + b + c;
      System.out.println("The generated Sum of 3 numbers is :" + sum);
      return sum;
   }
   public Function_Return_Type Function_1(… … ..) { … …   };
  public Function_Return_Type Function_2 (… … ..) {… … .};
 … … ..
  public Function_Return_Type Function_n (… … ) { … … . };

  // Built-in Test Code
   public static void Test_Script_I (String[] args)throws NoSuchMethodException, RemoteException
   {
      //  Set up a specific test case for sunMerthod(… )

     // Values for the className,methodName & arguments are supplied by the user either in the code or  through GUI
       String className = "SampleComponent";              // Set the class name
       String methodName = "sumMethod";                   // Set the method name
       Object[] argValue_List = new Object[3];            // Create the Object array for method arguments
       argValue_List[0] = new Integer(4);                 // Create the Object for the parameter value
       argValue_List[1] = new Integer(5);
       argValue_List[2] = new Integer(6);
       Object[] expectedTestData_List = new Object[1];  // Create the Object array for the expected test data
       expectedTestData_List[0] = new Integer(15);
       Object[] parameter_List = new Object[3];

       TestData testDataObj = new TestData(argValue_List, expectedTestData_List);
      // Set the TestAgentName with the actual Test Agent name.
      IBeanTester tester = (IBeanTester)Naming.lookup(TestAgentName);

      parameter_List = tester.setParameters(className, methodName, testDataObj); // Calling the setParameters

      try {   // Invoke a component function with a given test case
      int run_status = tester.runMethod(className, methodName, testDataObj); // Executing the method
      }
      catch(NoSuchMethodException ex)   { // Exception handling code …  . }
      catch(RemoteException ex) { // Exception handling code …   }

      // Validate test result
      int  validateResult  = tester.validateTestResult(className, methodName, testDataObj);    // Performing the validation
     }
}
```

Figure 10. A Component Example with Built-in Test

Components with Built-in Tests:

Figure 10 shows a simple example to explain how to extend a component to include a built-in test function, *Test_Script_I(…),* as a built-in test script for the component. It is clear that this

test script can be easily extended to handle more than one test case. The built-in code in this function consists of two parts: a) the programming code for setting up a test case, and b) the programming code that sets up and run a component function with the test case, and checks its execution result. The first part depends on a specific test case, and the second part is dependent on the test interface and the implemented test agent. It is obvious that the second part can be systematically added.

It is clear that adding built-in tests into a software component is a simple way to achieve the goal of run-and-test. However, there are several limitations. First, components with built-in tests usually consume more system resources. Next, the built-in tests of a component are usually a fixed-code. Thus, they are not easy to be configured and updated unless they are generated based on a test case repository. In addition, some test cases, such as performance test cases, are not suitable to use with this approach.

5. Conclusions and Future Work

In this paper, we discussed the testability of software components, and introduced a new concept for software components, known as *testable beans.* The purpose is to help engineers understand how to construct testable software components and increase component testability. We provide our solution to build testable components with a well-defined architecture model, and standardized well-structured component test interface. With this method we can generate testable software components in a systematic manner. In addition, we present our idea on how to set up a generic test environment to support testable beans. The two-level testing architecture based on *Testing Client* and *Testing Agent* provides the scalability. For example, *Testing Agent* on a machine is set up as a multithreaded program to support many testable components. Each thread is a lightweight process communicating with one component's tester. In addition, the *Testing Agent* provides a way to test components in a distributed environment.

The presented solution is applicable to both in-house components and third-party components. Its major advantages are summarized below:

- It helps engineers to construct testable components with small programming overhead.
- It generates testable components with a well-structured component architecture and a consistent interface to support software testing.
- It uses a client-server structure to provide a linkage to connect to different component test tools.

It provides a promising approach to get started on the road to test automation for software components.

Unlike other approaches, we focused on Java-based software components and provided engineers a well-defined testing framework to construct testable components. A flexible

supporting environment (including Testing Client, Testing Agent and RMI communication) are developed based on the testing framework.

Although we address several issues in developing testable components, several other issues have not been covered. One of the typical problems is how to provide a generic gateway in a test agent to link to software test tools, such as a test database system. The next problem is how to deal with complex parameters, such as class objects and arrays, in parameters set-up, and validation. The third issue has something to do with component performance testing, such as performance test model and analysis. The other issue is how to remove (or reduce) and control the test overhead of a component. The last issue is how to define and generate component test stubs in a systematic way.

We are developing a prototype environment for supporting component test automation. We are working on a tool to generate and test testable beans. Our tool is partially implemented and has already successfully verified our approach of creating a testable bean. We are extending the environment to provide a graphic user interface and a test tool gateway. In addition, we are investigating new models and methods to support component testing, component integration, and component performance analysis.

Our research suggests that software testability is dependent on component testability. Component testability is dependent on how well a component is designed and structured to facilitate component testing and software testing. We believe testable components must be defined with a standard component test interface and a well-defined architecture model for testing. Component test automation needs systematic solutions to generate, test, and support testable components.

6. Reference

[1] Elaine J Weyuker, "Testing Component-Based Software: A Cautionary Tale", ***IEEE Software,*** September/October 1998.

[2] Jerry Z. Gao, "Testing Component-Based Software", ***STAR'99***.

[3] Jerry Z. Gao, "Perspectives of Software Component Testability", submitted for publication in 1999.

[4] Jerry Gao, et al, "Monitoring Behaviors of Software Components", accepted by ***Journal of Object-Oriented Programming***.

[5] Voas, J. M. and Miller, Keith W.," Software Testability: The New Verification", ***IEEE Software,*** Vol. 12, No. 3: May 1995, pp. 17-28.

[6] Roy S. Freedman, "Testability of Software Components***", IEEE Transactions on Software Engineering,*** Vol. 17, No. 6, June 1991.

[7] David S. Rosenblum, "Adequate Testing of Component-Based Software", Department of Information and Computer Science, University of California, Irvine, Technical Report UCI-ICS-97-34, Aug. 1997.

[8] William T. Councill, "Third-Party Testing and the Quality of Software Components", IEEE Software, Vol. 16, No. 4, pp. 55-57, July/August 1999.

[9] Yingxu Wang, Graham King, and Hakan Wickburg, “A Method for Built-in Tests in Component-based Software Maintenance”, Proceedings of the Third European Conference on Software Maintenance and Reengineering, 1998.

QW2000 Paper 6W2

Ms. Lisa Crispin
(TRIP.com)

Stranger in a Strange Land -- Bringing Quality Assurance to a Web Startup

BACK TO QW2000 PROGRAM

Key Points

- How to get buy-in from management, information systems, development and marketing in a startup environment
- How to educate yourself in testing Web applications
- How to remove the testing bottleneck in Web development

Presentation Abstract

It perhaps goes without saying that a Web startup is not an environment in which quality testing is typically found. Development is fast and loose. Many developers are inexperienced. They're racing to be first to market. One might be tempted to label the environment as chaotic.

When I accepted the opportunity of being the first test engineer at TRIP.com, only 25 people worked for the Web startup. The developers had produced some exciting applications and felt they were ready to "grow up and play with the big boys." The development team thought they were intellectually prepared to introduce standards and procedures.

In reality, development was frenetic, and the developers didn't have a clue as to how to stop and analyze their processes, much less how to impose discipline on them.

For my part, I was a complete stranger to Web development. For years I had been testing databases, 4-GLs, and client/server software on UNIX, NT, and Windows platforms. I spoke ODBC, but not JDBC. I knew my customers. In my experience, the software development cycle had stretched on for months or even years-during which your typical Web application has gone though numerous incarnations.

This is the story of how I learned about Web application development, preached the quality gospel, and collaborated with the software and product developers and marketing managers to implement development standards and project processes that build quality into our applications. TRIP.com now employs 200 people, has three million registered customers, and has introduced such cutting-edge products such as intelliTRIP and companyTRIP.

About the Author

I have 18 years experience in the industry with the last 9 in Testing and Quality Assurance. I began my career as a programmer with the University of Texas Data Processing Division, where I helped develop the Online Catalog and circulation systems. My QA experience before my current job was with database, 4GL and client/server software on UNIX, NT and Windows at medium to large software companies " Software AG (now SAGA), Unidata (now Ardent Software).

I started my current job at TRIP.com in March of 1998 with little knowledge of the Web and no experience working in a startup environment. I was the first test engineer " no testing had been done previously even though the site had been online more than a year. The challenge of building quality into Web applications while meeting extremely tight development cycles has taught me much. We have grown from a test department of one (me) to seven test engineers, and I have been promoted to 'Quality Boss.' We test state-of-the-art, first-of-their-kind applications such as our intelliTRIP product which uses software from BEA Weblogic, StoryServer, and On Display to query partner airline Web sites for best fares. In addition, we play a key role in finding ways to build quality into applications and prevent errors before coding. To this end, I have been instrumental in implementing a configuration management tool as well as unit testing and memory/performance analysis tools. I work closely with Marketing and Product Management on process definition, formatting of specification documents, and planning.

Our latest testing challenge has been to move our Web sites to a new hardware and software configuration with two mirrored data centers running load-balanced clustered UNIX servers. TRIP.com has shown its commitment to quality by providing us with a test environment that exactly matches that of production: the same hardware, the same database, and a separate but equal network, so that we can effectively load test.

I have given successful presentations at both local and international user conferences to audiences of up to 60 people. I have many years experience training both technical and end users.

BACK TO QW2000 PROGRAM

Stranger in a Strange Land: Bringing QA to a Web Startup

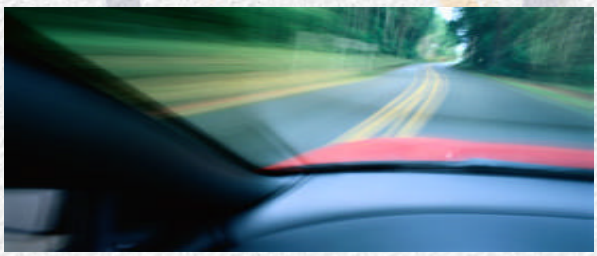
Lisa Crispin
Quality Boss, TRIP.com

1

Bringing QA to a Web Startup

What I found at TRIP.com

- Frenetic pace
- Lack of experience in process, standards
- Little comprehension of quality and testing



2

Bringing QA to a Web Startup

Who I was when I started at TRIP.com

- No experience in web testing
- Used to long development cycles
- Used to a corporate environment with lots of 'process'

3

Bringing QA to a Web Startup

This Presentation Will Give Tips On:

- How to get buy-in
- How to educate yourself
- How to prevent a testing bottleneck
- How to define a sane process
- How to create documentation

4

Bringing QA to a Web Startup

Get Buy-In

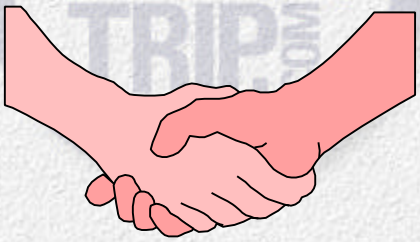
- Identify a top manager to champion your cause
  - Who understands your mission
  - In leadership position, eg. VP of Development

5

Bringing QA to a Web Startup

Get Buy-In

- Partner with someone outside of your group
  - Eg., project or operations manager
  - Educate them and garner their help



6

Bringing QA to a Web Startup

Get Buy-In

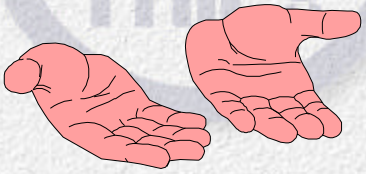
- Educate everyone about software quality assurance
  - Eg. - hold professional development session
  - What QA will do for them and the site
  - Meetings, face time

7

Bringing QA to a Web Startup

Get Buy-In

- Support Information Systems
  - Don't bug them for what you can do yourself
  - Let them do their job
  - They hold the keys and are on your team



8

Bringing QA to a Web Startup

Get Buy-In

- Listen to the developers
  - Try to understand their point of view
  - They may communicate differently
  - Quality Hero award

9

Bringing QA to a Web Startup

Get Buy-In

- Brainstorm
  - Developers, Database IS, Marketing, Customer Support
  - Potentially overlooked problems

10

Bringing QA to a Web Startup

Work Smart

- Evaluate and select appropriate tools
  - Test, defect tracking, configuration mgmt
  - Vendors who can help you
  - Consider learning and implementation time
  - Doesn't have to be expensive or famous

11

Bringing QA to a Web Startup

Work Smart

- Search the Web for resources
- http://www.softwareqatest.com/index.html
- http://www.kaner.com/writing.htm
- http://www.crl.com/~zaller/testing.html#Kerry

12

Bringing QA to a Web Startup

Work Smart

- Hire good help
  - Senior, experienced testers
  - Or train your own - attitude is key
  - Spend the $

13

Bringing QA to a Web Startup

Work Smart

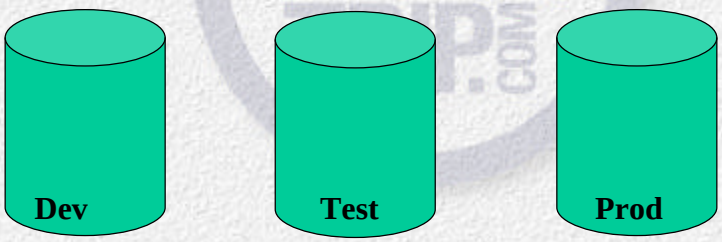
- Get input about quality from everyone
  - Form a quality board
  - Hold quality review panels

14

Bringing QA to a Web Startup

Work Smart

- Insist on a dedicated test environment
  - Looks like production
  - Can handle production level load test

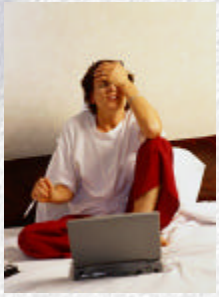


15

Bringing QA to a Web Startup

Work Smart

- Learn about the development tools
  - Quality issues - version control, workflow
  - Configuration and performance issues
  - Ask questions

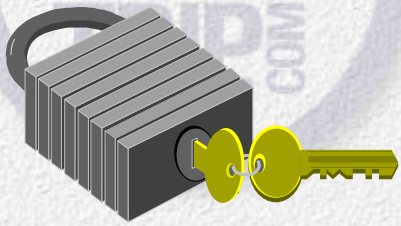


16

Bringing QA to a Web Startup

Define Process

- Get control of production environments
  - Lock developers out!
  - Work with Information Systems



17

Bringing QA to a Web Startup

Define Process

- Get involved from the beginning
  - Review all documentation
  - Agree on one vision for each project
  - Define QA and Dev process
  - Hire a good tech writer

18

Bringing QA to a Web Startup

Define Process

- Innovate!
  - New documentation formats
  - You have creative types who can help!

19

Stranger in a Strange Land:
Bringing QA To a Web Startup

Lisa Crispin, Quality Boss, TRIP.com

It perhaps goes without saying that a Web startup is not an environment in which quality testing is typically found. Development is fast and loose. Many developers are inexperienced. They're racing to be first to market. One might be tempted to label the environment as chaotic.

When I accepted the opportunity of being the first test engineer at TRIP.com, only 25 people worked for the Web startup. The developers had produced some exciting applications and felt they were ready to "grow up and play with the big boys." The development team thought they were intellectually prepared to introduce standards and procedures.

In reality, development was frenetic, and the developers didn't have a clue as to how to stop and analyze their processes, much less how to impose discipline on them.

For my part, I was a complete stranger to Web development. For years I had been testing databases, fourth-generation languages, and client/server software on UNIX, NT, and Windows platforms. I spoke ODBC, but not JDBC. I knew my customers. In my experience, the software development cycle had stretched on for months or even years— during which your typical Web application has gone through numerous incarnations.

This is the story of how I learned about Web application development, preached the quality gospel, and collaborated with the software and product developers and marketing managers to implement development standards and project processes that build quality into our applications. TRIP.com now employs two hundred people, has three million registered customers, and has introduced such cutting-edge products such as FlightTracker, intelliTRIP and companyTRIP.

TRIP.com a work in progress—we have a successful project process, but face continual challenges such as an inadequate test environment. Even when the whole company understands and is committed to the importance of quality assurance testing, , unexpected events can lead to surprises. The key is to keep plugging away at the following tasks:

- Get Buy-In
- Work Smart
- Define Processes

I. Get Buy-In

I won over my managers, developers, and marketing counterparts by following these tenets:

- **Identify** a top manager in your organization who believes in your cause and will champion it. In my case, it was the Vice President of Web Development and Chief Cat Herder (yes, that's really her title) . When I was hired, this person did not believe that five developers could keep one tester busy full time. But, in time, she became my biggest ally. She not only pushed the developers to work for quality, but she also lobbied the management team for testing resources.
- **Partner** with someone outside of your organization, such as a project or operations manager. Educate your ally to garner his or her help. We had a topnotch project manager who, once she understood what QA and testing would do for the company, did much of my job for me. She enforced processes such as document review and signoff, helped implement and police the defect tracking system, and tied up a million details involved with every big production launch. She became the prime channel of communication between marketing and development.
- **Educate** everyone you come into contact with at the company about software quality assurance—what it is and what it will do for them and the site. Early on, I held a "Lunch 'n' learn" professional

development seminar. The company bought lunch, so attendance was good. I explained why testing is essential and what it involves. Lots of meetings and one-on-one encounters are needed to get everyone on board and to establish priorities.

- **Support** Information Systems, the group that administers the production site. These people will benefit from not having their pager go off so much when applications are tested before being launched to production. The Vice President of Technology and the IS director fully supported me and refused to launch any update that had not been fully tested. This kept the rest of the organization from steamrolling over me, giving me a chance to prove the value of testing. I don't bug IS for the things I can do myself, but I let them do their job. For example, I installed Y2K patches on the test machines, but IS controls all the UNIX and NT account management.
- **Understand** the developers' point of view. You may have young, brilliant developers who don't communicate in ways you are accustomed to. Our original developers were mostly very young and inexperienced. Some had not finished high school!. Others weren't old enough to drink! The culture was anti-corporate, and they said what was on their minds. I found that if I listened, I learned, and they in turn were willing to listen to my ideas. I learned everything I now know about Web applications from the developers themselves. I present a "Quality Hero" award each month to a developer who has taken exceptional measures to prevent defects and improve quality. The prize is just a Nerf gun and the developer's name is added to the Quality Hero Award plaque, but it raises the visibility of high quality process and techniques.
- **Brainstorm** with developers and others about problems that may not come out in testing. For example, I didn't know that if you change a URL, search engines may not be able to find your site. When we implemented a content management tool that required changing every URL, this was important information!

II. Work Smart

Here's my advice for making the testing organization lean and mean:

- **Evaluate tools**. Put as much time as you can into tool evaluation, such as those for automated testing, defect tracking, and configuration management. Identify the vendors who can help you the most, and get as much information from them as you can. Ask fellow testers for their recommendations and experiences. Install new tools and try them out. Select tools that are appropriate for you and your company. It doesn't do any good to buy a tool you don't have time to learn how to use, especially if your testing team is small. I ended up choosing tools that are lesser known but stil meet our needs. For example:
  - For automated testing, we use WebART, an incredibly inexpensive, easy-to-learn , but powerful tool sold by OCLC Inc. (a non-profit company). They gave me invaluable advice and provided insights about Web testing. Pick everyone's brain including vendors!
  - For defect tracking, we chose a Web-based tool, TeamTrack, from a startup company . It, also is far less expensive than its competitors, but it is easy to implement and customize.
  - For configuration management, we again turned to a smaller, innovative company which produces an inexpensive, easy to implement and learn yet robust tool, Perforce.

  These tools won't necessarily meet *your* needs – just be open and creative when evaluating tools.
- **Search the Web for resources**. I would have quit after a week if I had not found an excellent Web site that points to information about testing Web applications and lists of tools. These sites, in turn, led me to more tools and information Here are some examples:

  http://www.softwareqatest.com/index.html
  Everything from basic definition and articles on how to test Web applications to comprehensive lists of Web tools to links to other informative sites.
  http://www.kaner.com/writing.htm
  Articles by Cem Kaner
  http://www.crl.com/~zallar/testing.html#Kerry
  long lists of associations, vendors, tools, training, reference information, conferences, interesting papers.

  Of course, user conferences are invaluable for both information-gathering and networking.

- **Hire good help**. It's difficult to find experienced test engineers in our area, so we hired bright but inexperienced people with the right qualities that make good test engineers: enthusiasm, dedication to the end user, and determination. A caveat: inexperienced testers who have no programming experience have a harder time learning a scripting language for an automated test tool. However, by using a combination of outside classes, hiring consultants and patient, one-on-one training, our testers have learned UNIX, SQL, test scripting, HTML, configuration management, and other technical skills. You're going to spend money either way – paying high salaries for experienced test engineers, or training novice testers.
- **Get input** about quality from all departments in the company. I formed a quality board with members from sales, marketing, customer support and travel to gather fresh ideas about error prevention and prioritization of regression testing. Whenever major problems occur, we hold a quality review panel where representatives from development and information systems hear short presentations from the people who experienced and fixed the problem. The panel studies the issues and recommends steps to prevent such problems from recurring. We also hold post-mortems after all major launches. By employing these methods, we have learned some invaluable lessons!
- **Insist on a test environment** that is exact replica of but is entirely independent from production. You can't emulate a production load without the equivalent of production hardware and software. This [what is a moving target?] is a moving target. The architecture is key too. If production is on a cluster, your test environment had better be on a cluster. If part of an application runs on a separate machine, it must do so in your test environment. This [what?] has been a constant source of problems for us. Even when the entire company is sold on the idea of a proper test environment, there are business and technical reasons (read: excuses) that get in the way of reproducing the production environment in for testing. Don't be complacent, and never give up. Make sure you have the best test environment you can get for each application going into production, and work actively with your information systems team to get the environment you really need. Even small applications can deceive you. For example, last December, we launched a simple application, SantaTracker, on Christmas Eve so kiddies could watch Santa's sleigh fly around the country. We could not test this on the same architecture that it was to run in production, but we weren't concerned. After all, it should have worked exactly like our regular FlightTracker application! Right? Wrong! It was a disaster!

  In short: Dig your heels in and refuse to launch until some semblance of a test environment is established. Use the SantaTracker example. And remember, it is harder to get the test environment once the new application is in production.
- **Learn about the development tools**. They present their own quality issues and offer some solutions, too. For example, the first version of our content management tool did not have any version control. It took more than a year to upgrade to the release that offered this capability, and even now it doesn't ***enforce*** version control. We have to constantly police the process to ensure that developers version their code. The software on which our Java applications are based has complex configuration parameters we didn't fully understand when we first put it into production. We had tested our intelliTRIP product with a production load in terms of transactions per second, but never with a realistic number of concurrent users. As a result, the servers kept crashing on the first day we put it in production. If we had understood the configuration and the user session management parameters better in our Java-application management tool better, we could have prevented this problem.

III. Define Processes

Collaborate with your counterparts to formalize your processes:

- **Get control** of the production environments. Work with your information systems team to create a production update procedure. When I started, developers launched their own changes to production. It was hard to wean them away from this bad habit. Even after we thought we had implemented good production update procedures, we lacked the discipline to enforce it under pressure. For example, since we did not have good configuration management, it became

impossible to build baselines of intelliTRIP, so developers would simply move new classes into production to fix problems. Only now—after two years—are we beginning to be able to require developers to build scripts and installation documentation before we accept any software from development for testing.

- **Get involved** from the beginning of each project. This is hard work. It forces you to juggle many tasks, but it is essential. Participate in all documentation reviews: Those for requirements, functional specifications, and design specifications. Make sure the documents are complete and clear. Look for ambiguities, gaps, lack of detail. All parties, including marketing, development, test, customer support, sales must agree on a vision for the product. This vision is a short phrase that describes the main thrust of the product. With intelliTRIP, development and test were told to produce a server-side version of the original client-side product as quickly as possible. Sales and marketing believed that the purpose of the product was to ***quickly*** locate fares from airline Websites. Since development and test wasn't told that part about ***quickly!*** We released the product even though we knew that is was sometimes slow to return results. This type of disconnect can be prevented by including a vision statement in the requirements
- **Define quality**. Work with marketing and product development to define quality IS for each product.: Should the priority be good, fast, or cheap? Even if you choose fast, don't sacrifice the process. We recently implemented a promotion that marketing believed to be simple and wanted to rush to production. Since the product manager did not hold documentation reviews and get signoff, the HTML pages produced by the developers had to be changed three times. This took much longer than a documentation review meeting. There is no need to get bogged down in process either. If you find that is happening, change the process, or train people how to use it properly.
- **Document** the internal processes of both test and development. You can't expect marketing and product development to follow best practices if you don't do it yourself. One of our development directors, with the help of our technical writer, is leading an effort to define and document the development process. By the way—no matter the size of the company, you need at least one experienced, skilled technical writer in your development organization.
- **Enforce the process.** If your QA team is large enough, dedicate one person to administering and enforcing configuration management and delivery of installation scripts and documentation. Don't accept software to test if it is not accompanied by all the documentation and software you need to promote, test, and launch it to production.
- **Innovate!** Look for ways to present documentation such as functional and specifications. Web applications require a new approach. You have creative people at your company who can help! Documentation format is a constant challenge for us. Our user interface team came up with a UI-oriented format for specifications that lends itself to front-end Web applications. If this is adopted as the company standard, we may use a similar format for our test plans. Get input from as many different groups as you can.

Summary – As You Grow

All companies change as they grow beyond the 'startup' size and environment. My team and I have endured countless frustrations when fast growth led to temporary chaos. As your organization grows, educate new employees about project process and quality practices. Listen to them and take advantage of their fresh outlook and new ideas. Take the initiative – if there is a gap due to a re-organization, fill it yourself. For example, when we were temporarily without a project management function in the company, the development director and I set up a weekly tactical meeting with representatives from all departments so that everyone could stay informed and juggle resources. Quality assurance can be a frustrating job, especially in a Web startup – pick your battles , keep striving for better quality and enjoy the experience!

QW2000 Paper 6M2

Mr. Brian Lawrence & Ms. Johanna Rothman
(Coyote Valley Software / Rothman Consulting, Inc.)

Testing in the Dark

BACK TO QW2000 PROGRAM

Presentation Abstract

A pragmatic approach to overcoming undocumented requirements.

This is a presentation based on the article that Johanna and I wrote for Software Testing and Quality Engineering Magazine, Volume 1, Issue 2, in March/April of 1999. To date, Testing in the Dark has been the most commented upon article that STQE Magazine has published.

In this presentation we discuss an analytical technique for discovering undocumented requirements, how to refine and distill rough requirements to obtain their essence, and how to base your acceptance testing on the refined requirements.

About the Author

Brian Lawrence (Coyote Valley Software) teaches and facilitates requirements analysis, peer reviews, project planning, risk management, life cycles, and design specification techniques.

He has served as a program chair for the SEPG'97 Conference as well as the IEEE Computer Society's 1998 International Conference on Requirements Engineering. Brian is the technical editor of Software Testing and Quality Engineering Magazine and serves on the editorial board of IEEE Software.

Johanna Rothman (Rothman Consulting, Inc.) observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people.

Johanna is the founder and principal of Rothman Consulting Group, Inc., and is ASQ certified as a Quality Auditor and Software Quality Engineer.

BACK TO QW2000 PROGRAM

Testing in the Dark

A Pragmatic Approach to Overcoming
Undocumented Requirements

Brian Lawrence
Coyote Valley Software
brian@coyotevalley.com
www.coyotevalley.com
(408) 578-9661

Johanna Rothman
Rothman Consulting Group, Inc.
jr@jrothman.com
www.jrothman.com
(781) 641-4046

© 1999-2000 by Brian Lawrence and Johanna Rothman. All Rights Reserved.

Acknowledgement

This presentation is based on the article

"Testing in the Dark,"

by Johanna Rothman & Brian Lawrence,

published in

Software Testing & Quality Engineering,
Volume 1, Number 2, March 1999.

© 1999-2000 by Brian Lawrence and Johanna Rothman

Testing in the Dark

- You're surprised by the request
  - "Here, test this..."
  - "Just go over this lightly"
- How do you know what the product is supposed to do?
- How are you supposed to know if it's doing it?

Turn on some lights

1. Discover the requirements
   - to know what testing needs to be done
2. Define what quality means to the release
   - to know how much time and effort to apply to testing
3. Define a test plan including release criteria
   - to know when we're ready to ship

© 1999-2000 by Brian Lawrence and Johanna Rothman

Part 1: Discover the requirements

- Play detective for this release's requirements
  - Gather data
  - Transform the data
  - Filter the data
- Requirements drive design choices

Customer requirements

- Users
  - Who is affected by the product, just by its existence?
  - Who uses the product?
- Attributes
  - How reliable, how fast?
  - What characteristics do users need?
- Functions
  - What does your product do for the users?

© 1999-2000 by Brian Lawrence and Johanna Rothman

The test planner' s job

- Find the different pieces of the picture that created this disk.
- What decisions were made?
- Did the requirements get implemented in the product?

Testing in the Dark - Quality Week 2000 7

Gather data to determine design choices

- Read the documentation
- Examine the product's architecture
- Run the product
- Ask the developers
- Ask the release manager

Testing in the Dark - Quality Week 2000 8

Transform " goodies" into requirements

- You get statements, you convert them into requirements
- Organize requirements by User, Attribute, Function
- Example: "Provide 5 day per week up-time."
  - this is about *reliability*, an attribute

Requirements Snapshot

Type	Requirement Specifics
Users	***Operator*** *– Person who administers the system.* ***Guest*** *– Visitor, but not a normal user.* ***Intruder*** *– Someone not authorized to use the system. This user is disfavored.*
Attributes	***Reliable*** *– The system is available; it never fails, 24 hours a day on weekdays.* ***Secure*** *– Only authorized users may access this system.* ***Integrated*** *– This system brings together different components.* ***Remotely accessible*** *– There are ways to reach this system other than by the standard interface.* ***Easy-to-install*** *– Installing this system is simple and intuitive.*
Functions	***Feedback*** *– Operators and users get indications about how their interaction with the system is working.* ***Inventory*** *– Keep track of operating assets.* ***Report*** *– Collate data and produce reports.* ***Schedule*** *– Show future events.* ***Plan*** *– Support selecting, trading-off, and scheduling system events.*

© 1999-2000 by Brian Lawrence and Johanna Rothman

Make the requirements testable

- MTBF: Desired level is 168 hours
- Testable
  - Specific
  - Measurable

Part 2: What quality means to this release

- Rank your release's goals
  - Time to market
  - Feature set
  - Low defect levels
- Choose one as your top priority, manage the other two in that context
  - Helps you define your test plan

© 1999-2000 by Brian Lawrence and Johanna Rothman

Requirements with quality ranking

Type	Requirement Specifics	Critical for TTM?
Users	***Operator*** *– Person who administers the system.*	***Yes***
	Guest *– Visitor, but not a normal user.*	***No***
	Intruder *– Someone not authorized to use the system. This user is disfavored.*	***Yes***
Attributes	***Reliable*** *– The system is available, 24 hours a day. Weekdays only.*	***Yes***
	Secure *– Only authorized users may access this system.*	***Yes***
	Integrated *– This system brings together different components.*	***No***
	Remotely accessible *– There are ways to reach this system other than by the standard interface.*	***Yes***
	Easy-to-install *– Installing this system is simple and intuitive.*	***No***
Functions	***Feedback*** *– Operators and users get indications about how their interaction with the system is working.*	***No***
	Inventory *– Keep track of operating assets.*	***No***
	Report *– Collate data and produce reports.*	***Yes***
	Schedule *– Show future events.*	***No***
	Plan *– Support selecting, trading-off, and scheduling system events.*	***No***

Testing in the Dark - Quality Week 2000 13

Part 3: Define a test plan including release criteria

- Choose what to test, what not to test
- Use release criteria to illuminate testing and product goals
  - Must we meet this requirement by the requested release date?
  - What is the effect on our customers if we do not meet this requirement by the release date?
- Possible release criteria
  - Attribute: Reliability: Verify 5x24 uptime.
  - Function: Report: Produce reports over a 1-week period.
  - User: Operator functionality complete.

Testing in the Dark - Quality Week 2000 14

Reasonable criteria

- Surface assumptions and fears
- Allow you to resolve those assumptions and fears before you release the product

Success tips

- Clarify what management gets from their testing dollars and time
- Release criteria belong to the whole release
- Release the product based on the release criteria and no other criteria

© 1999-2000 by Brian Lawrence and Johanna Rothman

The Dark Side

Possible mistaken assumptions

- Testing-in-the-dark is a workable alternative at all
- You work in an open communication environment
- People want to know the truth

When you find yourself in the dark

- Uncover the requirements,
- Decide what quality means,
- Define a plan and release criteria, so you can measure when you've met your plan.

Lead yourself back into the light

© 1999-2000 by Brian Lawrence and Johanna Rothman

References

James Bach, "Good Enough Software: Beyond the Buzzword", *IEEE Computer* (Software Realities column), August 1997.

Brian Lawrence, “Requirements Happens...”, *American Programmer*, April 1997, Vol. 10. No. 4.

Johanna Rothman, "Defining and Managing Project Focus", *American Programmer* , February 1998, Vol. 11. No. 2.

Johanna Rothman and Brian Lawrence, “Testing in the Dark”, *Software Testing and Quality Engineering*, Vol. 1, No. 2

© 1999-2000 by Brian Lawrence and Johanna Rothman

QW2000 Vendor Technical Paper 7V1

Greta Cohen and Jim Baack
(Superior IS)

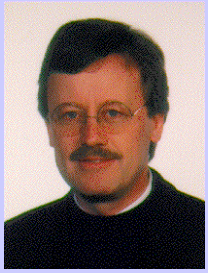
Outsourced Testing for the Web

BACK TO QW2000 PROGRAM

Key Points

- Outsourcing web testing to speed delivery and increase quality
- Aspects of functional testing for web systems
- Web performance testing services

BACK TO QW2000 PROGRAM



QW2000 Paper 7T1

Dr. Andreas Spillner & Dr. Ulrich Breymann (Hochschule Bremen)

Semantic Differences Between C++ and Java: Consequences for the Review and Test Process

BACK TO QW2000 PROGRAM

Key Points

- Main conceptual differences between C++ and Java
- Identification of main issues arising from migration from C++ to Java
- Recommendations for analysis, review and test

Abstract...

More and more projects use Java as an object-oriented implementation language. Often there is already knowledge of C++ and because of the syntactical similarity of the programming languages migration seems to be simple. Sometimes this is not the case due to important conceptual differences between C++ and Java. Selected problems show the implications for program analysis, understanding and testing. Early detection of problems is desirable, so the paper provides a checklist for the test managers.

Author Bio...

Dr. Spillner is currently working as Professor at the Hochschule Bremen (University of Applied Sciences) where he is responsible for software engineering and real time systems. Dr. Spillner has over 20 years experience in software development and teaching. He has been involved with development projects for government and also research projects in collaboration with industry. He has a degree in computer science from the Technical University of Berlin and received his PhD on the dynamic integration testing of modular systems from the University of Bremen in 1990. He has also worked as Research Assistant and as Associate Professor at the University of Bremen.

Dr. Spillner is currently the chairman of the German Computer Society Special Interest Group for the Testing, Analysis and Verification of Software Systems. (GI-TAV). His research interests include the validation of software, test methods (especially for large software systems) and the testing of object-oriented software systems.

Dr. Breymann is currently with Hochschule Bremen (University of Applied Sciences) as a professor

with the special focus on software engineering and object orientation. Before that he worked several years in industry as analyst and project leader, among others on the subjects process control systems for power plants and software for the international space station project. He was a member of the German working group for the standardization of the programming language C++.

BACK TO QW2000 PROGRAM

Semantic Differences between C++ and Java Consequences for the Review and Test Process

Ulrich Breymann
Andreas Spillner

University of Applied Sciences Bremen
Germany

Phone: ++49 421 5905 –3425 / –3467
Fax: ++49 421 5905 –3412
{breymann,spillner}@informatik.hs-bremen.de

Agenda

- Introduction
- Important differences between C++ and Java (pitfalls, examples and recommendations):
  - Assignment operator
  - Equality operator
  - Immutable objects
  - Argument passing
  - General rules
  - Clean up
  - Polymorphic behaviour
- Check list for code inspection
- Conclusion

2

Introduction

- A strong understanding of object-oriented concepts is necessary.
- Java is more and more the language of choice for a lot of applications.
- The syntax is similar to that of C++, but much simpler, which makes learning and using it easy.

- This talk focuses on the following subjects:
  - Possible pitfalls in program understanding for developers with C++ experience, important for the coding and 'developer test'-phase of a project.
  - What are the special issues when Java code is reviewed during testing by a C++ expert?

3

Assignment operator

Primitive data types

```
// Java and C++
int a;
int b = 4;
a = b;
a = 5;                    // b is not changed here
```

4

Assignment operator

class objects in Java

```
// Java
thisArticle = thatArticle;
                    // Two objects of class Article

thisArticle.changeFont("Helvetica");
                    // Changes thisArticle and thatArticle
```

5

Assignment operator

```
// Java, but C++ like behaviour of assignment

thisArticle = (Article)thatArticle.clone();
                        // deep copy

thisArticle.changeFont("Helvetica");
                        // Does not change thatArticle!
```

6

Assignment operator
Recommendation

- Assigning class objects in Java does not mean making a copy.
- It rather creates an alias for the right hand side object and passes the left hand side object directly to the garbage collector.
- You can focus on this problem by code inspection:
  - Find any assignment to class objects in the code and check the correct use.
  - You can also use a (self-written or off-the-shelf, if there is one) code analyser to list the alias names of all objects.

7

Equality operator

In C++, the equality-operator compares two objects and returns true if they are equal. What does that mean? For our purposes, we define here:

- Two objects are *identical,* if they cannot be distinguished.
- Two objects are *equal*, if they are of the same type and are in the same state with respect to *all or some selected properties*.

8

Equality operator

```
// Java
Integer I1  = new Integer(77),
        I2  = new Integer(77);
int  i1 = 3, i2 = 3;
Integer I3 = I2;

i1 == i2          // true
I1 == I2          // false!

I1.equals(I2)     // true
I3 == I2          // true
I3.equals(I2)     // true
```

9

Equality operator
Recommendation

- In Java, operator **==** yields false if equal, but not identical class objects are compared.
- It returns true only if the compared *references* have equal values, i.e. the underlying objects are identical.
- You can focus on this problem by code inspection:
  - Find any comparison between class objects with operator **==** in the code and check if they should be replaced by the **`equals()`**-method.
  - You can also use a code analyser to list all comparisons.

10

Immutable objects

```
// C++
const string s = "hello";
s.append(" world!");          // Error! s is const
s += " world!";               // Error! s is const
// Java
String s = new String("hello");
```

Java strings are immutable and there is no method to change a **`String`** object. But ...

```
s += " world!"; // This is ok for Java (equals s=s+" world!";)
System.out.println(s);        // Output: hello world
```

11

Immutable objects

Remember: **`s`** is a reference, not the object! References to immutable objects are not immutable themselves. They can be made immutable:

```
final String s = new String("hello");.
s += " world!";               // Error! s is immutable
```

But what about calling a state-changing method?

```
// Java
final MyClass aMyClass = new MyClass();
aMyClass = anotherMyClass;
                  // Error! Reference aMyClass is immutable
aMyClass.setAttribute(17);
                  // (Object attached to) aMyClass is changed!
```

12

Immutable objects

Only for primitive data types, **`final`** means the same as **`const`**.

There are some possible workarounds in Java to make also class objects immutable. These workarounds are more or less clumsy and expensive in terms of runtime.

- *State checking*
- *Proxy*
- *Common Baseclass*

13

Immutable objects
Recommendation

- Trying to implement the C++ **`const`** for class objects in a Java program is a little bit complicated and therefore will probably introduce errors instead of preventing them.
- If a C++ program is to be rewritten in Java, we recommend just to ignore the **`const`** keyword for non-primitive data types.

14

Argument passing

```
// same syntax in Java and C++, but different meaning

void f(Window w) {
  w.setTitle("new Title");
}
```

15

Argument passing

```
// Java (wrong example)
void fullFilename(String path, String filename,
                  String result) {
// Error: caller's result is not modified, only a local copy of result:
result = path + filename;
}
```

The (copied) reference of **`result`** is changed, not the object it refers to!

Correct example:

```
// Java (now ok, but different interface)
String fullFilename(String path, String filename) {
return new String(path + filename);
}
```

16

Argument passing
Recommendation

- Be sure that the parameter passing mechanism is used correctly.
- Is there an assignment to a parameter within a method, it is only a assignment to the local copy of the parameter. It has no effect for the caller of the method, so there must be a special reason, which should be documented in a comment – otherwise it is wrong and the parameter should be declared as **`final`**.
- Use code inspection to find such cases.

17

General rules

C++ example	**direct semantic Java equivalent**
`myClass x;`	None, there are no automatic (stack) class objects in Java (only references)
`myClass* p = 0;`	`myClass p = null;`
`p = new myClass;`	`p = new myClass();`
`p->method1()`	`p.method1()`

18

General rules

C++ example	direct semantic Java equivalent
declaration: `void f(myClass)`	`void f(myClass)`
call: `f(*p)`	`f((myClass)p.clone())`
declaration: `void g(myClass*)`	`void g(myClass)`
call: `g(p)`	`g(p)`
declaration: `void g(myClass&)`	None, there is no passing by reference in Java
call: `g(x)`	

19

General rules

C++ example	direct semantic Java equivalent
`myClass* const q = new myClass;`	`final myClass q = new myClass();`
`const myClass* q = new myClass;`	None

20

General rules

Java example	direct semantic C++ equivalent
`String s = "hello";`	`const string* s = new string("hello");`
`s = "abc";`	`delete s; s = new string("abc");`
`char c = s.charAt(0);`	`char c = s->at(0);`

21

General rules
Recommendation

- C++ and Java look similar, but are different.
- Java is often regarded as ”C++ without pointers,” which obviously is not true.
- Java is more like ”C++ with dynamic (=heap) class objects only”.
- If your company switches from C++ to Java, special training of developers is a must. They have to know the semantic differences of both languages.

22

Clean up

```
// C++
{
    Object myObject;    // Constructor acquires resources
    myObject.doSomeThing();
}                       // Destructor is called here automatically
                        // Its task is to release resources
```

In Java, the **`finalize()`** method roughly corresponds to the destructor in C++, but it has to be called explicitly, and it is left to the JVM (Java Virtual Machine) when **`finalize()`** will be called.

23

Clean up

```
// Java
{
   myClass myObject = new myClass();
                        // Constructor acquires resources
   myObject.doSomeThing();

   // recommended add-on:
   myObject.cleanup();          // Release resources now!
   myObject = null;   // Release object for garbage collector
                      // and prevent further access
                      // (if more code is to follow)
}  // The garbage collector can do the rest, when it wants to!
```

24

Clean up - a safer approach

```
{   // begin of Java-block
    myClass myObject = null;
    try {
      myObject = new myClass(); // might throw exception
      myObject.doSomeThing();   // might throw exception
    }
    catch(Exception e) {
        // exception handling goes here
    }
```

... continued

25

Clean up - a safer approach

```
    finally {
      if(myObject != null) {
        myObject.cleanup(); // release resources
        myObject = null;    // release object
      }
    }
}   // The garbage collector can do the rest, when it wants to!
```

26

Clean up
Recommendation

- If the immediate release of resources (other than the memory of the object itself) is important, you have to do it by yourself.
- The Java Virtual Machine does not guarantee a defined moment for release.
- Use code inspection to find critical classes and places.

27

Polymorphic behaviour

- All methods with the exception of static methods are virtual in Java.
- Static methods are class methods.
- This is no problem for those C++ people who write in good style, i.e. they
  - never override non-virtual C++ methods and
  - always call static methods by a class name, not an object's name.
- The second is also our recommendation for Java.

28

Check list for code inspection

- Assignment of class objects:
  Shall an alias (1) be created or a copy (2)?
  1) ok
  2) use **`clone()`**

- Comparison of class objects using:
  Check of identity (1) or equality (2) intended?
  1) ok
  2) use **`equals()`**

29

Check list for code inspection

- Does parameter passing conform to the ”General rules” above?
  If not, rethink and recode.

- Is there a local parameter on the left hand side of an assignment in a method?
  If yes: It has no effect for the caller of the method, so there must be a special reason, which should be documented in a comment, otherwise it is wrong.

- End of scope: Is an immediate release of resources required?
  If yes, use a special self-written method for release.

30

Check list for code inspection

- Is reproducing the C++ **`const`** in a Java program really necessary for class objects?
  If not, avoid introducing unnecessary code complexity.

- Are class methods only called by using the class name?
  If not, rethink and recode.

31

Conclusion

- There are syntactically identical elements in the programming languages C++ and Java, where Java uses reference semantic, but C++ employs value semantic.

- Developers have to know the semantic differences between these similar syntactic elements.

- On the pretended simple way from C++ to Java there lie some stumbling blocks. Overlooking them can lead to unforeseen errors.

32

Semantic Differences Between C++ and Java: Consequences for the Review and Test Process

Ulrich Breymann and Andreas Spillner

Hochschule Bremen
University of Applied Sciences
Neustadtswall 30
D-28199 Bremen
Germany

Phone: ++49 421 5905 –3425 –3467
Fax: ++49 421 5905 –3412
{breymann,spillner}@informatik.hs-bremen.de
http://www.informatik.hs-bremen.de/~brey/
http://www.fbe.hs-bremen.de/spillner/

Abstract

More and more projects use Java as an object-oriented implementation language. Often there is already knowledge of C++ and because of the syntactical similarity of the programming languages migration seems to be simple. Sometimes this is not the case due to important conceptual differences between C++ and Java. Selected problems show the implications for program analysis, understanding and testing. Early detection of problems is desirable, so the paper provides a check-list for the test managers.

Key words:
object-oriented programming languages, C++, Java, migration, program understanding, review

Introduction

The object-oriented programming language Java is more and more the language of choice for a lot of applications. Its platform independence and networking capability promise easier solutions than other programming languages for a broad range of problems. Java [GJS96] appeared at the right time and has a lot of capabilities also beyond the Internet. The syntax is similar to that of C++ [ISO98], but much simpler, which appears to make learning and using it easy.

Migration from C++ to Java has to consider a lot of different things. A strong understanding of object-oriented concepts is necessary, because procedural programs are not possible – an often occurring misuse of C++. Event-driven GUI development and development of distributed applications are main features of Java programs. An effective migration strategy has to include training and self-education, depending on the kind of Java users, for example software developers, software architects, analysts or project managers [SU99]. Our paper, however, focuses on the following subjects:

- Possible pitfalls in program understanding for developers with C++ experience, important for the coding and 'developer test'-phase of a project.
- What are the special issues when Java code is reviewed during testing by a C++ expert?

Reviews and inspection had and have the best influence on software engineering, a known fact, which was emphasized again recently [McC00]. Of course, if developers and reviewers have got sufficient cross-training, there will be no big problem. But have they? We used to teach C++ as the first programming language at our institution, the University of applied sciences Bremen, but switched to Java only recently. Many advanced students, having a sound knowledge of C++, now use Java in study projects, learning Java by themselves, reading a textbook, and starting to code quickly. We realize that they often make some special mistakes in Java. Therefore we consider the migration from C++ to Java not as easy as one might think at a first glance. We assume that this problem also occurs in the industrial practice, if software is written or re-written in Java by developers with C++ experience.

Some but not all of the difficulties arise, because a Java program is determined by a reference-semantic behaviour of objects like in Smalltalk, whereas the expectations of Basic/C/C++/ Pascal-programmers are based on a value-semantic behaviour. These wrong expectations lead to errors and confusion. Examples demonstrate selected problems and we propose a few hints in order to facilitate the detection of errors.

Caught in the assignment act

Programmers familiar with other languages like Pascal, Basic or C++ approach Java assuming that it behaves in a similar way. So the statement `A = B;` is intended to copy object `B` to object `A` with the result of two equal objects. Furthermore, it is expected that a method call `A.method1();` has no effect on object `B`. Primitive data types behave as known from C++:

```
// Java and C++
int a; int b = 4;
a = b;
a = 5;                                  // b is not changed here
```

However, this is not the case with *class objects* in Java:

```
// Java
thisArticle = thatArticle;            // Two objects of class Article.
thisArticle.changeFont("Helvetica"); // Changes thisArticle and thatArticle!
```

The second statement changes also the font of `thatArticle`! The reason is of course, that assignment does not modify an object, but the reference to an object. The result of such an assignment is that there is *only one object*, which can be accessed via *two different names* (aliases).

If we want C++-like behaviour, we have to copy the right-hand-side object (this is also called ”deep copy” in contrast to the ”shallow copy” above):

```
// Java, but C++ like behaviour of assignment
thisArticle = (Article)thatArticle.clone(); // deep copy
thisArticle.changeFont("Helvetica");        // Does not change thatArticle!
```

The cast is necessary because the type of `clone()` is `Object`, the top superclass which is inherited by all other classes. Cloning objects is discussed more in depth in [B00].

Recommendation
Assigning class objects in Java does not mean making a copy. Instead, it creates an alias for the right hand side object and passes the left hand side object directly to the garbage collector. You can focus on this problem by code inspection. Find any assignment to class objects in the code and check the correct use. You can also use a (self-written or off-the-shelf, if there is one) code analyzer to list the alias names of all objects.

Equality is not equality

In C++, the equality-operator compares two objects and returns true if they are equal. What does that mean? For our purposes, we define here:

- Two objects are *identical*, if they cannot be distinguished.

- Two objects are *equal*, if they are of the same type and are in the same state with respect to *all or some selected properties*. For example, two `name`-objects (Peggy Sue, Mrs) and (Peggy Sue, Dr) are considered equal for the purpose of sorting, when they are alphabetically listed and titles don’t play any role.

Identical objects are therefore always equal, but the reverse is not true. Applying these definitions to objects in Java or C++ programs, we declare those objects exactly occupying the same address space as identical, i.e. there is actually only one object, maybe with several names. Of course, these definitions depend on the context and may not hold if we think of persistent objects. Objects with different memory addresses are distinguishable (not identical) and can independently change their states. These objects may be equal (or not). In Java, the equality operator == checks *identity* for class objects and *equality* for primitive data types. Equality of class objects is checked by the `equals()` method as can be seen in the example:

```
// Java
Integer I1  = new Integer(77),    // class objects
        I2  = new Integer(77);

int i1 = 3,                       // primitive types!
    i2 = 3;

Integer I3 = I2;                  // Create another reference for I2.

i1 == i2;           // true      (primitive type: not identical, but equal)
I1 == I2            // false!    (not identical)

I1.equals(I2);      // true      (equal)
I3 == I2            // true      (identical)
I3.equals(I2);      // true      (equal)
```

This and some of the following points are discussed more in depth in [A98]. Of course, each author of a class has to write a special `equals()` method, if objects of this class shall be checkable for equality. If she does so, the method overrides the corresponding method of the top class `Object`, which is inherited automatically by all classes.

Recommendation
In Java, operator == yields false if equal, but not identical class objects are compared. It returns true only if the compared *references* have equal values, i.e. the underlying objects are identical. You can focus on this problem by code inspection. Find any comparison between class objects with operator == in the code and check if they should be replaced by the `equals()`-method. You can also use a code analyzer to list all comparisons.

Constant problem or final confusion?

If the `const` keyword is used properly in C++, it helps to create safer programs. The compiler will flag inadvertent incorrect uses of constant or immutable objects. In Java, the keyword `final` prevents changing an object after initialization. But `final` is not `const`!

Example:

```
// C++
const string s = "hello";
s.append(" world!");          // Error! s is const
s += " world!";               // Error! s is const

// Java
String s = new String("hello");
```

Java strings are immutable and there is no method to change a `String` object. But ...

```
s += " world!";               // This is ok for Java (equals s = s + " world!";)
System.out.println(s);        // Output: hello world
```

Remember: `s` is a reference, not the object! References to immutable objects are not immutable themselves. They can be made immutable:

```
final String s = new String("hello");.
s += " world!";                // Error! s is immutable
```

But what about calling a state-changing method? Have a look at:

```
// Java
final MyClass aMyClass = new MyClass();
aMyClass = anotherMyClass;     // Error! Reference aMyClass is immutable
aMyClass.setAttribute(17);     // (Object attached to) aMyClass is changed!
```

Only for primitive data types, `final` means the same as `const`. There are some possible workarounds in Java to make also class objects immutable. These workarounds are more or less clumsy and expensive in terms of runtime. Some of them are listed here:

State checking
Introduce a boolean attribute to the class and the corresponding question-method `boolean isConst()`. Each setter method has to throw an exception if `isConst()` yields `true` . Disadvantages: Only runtime check, careful programming necessary!

Proxy
This means to provide a surrogate or placeholder for the object [GHJV95, p.207 f]. The interface of the Proxy class contains only methods which do not modify the object.

Common Baseclass
Split your class X into three classes:

```
public class XBase {...}    /* contains private attributes and get-methods.
                               Only the constructor can set the attributes. */
public final class constX extends Xbase {...} /*  inherited API + constX(XBase)
                                                   constructor */
public class X extends Xbase {...} //  inherited API + set-methods
```

`constX` has to be final because otherwise we could derive a subclass with non-const behaviour.

Recommendation
Trying to implement the C++ `const` for class objects in a Java program is a little bit complicated and therefore will probably introduce errors instead of preventing them. If a C++ program is to be rewritten in Java, we recommend just to ignore the `const` keyword for non-primitive data types.

Argument passing surprises

Arguments including pointers can be passed a C++ method *by value or by reference*. Arguments are *always passed by value in Java*. Smalltalk users will not have a problem with this, but people coming from C++ have to realize that parameter passing does not work as they are used to, if objects of class type are involved.

Let's have a look at some examples:

```
// same syntax in Java and C++, but different meaning
void f(Window w) {
  w.setTitle("new Title");
}
```

In C++, a function call `f(aWindow)` does not change the passed object `aWindow`, because a local copy is generated, whose method is called. At the end of the function, the local copy will be destroyed. (Of course, in C++ the above function doesn't seem to make much sense.)

In Java, however, the window (`aWindow`) gets a new title, because a local copy of `aWindow` actually means a copy of a reference, i.e. passing a reference by value. The method `setTitle()` is called via this copy of a reference which refers to the very same object as before the call.

Another example:

```
// Java (wrong example)
void fullFilename(String path, String filename, String result) {
        // Error: caller's result is not modified, only a local copy of result:
  result = path + filename;
}
```

The (copied) reference of `result` is changed, not the object it refers to! Preventing a change of a local copy can be achieved using `final`, but this does not help here. Correct example:

```
// Java (now ok, but different interface)
String fullFilename(String path, String filename) {
   return new String(path + filename);
}
```

The valuable C++ property `const` is also helpful within methods where arguments are passed as a reference to a constant object (`const&`) for performance reasons. Unfortunately, there is no similar support in the Java language. Immutable objects need not be copied; therefore the runtime-expensive creation of objects is reduced in C++ with `const&`. If the caller wants to be sure that a passed object will not be modified, it has to be passed as a clone in Java. This does not, however, prevent the clone from being unnecessarily modified, in contrast to C++.

Examples:

```
// C++ method:
void f(const X& aX) {
    doThisAndThat();
    aX.setAttribut(1);   // statically detected error!
    doEvenMore();
}
```

call:

```
myObject.f(anotherX);
```

```
// Java method:
void f(X aX) {
    doThisAndThat();
    aX.setAttribut(1);   // neither statically nor dynamically detected!
    doEvenMore();
}
```

call:
`myObject.f((X)anotherX.clone()); /* At least the caller can be sure that the object remains unmodified. */`

Recommendation
You can only pass arguments by value in Java. Be sure that the parameter passing mechanism is used correctly. Is there an assignment to a parameter within a method, it is only a assignment to the local copy of the parameter. It has no effect for the caller of the method, so there must be a special reason, which should be documented in a comment – otherwise it is wrong and the parameter should be declared as `final`. Use code inspection to find such cases.

General rules

All the examples above are easily understood by C++ experienced people by following the general rules:
1. Primitive data types like `float`, `char`, `int` etc. are treated equally in C++ and Java.
2. All Java objects of other types (including arrays) are nothing but C++ pointers in disguise.

The second point means that there are no automatic (i.e. execution stack) class objects in Java. What is called a reference in Java, resembles more a C++ pointer than a C++ reference. References in C++ cannot be null or undefined, but pointers can. To show this in more detail, some examples are listed in the following tables:

C++ example	**direct semantic Java equivalent**
`myClass x;`	None, there are no automatic (stack) class objects in Java (only references).
`myClass* p = 0;`	`myClass p = null;`
`p = new myClass;`	`p = new myClass();`
`p->method1()`	`p.method1()`
declaration: `void f(myClass)` call: `f(*p)`	declaration: `void f(myClass)` call: `f((myClass)p.clone())`
declaration: `void g(myClass*)` call: `g(p)`	declaration: `void g(myClass)` call: `g(p)`
declaration: `void g(myClass&)` call: `g(x)`	None, there is no passing by reference in Java. It can be simulated by a more circumstantial mechanism, though. Encapsulation of a reference in a container may be such an indirect semantic equivalent (see below ”Simulating reference parameters in Java ”).
`myClass* const q = new myClass;` `const` pointer, very seldom used in C++. Don’t confuse with pointer to `const`! (`const myClass* t`, see below)	`final myClass q = new myClass();`
`const myClass* q = new myClass;`	None.

Java example	**direct semantic C++ equivalent**
`String s = "hello";`	`const string* s = new string("hello");`
`s = "abc";`	`delete s; s = new string("abc");`
`char c = s.charAt(0);`	`char c = s->at(0);`

Simulating reference parameters in Java

Simulating reference parameters has nothing to do with our intention to show possible problems in program understanding during reviews of syntactically alike C++ structures in Java, but is given for completeness. In the function (already shown above)

```
// Java (wrong example!)
void fullFilename(String path, String filename, String result)
    result = path + filename;
}
```

the problem is: How to get *out* some information *via the parameter list*? There is no direct way to pass arguments by reference, but a little (not very elegant) trick will do: Just wrap the reference in a container, e.g. an array. The function then looks like

```
void fullFilename(String path, String filename, String[] cont) {
        cont[0] = path + filename;
}
```

A possible usage is as follows:

```
String[] container = new String[1];
fullFilename("thePath", "Filename", container);
String result = container[0];        // retrieve reference
System.out.println(result);          // thePathFilename
```

Recommendation
C++ and Java look similar, but are different. Java is often regarded as ”C++ without pointers,” which obviously is not true. Actually, Java is more like ”C++ with dynamic (=heap) class objects only”. If your company switches from C++ to Java, special training of developers is a must. They have to know the semantic differences between these languages.

Lazy clean up

Big advantages of Java are (among others) that memory addresses cannot be taken and misused and that there is an automatic garbage collection. In C++, the destructor of an automatic object is called at the very moment when the object goes out of scope or when `delete` is called for a pointer referring to the object. Normally the destructor is used to release resources held by the object. Example:

```
// C++
{
   Object myObject;           // Constructor acquires resources.
   myObject.doSomeThing();
}  // Destructor is called here automatically. Its task is to release resources.
```

The destructor is also called if an exception is thrown within the block. In Java, the `finalize()` method roughly corresponds to the destructor in C++, but it has to be called explicitly, and it is left to the JVM (Java Virtual Machine) when `finalize()` will be called, even if the JVM is explicitly asked to do so by calling `System.runFinalization()`. If resources (other than the memory of the object itself) are to be released a.s.a.p., the best is to write a special

routine `cleanup()` for that purpose and call it directly before transferring the object to the garbage collector.

```
// Java
{
   myClass myObject = new myClass(); // constructor acquires resources.
   myObject.doSomeThing();

   // recommended add-on:
   myObject.cleanup();          // release resources now!
   myObject = null;             // release object for garbage collector and prevent
                                //      further access (if more code is to follow)
}  // The garbage collector can do the rest, when it wants to!
```

A safer approach is to use exceptions. The `finally`-block will be executed, no matter how the `try`-block exits – normally or with an exception generated in one of the statements.

```
// Java
{
   myClass myObject = null;
   try {
      myObject = new myClass();     // might throw exception
      myObject.doSomeThing();       // might throw exception
   }
   catch(Exception e) {
       // here goes the excepton handling
   }
   finally {
      if(myObject != null) {
         myObject.cleanup();        // release resources
         myObject = null;           // release object
      }
   }
}  // The garbage collector can do the rest, when it wants to!
```

If applicable, `cleanup()` should call `super.cleanup()` as the last action to release resources previously aquired by the superclass.

Recommendation
If the immediate release of resources (other than the memory of the object itself) is important, you have to do it by yourself. The Java Virtual Machine does not guarantee a defined moment for release. Use code inspection to find critical classes and places.

Polymorphic behaviour

All methods with the exception of static methods are virtual in Java. Static methods are class methods (not comparable with class methods in Smalltalk, which are always dynamically bound). This is no problem for those C++ people who write in good style, i.e. they

- never override non-virtual C++ methods and
- always call static methods by a class name, not an object's name.

The second is also our recommendation for Java.

Implications for program analysis, understanding and testing

How can the problems described above be avoided or recognised and solved? The most important precondition is that developers have a good knowledge of Java. The assumption ”Java is a simpler C++” is wrong and leads to erroneous programming.

Besides a thorough training, the following ways may be suitable for finding errors:

- reviews and inspections
- tools for static code analysis
- test methods

There are two cases to be considered for developers with C++ knowledge: new software is to be developed in Java, or software is to be ported from C++ to Java. In the second case, a direct (maybe automated) comparison between the sources is possible. Check-lists for code-inspections should be extended by the following questions:

- Assignment of class objects: Should an alias (1) be created or a copy (2)?
  1) ok
  2) use `clone()`
- Comparison of class objects using ==: Check of identity (1) or equality (2) intended?
  1) ok
  2) use `equals()`
- Does parameter passing conform to the ”General Rules” above?
  If not, rethink and recode.
  Is there a local parameter on the left hand side of an assignment in a method?
  If yes, it has no effect for the caller of the method, so there must be a special reason, which should be documented in a comment – otherwise it is wrong. The compiler will prevent this usage if the parameter is declared as `final`.
- End of scope: Is an immediate release of resources required?
  If yes, use a special self-written method for release.
- Is reproducing the C++ `const` in a Java program really necessary for class objects?
  If not, avoid introducing unnecessary code complexity.
- Are class methods only called by using the class name?
  If not, rethink and recode.

Many of these difficulties could be detected by a static code analyser [K95], which could print a list of possible cases.

We do not recommend special test methods and test cases because often the test engineer will not be able to decide if there is really an error. Creating an alias with an assignment may just be what is intended, and also a comparison of identical objects with different aliases can make sense.

More important is that software developers really know the semantic differences between similar syntactic elements of C++ and Java.

Conclusions

On the apparently ”simple” way from C++ to Java there lie some stumbling blocks. Overlooking them can lead to unforeseen errors. The semantics of Java resembles the reference semantics of Smalltalk, not the value semantics of C++. There are syntactically identical elements in the programming languages C++ and Java, where Java uses reference semantics, but C++ employs value semantics. Developers have to know the semantic differences between these similar syntactic elements. Also the destruction of objects behaves differently in both languages, and there is no parameter passing by reference in Java. The phrase ”passing an object by reference” is surely acceptable in a colloquial language usage between Java developers, but the actual, precise meaning is ”passing a reference (referring to that object) by value”. Developers new to Java have to know this too.

This article shows problems which can arise from these misunderstandings, and gives recommendations and a check-list as an aid to detect possible errors.

Bibliography

[A98] Sherman R. Alpert: Primitive Types Considered Harmful, Java REPORT 3, No. 11 (Nov. 1998), pp 49-65

[B00] Steve Ball: Effective cloning, Java REPORT 5, No. 1 (Jan. 2000), pp 60-67

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley 1995

[GJS96] James Gosling, Bill Joy, Guy Steele: The Java Language Specification, Addison-Wesley 1996

[ISO98] Programming Language C++, ISO/IEC Industrial Standard 14882:1998 (available at www.ansi.org)

[K95] Edward Kit: Software Testing in the real world. Appendix G – Specific tools and tool selection, Addison-Wesley 1995

[McC00] Steve McConnel: The Best Influences on Software Engineering, IEEE Software Jan./Feb. 2000, pp 10-17

[SU99] Martin Schedlbauer, David Udin: How to successfully Migrate to Java, Java REPORT 4, No. 1 (Jan. 1999), pp 61-66

QW2000 Paper 7A1

Dr. Yingxu Wang
(Center for Software Engineering)

A Practical New Approach to COTS Testing (7A1)

BACK TO QW2000 PROGRAM

Key Points

- How to implement built-in tests in COTS.
- How to re-use built-in tests in COTS.
- How to test COTS at run-time.

Presentation Abstract

This paper presents a new approach to commercial off-the-shelf (COTS) component test and a practical technology for design and implementing test-re-usable COTS. Current technologies for COTS test are mainly based on object-oriented (OO) techniques. A new method of Built-In Test (BIT) components for self-testable and test-re-usable software has been developed and trialed in pilot projects.

A BIT-based COTS is a new kind of software component wherein tests are explicitly described in the component source code as special functions. BIT technology is considered to be a significant extension of OO technology to self-testable and test-re-usable COTS in component-based software engineering.

About the Author

Dr. Yingxu Wang is Professor of Software Engineering and project manager with the Center for Software Engineering at IVF, Moindal, Sweden. He was a visiting professor in the Computing Laboratory at Oxford University during 1995. He was awarded a PhD in software engineering by The Nottingham Trent University / Southampton Institute, UK, where he holds an academic title of Visiting Professor.

Dr. Wang is a member of IEEE TCSE/SESE, ACM, ISO/IEC JTC1/SC7, and chairman of the Computer Chapter of the IEEE in Sweden. He has accomplished a number of ED, Swedish and industry funded research projects as coordinator local manager and/or principle investigator. He is the lead author of a recent book, *Software Engineering Processes: Principles and Applications* and has published over 100 papers in software engineering. He has served a session chairman, co-editor, and program committee member for several international conferences and journals. He has won a dozen

research achievement and teaching awards in the last 20 years.

BACK TO QW2000 PROGRAM

The 13th International Software / Internet Quality Week (QW2000)

A Practical New Approach to COTS Testing

Yingxu Wang, PhD

Centre for Software Engineering, IVF
Argongatan 30
S-431 53, Molndal, Gothenburg, Sweden
Tel: +46 31 706 6174, Fax: +46 31 27 6130
yingxu.wang@acm.org

Agenda

1. Introduction
2. COTS test: the new challenges
3. Built-in-test-based COTS
4. Test-reusable COTS
5. Run-time testable COTS
6. Conclusions

COTS and BITs

COTS: Commercial off-the-shelf software components

BIT: Built-in Test

- Special test functions that are explicitly described and embedded in COTS source code for self-testing the component.

Challenges to COTS test

- Low testability for end-users
- Low maintainability for end-users
- No support for run-time testing
- Separated software code and test cases

Focuses

- How to implement built-in tests in COTS?
- How to reuse built-in tests in COTS?
- How to test COTS at run-time?

A COTS prototype with built-in tests

```
BIT_COTS  ::=   interface
                || implementation

            ::= { Data declaration;
                  Constructor declaration;
                  Destructor declaration;
                  Function declarations;
                  Tests (BITs) declarations }

                || { Constructor;
                     Destructor;
                     Functions;
                     BIT-TestCases }
```

Mechanisms of BIT-based COTS

- **Normal mode**
  - With same behaviors as that of the conventional COTS

    BIT_COTS :: FunctionI;

- **Test mode**
  - BITs can be activated:

    BIT_COTS :: TestCase1;

    BIT_COTS :: TestCase2;

    .....

    BIT_COTS :: TestCaseN;

A sample COTS with BIT

```
Class BITsBinarySearch_COTS {
//////////////////////////////////////////////////////////////////////////
//  Interface
//////////////////////////////////////////////////////////////////////////
// Member functions
BITsBinarySearch();                 // The constructor
~BITsBinarySearch();                // The destructor
int BinarySearch (int Key;  int DataSet[10]);
                                    // The conventional object
void BIT1();                        // The built-in-test

//////////////////////////////////////////////////////////////////////////
//  Implementation
//////////////////////////////////////////////////////////////////////////
// ================================
//  Part 1: The conventional function code
// ================================
```

A sample COTS with BIT (Cont'd)

```
int BinarySearch (int Key,  int DataSet[10])
{
// The conventional object
// Assume: DataSet is ordered
//              LastElement -FirstElement >=0
//              and FirstElement >=0
// Input:      Key to be found in the DataSet
// Output:   TestElemIndex
Private:
      int bott, top, i;
      int found;
found = false;
Bott = 1;
Top = ArraySize (DataSet);   // The last element in DataSet
while (bott <= top) && (not found)
        {
         i = floor ((bott + top)/2));
           if DataSet[i] == Key
             Found = true;
             else   if   DataSet[i] < Key
                         Bott = i +1
                         else  Top = i +1;
        }
if found == true
   return i;            // The index of the element
   else return 0;       // An indicator of not existence
}
```


A sample COTS with BIT (Cont'd)

```
// ===================================
//  Part 2: The BITs
// ===================================

// BIT case 1
void BIT1()
{
// BIT case 1: Array size of 1, key in array
private:
   int DataSet[1] = {16};
   int Key  =  16;
   int StdElemIndex = 1;
   int TestElemIndex;
   char TestResult1[5];
// Test implementation
   TestElemIndex = BinarySearch (Key, DataSet);
// Test analysis
   cout  << "StdElemIndex1 =  "    <<  StdElemIndex    <<  "\n";
   cout  << "TestElemIndex1 =  "   <<  TestElemIndex   <<  "\n";
   if  TestElemIndex == StdElemIndex
      TestResult1 = "OK";
      else  TestResult1 = "FALSE";
   cout << "TestResult1: "  <<  TestResult1 <<  "\n";
}
}}
```

Behaviors of the BIT-based COTS

Normal mode

BITsBinarySearch_COTS::
BinarySearch(int Key, int DataSet[10]);

Test mode

BITsBinarySearch_COTS::BIT1();

Testing results for this COTS by the BIT:

StdElemIndex1 = 1
TestElemIndex1 = 1
TestResult1 = OK

Reuse of BITs in COTS

Application		Structure: Interface	Structure: Implementation		Behaviour
Normal mode	→	Function interface	Member functions	→	Normal functions
Test mode	→	Test interface	Built-in test cases	→	Tests and results

BIT reuse in COTS: An example

```
class DatabaseQuery: public BITsBinarySearch_COTS
{
////////////////////////////////////////////////////////////////////////////
//  Part 1: The inherited conventional functions
////////////////////////////////////////////////////////////////////////////

int DatabaseQueryBinarySearch (int Key,  int DataSet[10]) :
    BITsBinarySearch_COTS::BinarySearch(int Key;  int DataSet[10]);

////////////////////////////////////////////////////////////////////////////
//  Part 2: The inherited BIT functions
////////////////////////////////////////////////////////////////////////////
void BIT1() : BITsBinarySearch_COTS::BIT1();
```


BIT reuse in COTS: An example (Cont'd)

```
////////////////////////////////////////////////////////////////////////////
//  Part 3: The newly developed BITs
////////////////////////////////////////////////////////////////////////////

// BIT case 2
void BIT2()
{
// BIT case 2: Even array size, key 1st element in array
Private:
   int DataSet [6] = {16,18,21,23,29,33};
   int Key =  16;
   int StdElemIndex = 1;
   int TestElemIndex;
   char TestResult4 [5];
// Test implementation
   TestElemIndex = BinarySearch (Key, DataSet);
// Test analysis
   cout  << "StdElemIndex4 =  "   <<  StdElemIndex   <<  "\n";
   cout  << "TestElemIndex4 =  "   <<  TestElemIndex   <<  "\n";
  if  TestElemIndex == StdElemIndex
     TestResult2 = "OK";
     else  TestResult2 = "FALSE";
   cout << "TestResult2: "  <<  TestResult4 <<  "\n";
}
}}
```

Invocation of the BITs in COTS

- **Invoke the old BIT in COTS: BITsBinarySearch_COTS**

  DatabaseQuery::BIT1();
  // Equivalent to BITsBinarySearch_COTS::BIT1

- **Invoke the new BIT:**

  DatabaseQuery::BIT2();
  // The new BIT only in COTS: DatabaseQuery

COTS fault detection/diagnosis at run-time

System structure			Fault cause allocation				
			Hardware	Decay	Random	Environment	Configuration
System			Y	Y	Y	Y	Y
Subsystem$_1$			Y	Y	Y	Y	Y
	Class$_1$		Y		Y		
	Object$_{11}$			Y			
		Function$_{111}$					
		Function$_{112}$		Y			
		.....					
		Function$_{11n}$					
	Object$_{12}$		Y		Y	Y	
		Function$_{121}$				Y	
		Function$_{122}$	Y				
		.....					
		Function$_{12r}$	Y		Y		
		.....					
	Object$_{1n}$		Y		Y		Y
		Function$_{1n1}$	Y		Y		
		Function$_{1n2}$	Y				Y
		.....					
		Function$_{1ns}$			Y		
		.....					
	Class 2		Y		Y		
	Object$_{21}$						
	Object$_{22}$		Y				
	.....						
	Object$_{2m}$				Y		
	.....						
Subsystem$_2$							
.....							
Subsystem$_k$							

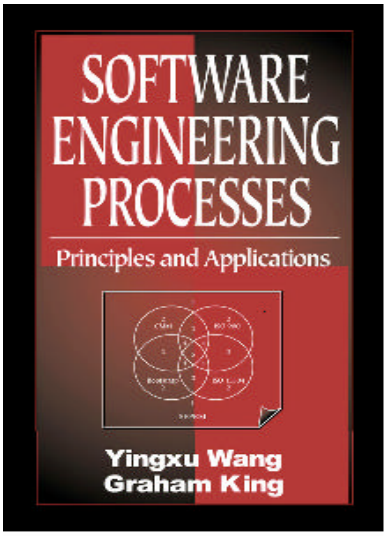
COTS run-time fault handling

Exception diagnosis result	Exception handling strategy						
	Alarm and report	System reset	Reload objects	Reload data	Re-configuration	Switch to stand-by system	Replace hardware
System	x	x	x	x	x	x	x
Subsystem $_1$	x	x	x	x	x	x	x
Class $_1$	x	x	x	x	x		x
Object $_{11}$		x					x
Function $_{112}$		x					x
Object $_{12}$	x		x				
Function $_{121}$	x		x				
Function $_{122}$	x		x				
Function $_{12c}$	x		x			x	
Object $_{1n}$	x			x	x		
Function $_{1n1}$	x			x			
Function $_{1n2}$	x					x	
Function $_{1ms}$	x				x	x	
Class $_2$	x		x				
Object $_{22}$	x		x				
Object $_{2m}$	x		x				

Conclusions

- New type of tests for COTS
- New philosophy for COTS test and test reuse
- Extended software reuse from code to tests
- Enable self-testable COTS
- Enable test-reusable COTS
- Enable run-time testable COTS
- Find useful in a wide range of applications

Related Work



A Unified Framework and a Rigorous Approach to Software Engineering Processes

(Suitable as a textbook)

CRC Press, USA

ISBN: 0-8493-2366-5

http://www.crcpress.com/index.htm?catalog/2366

About the Author

Yingxu Wang is Professor of Computer Science, and project manager with the Center for Software Engineering at IVF, Gothenburg, Sweden. He was a visiting professor in Computing Laboratory at Oxford University during 1995. He was awarded a PhD in software engineering by The Nottingham Trent University / Southampton Institute, UK.

Dr. Wang is a member of IEEE TCSE/SESC, ACM, and ISO/IEC JTC1/SC7, and is Chairman of the Computer Chapter of IEEE (Sweden). He is the coordinator of the National Trial Centre for ISO 15504 (SPICE) of Sweden. He has accomplished a number of European, Swedish, and industry-funded research projects as manager and/or principal investigator, and has published over 100 papers in software engineering. He has won a dozen research achievement and academic teaching awards in the last 20 years.

A Practical New Approach to COTS Testing

Yingxu Wang

Ph.D., MIEEE-TCSE/SESC, MACM and MISO/IEC JTC1/SC7
Centre for Software Engineering, IVF
Argongatan 30, S-431 53, Gothenburg, Sweden
Tel: +46 31 706 6174, Fax: +46 31 27 6130
Email: yingxu.wang@ivf.se or yingxu.wang@acm.org

Abstract

This paper presents a new approach to commercial off-the-shelf (COTS) software component test and a practical technology for design and implementing test-reusable COTS. Conventional technologies for COTS test are mainly based on object-oriented (OO) testing techniques. A new method of Built-In Test (BIT) for self-testable and test-reusable COTS is developed in this paper.

A BIT-based COTS is a new kind of software component wherein tests are explicitly described in the component source code as special functions. BIT technology is considered to be a significant extension of OO technology to self-testable, test-reusable, and run-time testable COTS in component-based software engineering.

Keywords: Software engineering, component, COTS, test, built-in test, test reuse, run-time testing, real-time software

1. Introduction

Component-based software engineering is a new approach to programming and software development. Software components are adopted in order to improve software development and maintenance efficiency and quality and also to increase reuse rate of existing software in multiple applications. Viewing software architectures as being composed of components is helpful for enabling software development, test, and maintenance to be carried out at a higher level than that of language statements.

Component-based software engineering studies methods and techniques for building, acquiring, maintaining, and managing software systems consisting of commercial off-the-shelf (COTS) and in-house software components. The current technologies for COTS component development is mainly based on object-oriented (OO) techniques. A new method of Built-In Test (BIT) components for self-testable and test-reusable COTS is presented in this paper. A BIT-based COTS is a new kind of software component where tests are explicitly described in the component source code as special functions. BIT technologies [1-5] are considered to be a significant extension of OO technology to self-testable and test-reusable software in component-based software engineering.

This paper presents a new type of COTS incorporating BITs. The BIT components can be embedded in any conventional COTS for enabling test reuse as well as code reuse. The BIT component technologies are an extension of OO methods, and have found a wide range of applications in component-based software engineering.

Adoption and wide availability of COTS have been viewed as the key precondition for enabling component-based software engineering. However, current COTS technologies are concentrating on code reuse. The following problems inherited in conventional COTS technologies have been identified:

- Low testability for end-users
- Low maintainability for end-users
- No support for run-time testing
- Separated software code and test cases

Being oriented to the current problems and challenges identified above, this paper presents new solutions and techniques for COTS testing on the basis of a recent European pilot project:

- A new technology for implementing BIT-based COTS
- A new approach to COTS test
- An extension of OO technology from code reuse to test reuse in COTS development
- A new approach to enabling run-time COTS test

The BIT-based COTS possesses a set of novelty in component-based software engineering. A wide range of applications of COTS with BITs has been identified, *inter alia*, as follows: BIT-based COTS test, COTS test reuse, BIT-based COTS maintenance, run-time testable COTS with BITs, BIT-based COTS for safety critical systems, and real-time system fault-tolerance with BIT-based COTS.

2. COTS Test: New Challenges and New Solutions

Along with the development of the Internet, and inspired by the hardware engineering approaches, a new approach to software engineering, component-based software engineering, has been emerged and been widely accepted in the software industry. This approach is based on the 'plug-in' and/or 'add-on' software framework structure, and the broad availability of COTS components. This section analyses problems and challenges for COTS testing, and presents new solutions and new type of COTS for enabling test reuse as well as code reuse.

2.1 Problems Identified in COST Test

The foundations of component-based software engineering are based on technologies of OO, reuse, COTS, middleware, patterns and frameworks. Adoption and wide availability of COTS have been viewed as the key precondition for enabling component-based software engineering. However, current technologies of COTS are focused on code reuse. The following problems inherited in conventional COTS technologies have been identified as analysed below.

- Low testability for end-users

Because of much closer encapsulation and distribution with only executable code, end-users of COTS components have found it was very difficult to test a purchased and adopted COTS component. As a result, the quality and reliability of software based on COTS and in-house components are limited.

- Low maintainability for end-users

For the same reasons described above, the maintainability of COTS and in-house software components has been tied up with the original component developers. As a result the maintainability of a purchased or adopted COTS component has been found to be lower than that of conventional technologies.

- No support for run-time testing

Run-time testing comprises special test requirements for COTS and systems. In component-based software engineering, as well as in conventional programming, run-time test is very much in demand because almost all crucial software faults, such as code corruption, hardware platform faults, random faults caused by external interferences, dynamic memory allocation faults, etc, can only be detected and tested at run-time. However, run-time test of COTS has been found to be extremely hard to implement compared with conventional static test of software components.

- Separated software code and test cases

Conventionally, software tests are regarded as extra entities other than functional code. Therefore, for a COTS, the source code is separated from its tests, and the tests are only available for the original developers or vendors, rather than the end-users as required in component-based software engineering. This convention has been found cost-intensive and energy-wasting in programming, especially in component-based software development, because all the tests have to be regenerated by the end-users in the phases of system integration and maintenance.

To avoid the above problem, this paper provides a new approach that incorporates software source code and tests within a single piece of software documentation, and within the same executable code. This is also considered to be a new evolution of programming style to meet the requirements for component-based software engineering.

2.2 New Solutions for COTS Test

Being oriented to solve the current problems identified in COTS test as analysed in Section 2.1, this paper develops a built-in-test (BIT) method that enables tests be embedded and reused as that of code in COTS and in-house software components. The BIT-based COTS possesses a set of novelty, as follows, in component-based software engineering.

- A new technology for implementing BIT-based COTS

As analysed in Section 2.1, software tests in conventional programming technologies were regarded as extra entities separated from code. Whilst, interestingly, in the author's research in component-based software engineering, it is found that tests should be treated as software

components too. This means that built-in tests can be embedded in COTS and systems as special code. The explicitly programmed BITs in a COTS component are special functions for test and maintenance of the component. In this way the BIT components possess the same syntax and semantics as those of functional code. Therefore, the BIT components can be seen as a combination of conventional code and its test cases in a unified COTS component. This provides a novel type of software encapsulation and documentation.

- An extension of OO technology from code reuse to test reuse

By using BITs, tests can be reused for the first time just as is the code for COTS. As a result, the reusability of OO technology can be extended from code to test. When a BIT component is inherited, both functions and tests of the component are reusable. This adds value significantly to the emerging new technologies of component-based software engineering. All COTS components can benefit from this advantage. As a result, reuse software engineers and end-users of COTS need no longer to be worried about the testing and maintaining of a component being as a black box.

- A new approach to OO software test in component-based software engineering

As analysed in Section 2.1, COTS test techniques were mainly the same as those of non-OO software. These tests are hardly ever available for end-users and maintainers. This is one of the major barriers for component-based software engineering. This paper provides the BIT and reusable tests for COTS components. It is a significant progress toward component-based software engineering.

- A new approach to enabling run-time software test

As analysed before, run-time tests of COTS are a hard problem and in high demand for dealing with dynamic faults such as code corruption, hardware platform faults, random faults caused by external interferences, and dynamic memory allocation faults, etc.

A new approach is provided that enables the BIT-based COTS components be self testable and the dynamic faults be detected at run-time, whenever the BITs are periodically executed and/or manually triggered. Run-time testable COTS by using BITs is considered to be another significant technological advance for improving software reliability and fault-tolerant capability in component-based software engineering.

- A new style of programming: Software = functional code + testing code

Conventionally, software development technologies were focused on implementation of normal functionality. Therefore, what the end-users of COTS have purchased are purely functional code packaged in a component as a black box. However, in the later phase of integration and maintenance, COTS users will realise that they need more than just the executable code.

From this scenario it can be deduced that COTS users in component-based software engineering need a new style of programming documentation – a combinatorial source code and the built-in test code. By adopting the BIT technologies, the combinatorial software code and tests within the same COTS can be implemented in a practical way.

3. A Novel Method for BIT-Based COTS Test

This section describes new methodologies and techniques of BITs and BIT-based COTS. It is noteworthy that a good new technology is not necessarily complicated. The BIT technologies can be implemented much easier and be used in a wide range of applications.

3.1 The Architecture of a BIT Component

The fundamental attributes that can be commonly identified in OO-based COTS technologies are encapsulation, inheritance, reusability and polymorphism. A conventional COTS consists of two structural parts: an interface and an implementation. The 'interface' of a COTS is the only means of external access to the functions packaged in the component; The 'implementation' of a COTS is the description of codes for all internal functions. A COTS component is reusable because of its natural encapsulation and inheritability.

There are two basic findings in this paper on software component architectures: a) the tests of COTS components are code too; and b) tests and code can be integrated into a unified encapsulation of software components. Based on these principles, a BIT-based COTS component can be described by using CSP-like notations as shown in Figure 1.

```
BIT_COTS ::=   interface
           || implementation

         ::= { Data declaration;
                 Constructor declaration;
               Destructor declaration;
               Function declarations;
               Tests (BITs) declarations; }
           || { Constructor;
               Destructor;
               Functions;
               BIT-TestCases; }
```

Figure 1. The architecture of a BIT-based COTS

In Figure 1, the built-in tests of a COTS component are declared in the interface and are implemented in the component's body. In this way, the BITs can be naturally inherited and reused in the same way as that of code in the COTS. The BITs adopt the same syntax and semantics as those of the conventional functions in a COTS component. Thus BITs can be well fit into any OO components and may be supported by any OO languages.

A BIT-based COTS has the same behaviours as that of the conventional COTS when normal functions of the component are executed. Whilst if the *i*th BIT is called as a special built-in function of the COTS, e.g.:

```
BIT-COTS :: TestCase1;
BIT-COTS :: TestCase2;
 ......
BIT-COTS :: TestCaseI;
 ......
BIT-COTS :: TestCaseN;
```

the component can be automatically tested by the BITs in static environment and at run-time. This provides a new capability to enable self-testable, test-reusable, run-time testable, and end-user-testable COTS components to be developed and implemented at the same technical platform of conventional OO components.

Inheritance and reuse of the BIT components in development and maintenance of BIT-based COTS are useful in two ways: a) to activate the BIT components for self-test, maintenance and/or fault diagnosis; and b) to derive a new COTS that inherits the existing BITs in the parent components. Applying the BIT technology in COTS, what the programmer, tester, maintainer, and end-user inherited, is instant and self testable. Assuming that the existing software systems are reengineered using the BIT-based COTS technologies, the future software production, testing and maintenance will benefit strongly from the reuse of the BITs in the COTS components.

3.2 A Sample COTS with BITs

In this subsection, methods for design and implementation of BIT-based COTS components are developed that treats tests as software components too. The basic approach is embedded BITs into the conventional COTS as special code. The explicitly programmed BITs in a COTS are special functions for test and maintenance of the component. Therefore, the BIT components can be seen as a combination of conventional code and its test cases in a unified COTS encapsulation and a new type of software documentation.

A case study on implementing BIT-based COTS is provided in Figure 2. A typical example, a binary search COTS, is taken to show how the BIT method is used to develop test built-in COTS. In Figure 2, the BIT-based COTS of binary search is implemented in two parts: the conventional functions and the BIT functions.

For the binary search function listed in Part 1 of Figure 2, a set of test cases may be generated by using equivalency partitioning technique or others. In Figure 2, one of the test cases is built-in to show the method of BITs. It is significant that the BIT method can incorporate any test cases generated by the black-box (functional) and/or white-box (structural) testing methods as the BITs.

```
Class BITsBinarySearch_COTS {

//////////////////////////////////////////////////////////////////////////////
//  Interface
//////////////////////////////////////////////////////////////////////////////

// Member functions
BITsBinarySearch();                                  // The constructor
~BITsBinarySearch();                                 // The destructor
int BinarySearch (int Key;  int DataSet[10]);        // The conventional object
void BIT1();                                         // The built-in-test

//////////////////////////////////////////////////////////////////////////////
//  Implementation
//////////////////////////////////////////////////////////////////////////////

// ================================
//  Part 1: The conventional COTS code
// ================================
int BinarySearch (int Key,  int DataSet[10])
{
// The conventional COTS
// Assume: DataSet is ordered
//              LastElement -FirstElement >=0
//              and FirstElement >=0
// Input:      Key to be found in the DataSet
// Output:   TestElemIndex

Private:
int bott, top, i;
int found;

found = false;
Bott = 1;
Top = ArraySize (DataSet);   // The last element in DataSet
while (bott <= top) && (not found)
        {
          i = floor ((bott + top)/2));
          if DataSet[i] == Key
             found = true;
             else  if  DataSet[i] < Key
                        bott = i +1
                        else  Top = i +1;
        }
if  found == true
    return i;              // The index of the element
    else return 0;      // An indicator of not existence
}

// ==================================
//  Part 2: The BITs
// ==================================

// BIT case 1
void BIT1()
{
// BIT case 1: Test for odd array size, and key not in array
private:
int DataSet[7] = {16,18,21,23,29,33,38};
int Key =  25;
int StdElemIndex = 0;
int TestElemIndex;

char TestResult1 [5];
// Test implementation
TestElemIndex = BinarySearch (Key, DataSet);
// Test analysis
cout  << "StdElemIndex1 =  "    <<  StdElemIndex    <<  "\n";
cout  << "TestElemIndex1 =  "   <<  TestElemIndex   <<  "\n";
if  TestElemIndex == StdElemIndex
    TestResult1 = "OK";
    else  TestResult1 = "FALSE";
cout << "TestResult1: " <<  TestResult1  <<  "\n";
}
}
```

Figure 2. A BIT-based COTS of binary search

4. Test-Reusable COTS with BITs

The methods for developing BIT-based COTS have been developed in Section 3. Corresponding to the design of the BIT-based COTS, reuse of the BITs can be implemented systematically.

4.1 BIT Reuse in COTS

The tests of conventional COTS are hardly ever available for maintainers and end-users. This is one of the major barriers for component-based software engineering. This subsection develops practical technologies for built-in test reuse in a COTS, including inherence and invocation technologies for BITs.

Functions of a BIT-based COTS can be categorised into normal mode and test mode as shown in Figure 3. The former is applied for code reuse and the latter for test reuse.

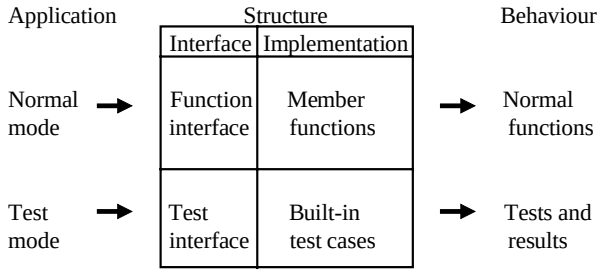


Figure 3. Reuse modes of a test-built-in COTS

In the normal mode, a BIT-based COTS component has the same functions as that of conventional COTS. Its static and dynamic behaviours are the same as those of the conventional ones. The application-specific member functions can be called by: ClassName::FunctionName, such as:

```
BITsBinarySearch_COTS::
BinarySearch(int Key,  int DataSet[10])
```

and the BITs are stand-by and without any effect to run-time behaviours.

In the test mode, the test-built-in COTS can be activated by calling the test cases as member functions: ClassName::TestCaseI, such as:

```
BITsBinarySearch_COTS::BIT1();
```

Each TestCaseI consists of a BIT driver and test cases for the specific object. Test results can be automatically reported by the BIT driver.

A BIT-based COTS has testing mechanisms ready as well as functional code. This enables end-users of an applied test-built-in COTS to call and reuse all BITs as member functions in the test mode.

4.2 A Case Study on BITs Reuse

The approach to reuse BIT-based COTS components in software testing and maintenance is shown in Figure 4. For instance, when a new COTS, DatabaseQuery, is needed, the BIT components (Part 2, Figure 2) developed in the BITsBinarySearch_COTS can be inherited and reused directly as that of the conventional member functions (Part 2, Figure 4). Also, additional BITs can be incorporated into the new COTS as shown in Part 3 of Figure 4.

```
Class DatabaseQuery: public BITsBinarySearch_COTS
{
////////////////////////////////////////////////////////////////////////////
//  Part 1: The inherited conventional functions
////////////////////////////////////////////////////////////////////////////

int DatabaseQueryBinarySearch (int Key,  int DataSet[10]) :
     BITsBinarySearch_COTS::BinarySearch(int Key;  int DataSet[10]);
…

////////////////////////////////////////////////////////////////////////////
//  Part 2: The inherited BIT functions
////////////////////////////////////////////////////////////////////////////
void BIT1() : BITsBinarySearch_COTS::BIT1();
…

////////////////////////////////////////////////////////////////////////////
//  Part 3: The newly developed BITs
////////////////////////////////////////////////////////////////////////////
//
// ---------------------------------------------------------
// BIT case 2
// ----------------------------------------------------------------
void BIT2()
{
//  BIT case 2: Test for even array size, key in array, and key is not first or last
private:
    int DataSet[6] = {16,18,21,23,29,33};
    int Key =  23;
    int StdElemIndex = 4;
    int TestElemIndex;
    char TestResult2 [5];
// Test implementation
    TestElemIndex = BinarySearch (Key, DataSet);
// Test analysis
    cout  << "StdElemIndex2 =  "    <<  StdElemIndex    <<  "\n";
    cout  << "TestElemIndex2 =  "   <<  TestElemIndex   <<  "\n";
    if  TestElemIndex == StdElemIndex
        TestResult2 = "OK";
        else  TestResult2 = "FALSE";
    cout << "TestResult2: "  <<  TestResult2  <<  "\n";
}
}
```

Figure 4. Inheritability and reusability of BITs in BIT-based COTS

In the new object DatabaseQuery as listed in Figure 4, the existing BITs developed in the BITsBinarySearch_COTS object can be activated by calling:

```
DatabaseQuery::BIT1(); // equivalent to BITsBinarySearch_COTS::BIT1
```

and the new BITs supplemented in the DatabaseQuery object can be activated in the same way:

```
DatabaseQuery::BIT2(); // new BITs only in class DatabaseQuery
```

It is interesting to note that in the BIT approach, software tests themselves are software too. As shown in Figures 2 and 4, the effort for implementing BITs in the COTS "BITsBinarySearch_COTS" has been repaid by the ideal inheritability and reusability of tests in the case shown in Figure 4 and all the subsequent reuse of the BITs in any future applications. Assuming that the existing COTS components and systems can be reengineered using the BITs method, the future software production will benefit strongly from the reuse of BITs in new COTS development, testing and maintenance.

In the BIT approach, what the developer, end-user and maintainer of COTS inherit, is instant and self testable. It is a kind of well engineered software with complete design messages and high testability, reusability, and reliability.

5. Run-Time Testable COTS with BITs

With the BITs incorporating in a conventional COTS, faults of the COTS can be detected, diagnosed, and handled at run-time. A generic example is provided in this section for demonstrating how dynamic faults of COTS may be handled by BITs at run-time.

In case a fault of a COTS is detected, source and type of the fault can be diagnosed and allocated by the BITs deployed in the COTS as shown in Table 1. By this approach, detailed causes and locations of faults can be allocated by the corresponding BITs at run-time.

Table 1. A sample COTS fault detection and diagnosis report

System structure			Fault cause allocation				
			Hardware	Decay	Random	Environment	Configuration
System			Y	Y	Y	Y	Y
$Subsystem_1$			Y	Y	Y	Y	Y
	$Class_1$		Y		Y		
	$Object_{11}$			Y			
		$Function_{111}$					
		$Function_{112}$		Y			
		.....					
		$Function_{11p}$					
	$Object_{12}$		Y		Y	Y	
		$Function_{121}$				Y	
		$Function_{122}$	Y				
		.....					
		$Function_{12r}$	Y		Y		
		.....					
	$Object_{1n}$		Y		Y		Y
		$Function_{1n1}$	Y		Y		
		$Function_{1n2}$	Y				Y
		.....					
		$Function_{1ns}$			Y		
		.....					
	Class 2		Y		Y		
	$Object_{21}$						
	$Object_{22}$		Y				
	.....						
	$Object_{2m}$				Y		
	.....						
$Subsystem_2$							
.....							
$Subsystem_k$							

The diagnosis results in Table 1 show the accurate sources and reasons of faults occurred in a COTS. Observing Table 1 it can be seen that one subsystem, two classes, five objects and seven functions have been allocated for a specific or hybrid fault(s).

After the detection and allocation of any faults in a COTS at run-time, a software system can invoke an appropriate fault handling subroutine according to a pre-designed fault processing strategy. Typical fault handling strategies can be measures such as to alarm, report, reset system, reload objects, reload data, reconfiguration, switch to stand-by system, replace hardware, etc. [6]. A decision table of COTS fault handling for a real-time software system, for instance, is given in Table 2.

Table 2. A sample COTS fault handling decision table

Fault diagnosis result	**Fault handling strategy**						
	Alarm and report	System reset	Reload objects	Reload data	Re-configuration	Switch to stand-by system	Replace hardware
System	x	x	x	x	x	x	x
$Subsystem_1$	x	x	x	x	x	x	x
$Class_1$	x	x	x	x	x		x
$Object_{11}$		x					x
$Function_{112}$		x					x
$Object_{12}$	x		x				
$Function_{121}$	x		x				
$Function_{122}$	x		x				
$Function_{12r}$	x		x			x	
$Object_{1n}$	x			x	x		
$Function_{1n1}$	x			x			
$Function_{1n2}$	x					x	
$Function_{1ms}$	x				x	x	
$Class_2$	x		x				
$Object_{22}$	x		x				
$Object_{2m}$	x		x				

Based on the decision table, automatic or manual system fix can be carried out for a real-time system, according to the pre-designed fault handling strategies for the COTS. This example shows that the BIT-based COTS enables real-time detection, diagnosis and handling of software faults at COTS and system level. BIT provides a systematic approach to improve COTS and software system reliability, maintainability, test-reusability, and run-time testability.

6. Conclusions

This paper has developed a new type of COTS architecture incorporating built-in tests. BIT-based COTS components have enabled tests be reused at both debugging phase and run-time. The BIT method has not only extended inheritability and reusability of conventional COTS from code to tests, but also significantly improved COTS run-time testability.

The BIT-based COTS technologies provide a number of significant advantages for programmers and system designers in component-based software engineering, such as:

- Self-testable COTS and in-house components
- Test-reusable COTS and in-house components
- Run-time testable COTS and in-house components
- End-user testable COTS and in-house components
- Easy maintainable COTS and in-house components
- Built-in run-time fault handling capability for COTS and in-house components
- Higher quality and more reliable COTS and in-house components

The BIT-based COTS technologies have found a wide range of applications in the software industry. These include, *inter alia*: BIT-based COTS test, COTS test reuse, BIT-based COTS maintenance, run-time testable COTS, COTS for safety critical systems, and real-time system fault-tolerance with BIT-based COTS.

Acknowledgements

This work has been partially supported by the EC SPIRE and Swedish PALBUS projects. The author would like to acknowledge the funding organisations for their support and the software organisations for their participation in the pilot projects.

References

[1] Binder, R.V. [1994], Design for Testability in Object-Oriented Systems, *Communications of the ACM,* Vol. 37, No. 9, Sept., pp. 87-101.

[2] Wang Y., King, G., Fayad, M., Patel, D., Court, I., Staples, G., and Ross, M. (2000), On Built-in Tests Reuse in Object-Oriented Framework Design, *ACM Journal on Computing Surveys,* Vol.32, No.1, March.

[3] Wang Y., Wickberg, H. and King, G. (1999), A Method for Built-in Tests in Component-based Software Maintenance, *Proceedings of 3rd IEEE International Conference on Software Maintenance and Reengineering (IEEE CSMR'99),* IEEE CS Press, Amsterdam, March, pp.186-189.

[4] Jorgensen, P.C. and Erickson, C. [1994], Object-Oriented Integration Testing, *Communications of the ACM,* Vol. 37, No. 9, Sept., pp. 30-38.

[5] Wang, Y., Trujillo, J. and Palomar, M. (1997), On a Metric of Software Testability, *Journal of the Spanish Computer Society (Novatics)*, Vol.125, Jan/Feb Issue, pp.10-13.

[6] Wang, Y., King, G., Patel, D., Patel, S. and Dorling, A. (1999), On Coping with Software Dynamic Inconsistency at Real-Time by the Built-in Tests, Special Volume on Real-Time Software Engineering, *International Journal of Annals of Software Engineering,* Baltzer Science Publishers, Oxford, Vol.7, pp.1-14.

QW2000 Paper 7W1

Ms. Jeanette Folkes & Mr. Bert Lamar (Ogilvy Interactive)

The Challenges of Web Testing

BACK TO QW2000 PROGRAM

Key Points

- Challenges of managing WebSites
- Challenges of testing WebSites
- QA methodologies that work on WebSites

Presentation Abstract

The recent proliferation of interactive web site development in virtually every industry has created challenges for Quality Assurance Test departments. These challenges demand innovative solutions. This presentation will focus on those challenges and hopefully offer some insight for those new to web testing.

This will be a joint presentation discussing the challenges of web testing from a management and a production standpoint. Ms. Jeanette Folkes and Mr. Bert Lamar have worked together for three years - a year and a half on client server applications and most recently testing for the web.

Ms. Folkes will address the challenges of web testing from the management perspective. She will outline solutions for developing a company wide test methodology, recruiting and training a skilled test team, and scheduling adequate time for testing. She will discuss which areas have been the most and the least receptive to integrating quality assurance.

Mr. Lamar will address the challenges of testing Interactive web sites from a production standpoint. He will offer solutions for rapid test planning and test case execution, and will explore the limitations of automated testing in an interactive environment.

About the Authors

Jeanette Folkes - Jeanette is the Quality Assurance Manager at OgilvyInteractive, a division of OgilvyOne, a major advertising firm. Her experience has stemmed from supporting and managing a helpdesk, writing documentation, providing user training and managing a training group. She has studied to become a webmaster, has worked as a tester for 4 years and has been responsible for managing QA departments for the last three years. Her recent accomplishments have been to define and establish the Quality Assurance departments at Cushman & Wakefield for Citibank, at Grey Advertising for Grey Direct and currently at Ogilvy & Mather for Ogilvy Interactive. Jeanette is recently

married and lives in New York.

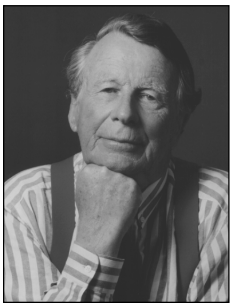
Bert Lamar - Bert is the Lead Quality Assurance Tester at OgilvyInteractive, a division of OgilvyOne. Bert got his start in Quality Assurance at the Vanguard Group of Investors, where he completed a six-month QA internship before accepting a full-time position testing financial transaction processing applications. In addition to six years of Quality Assurance test experience, Bert is a documentation specialist and trainer, and has developed training programs and materials for the Vanguard Group, Cushman & Wakefield, Citibank and Chase Manhattan. Bert also serves on the board of directors of the Girls' Vacation Fund, a non-profit organization that owns and operates Camp Oh-Neh-Tah, a summer camp for low-income girls from New York City and upstate New York. Bert is married and lives in New Jersey.

BACK TO QW2000 PROGRAM

Welcome to

OgilvyOne worldwide

World's Strongest, Most Experienced Network



OgilvyOne was founded over 25 years ago by David Ogilvy

"Direct Response was my first love. And later my secret weapon."

- During the 70's and 80's our company has grown outward from New York to cover the world

2

Ogilvy Interactive

- The first global advertising agency with a dedicated interactive capability (beginning in 1984).
- Today, OgilvyInteractive is the largest integrated interactive agency in the world, with 22 offices and 12 service bureaus servicing more than 40 countries.

3

New York: Client List

- Air France
- American Express
- Brooklyn Museum of Art
- Butch
- Childline
- Cotton Incorporated
- Ford Motor Company
- GTE
- Hershey's
- House of Mao
- IBM
- Jaguar
- Kimberly-Clark
- Kodak
- Kraft Foods
- Unilever
- Lotus
- MotherNature.com
- Ortho Biotech (J & J)
- Perrier Group of America
- Petersen Publishing
- Royal Mail
- Safari Lager
- TIAA-CREF

4

Jeanette Folkes

- OgilvyInteractive
  - QA Manager
- Grey Direct
  - QA Manager
- Cushman & Wakefield / Citibank
  - QA Manager for Java application
- Wall Street Access
  - Training Manager
  - Directory of Office Automation
  - WebMaster
- WSP&R
  - Trainer
- Transammonia
  - Help Desk Analyst
- The NPD Group
  - QA Tester for proprietary client server application
  - Project Manager

5

Bert Lamar

- OgilvyInteractive
  - Senior QA Tester
- Grey Direct
  - Senior QA Tester
- Cushman & Wakefield/Citibank
  - Senior QA Tester
- Vanguard Group of Investors
  - Lead QA Tester

6

QA Methodology

Ensures that a product or service satisfies the customer's needs, wants and expectations

- Team
- General Process
- Documents
- Integration

Quality Assurance Growth

- Dedicated staff established May 1999
- QA Methodology adopted August 1999
- Full-time staff hired September 1999
- QA lab established ?

QA Team

- QA Manager hired 5/99
- Senior Tester hired 9/99
- 2 Junior Testers hired 10/99
- Tester hired 2/00
- 2 additional Junior Testers hired 3/00
- Additional Tester hired 4/00
- Consultants on an as needed basis

9

Team hours

- 10:00am - 10:00pm, Monday - Saturday
- Consultants on an as needed basis

10

Skill Sets - Jr. Tester

- Responsible for all manual testing
- Entry level position
- Familiarity with basic HTML
- Own web site
- College editor
- Has own computer
- Good software background
- PROOF their resume
- Evening and weekend hours

11

Skill Sets - Tester

- Responsible for all testplanning
- At least 2 years testing
- At least 1 year testplanning
- Familiarity with basic HTML
- Good software background
- Flexible hours

12

Skill Sets - Senior Tester

- Responsible for all automated testing
- At least 4 years testing
- At least 2 years testplanning
- Familiarity with HTML, plug-ins, software
- Basic programming

13

General Process

- Test plans, outlines of the areas to be tested, are written
- Test cases, documentation of the actual steps, are written
- Automated testing is used to improve testing
- Exceptions are logged in an exception tracking database
- Reports are generated

14

Documents

- QA Checklist
- Test plan
- Exception reporting
- QA authorization report

15

QA Checklist

- Browsers / OS's
- Required plug-ins and versions
- Alt tags
- Print testing
- Links - pop-ups, new windows
- Forms - validation
- Resolution
- Load testing
- Connection speed
- Any other relevant information

16

Test Plan

- General overview of what's being tested
- Lists all test cases created
- Cannot be done until documentation is received

17

Exception Reporting

MS Access database

- URL
- Problem Type
- Reproducible
- Browser / OS
- Tester
- Test date
- Assignee

18

QA Authorization Report

- Exceptions by type, quantity
- Sign off from QA or authorizing person

19

Integration

- Include QA in all schedules and estimates
- Initial QA performed on Asset Inventory
- Test plans are written
- Test cases are generated
- Testing begins
- Exceptions logged
- QA authorization report

20

Real World Challenges

QA needs money and an enforcer

- Staff
- Equipment
- Working with developers
- Getting management buy-in
- CYA

21

Staff

- Keeping them motivated
- Showing appreciation
- Remaining flexible

22

Equipment

- QA Lab
- Each tester has 2 computers
- Dial up connection

23

Working With Developers

- Avoiding us vs. them

24

Getting Management Buy-in

- More than $
- Constant reporting
  - Let them know before the client does

25

CYA

- Adobe Acrobat
- MS Word
- Export all exceptions

26

Production-related challenges of Web Testing

- Finding adequate time for Test Planning
- Planning with incomplete or no specifications
- Finding time for regression testing
- Finding time for Automated testing
- Continuing Education

27

Documentation required for Test Planning

- Functional Design Document
- Sitemap
- Copy deck
- iBoards

28

Test Planning with no Site documentation

- QA Checklist
- Sitemap Tools
  - Astra Site Manager
  - Linkbot

29

Regression Testing

- Finding Time
- An automated regression test regime
  - Linkbot
  - Adobe Acrobat
  - SilkTest

30

Automated Testing

- Requires additional time for planning
- Requires code to be frozen or complete
- Does not work with certain interactive media

31

Continuing Education

- Research
- Training
- Industry Group Affiliations

32

QW2000 Paper 7M1

Mr. Patrick Copeland
(Microsoft)

Redesigning a Testing Organization for Delivery to the Web

BACK TO QW2000 PROGRAM

Key Points

- Businesses are becoming internet-centric. This prsents several challenges to a traditionally organized test team.
- Microsoft's WebData Team (which delivers XML) has shifted to this model and is a good case study for successful change.
- Focusing on web-centric delivery mechanisms and timelines required a complete redesign of our processes.

Presentation Abstract

WebData is a newly organized team with the mission to ship rich-featured high-quality world-class data access components to the Web. Moving away from longer product cycles, we've overhauled the way we do business.

By adjusting our thinking we developed a plan that prioritized and set attainable milestones that put WebData on the path of improvement and allowed us to deliver in the fast paced world of the internet. Six ingredients were essential: automation, development, growth, information, planning, and process.

About the Author

Patrick Copeland manages WebData's Quality Assurance Team for the SQL Programming Model at Microsoft Corporation. His team develops the testiong process and strategy for Microsoft's newest and most flexible database components, such as XML. Throughtout the product cycle, he's responsible for maintainina dn shipping high quality builds for dozens of configurations, multiple languages, and numerous client-server combinations.

As an undergraduate he attended the University of Arizona, and he received an MS from the University of Southern California.

BACK TO QW2000 PROGRAM

Redesigning a Testing Organization for Delivery to the Web

Patrick Copeland
SQL Server Quality Assurance Manager
Microsoft Corporation

Concepts

- Why WebRelease?
- What is success?
- Web-time
- Quality Bar
- Feedback

Political Speed Bumps

- Lack of agreement
- Team treading water
- Decision making slow
- Empires, home-grown processes, territoriality

Structural Speed Bumps

- Built for monolithic releases
- Functional team controlled labs
- Different processes and tools
- Duplication
- Underutilization

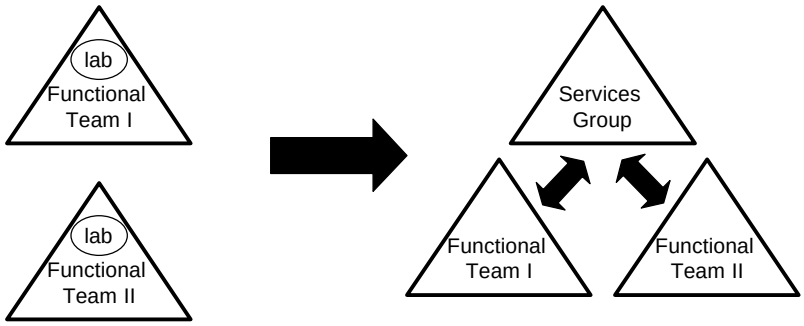
Reorganization Rules

- Break down fortifications
- Needed to be Surgical
- Faster delivery
- Quality levels same or higher

Service Group Defined

- Mission
  - “provide foundation of processes and solutions to help ship world class data platforms faster”
- Uniformity
  - Labs
  - Build
  - Tools
  - Setup and release
  - Sustained engineering teams

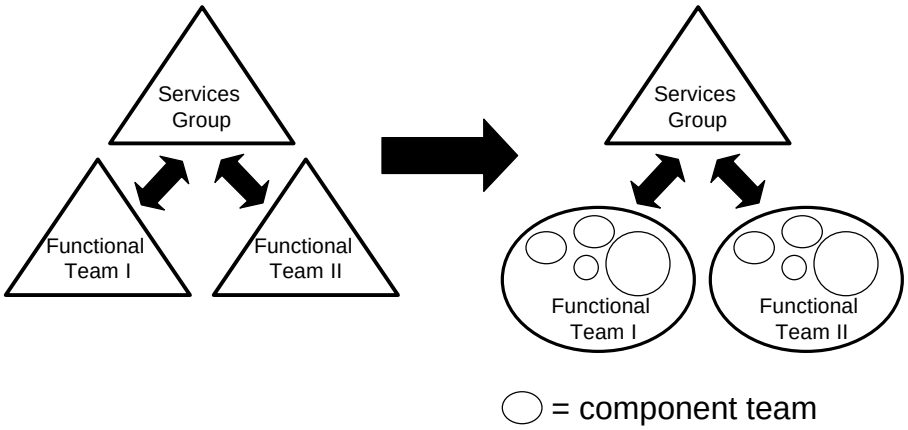
Service Group Transition



Component Teams Defined

- Empower smaller teams
- Faster Paced decisions
- Risks:
  - knowledge more compartmentalized
  - coordination would be more difficult
- No structural change
- Created virtual organizations

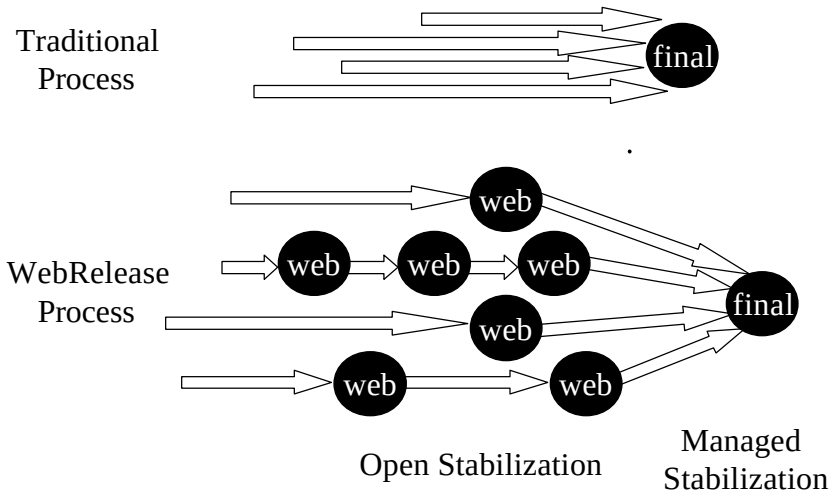
Component Teams Transition



Stabilization Phase

- Open Stabilization
  - Component teams drive
  - Lots of autonomy
  - Component WebRelease
- Managed Stabilization
  - Global team drives
  - Reduced autonomy
  - Integrated WebRelease and Final Product

WebRelease



Ongoing Improvement

thePlan

- **Automation**
- **Development**
- **Growth**
- **Information**
- **Planning**
- **Process**

Summary

WebRelease is…

- the buzzword to capture our structural, process, and cultural changes.
- a way to release products.
- the way that we focused and redesigned our team.

Microsoft®

Questions

Latest WebRelease can be found here:
http://www.msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp

Redesigning a Testing Organization for Delivery to the Web

Patrick Copeland
SQL Server Quality Assurance Manager
Microsoft Corporation

Introduction

Gone are the days when developers are willing to wait six months to glimpse new features. E-businesses need solutions to data access problems yesterday. They want to hear about what's coming down the pike, they want to see why it's valuable to them, and they want to know how they'll be able to use it.

Customers expect more partnering today. Nine months ago, we realized that we were getting creamed by the competition. We saw several rivals delivering new product features directly to the Web. They were beating Microsoft to the punch and, more importantly, stealing mind share from our solutions. By ignoring the new world of software Glasnost on the Web, we were putting our projects, and our development community, at risk. What follows is the story of how we restructured our thinking around what we call WebRelease. Most of the work was done during the Microsoft® SQL Server™ 2000 project cycle.

WebData was structured to accommodate the typical release model: spec, design, code, test, beta, release. As a component group, we had several major commitments to battleship projects, including Office, SQL, and Windows® 2000. We were moderately insolated from our customers and structured to deliver at what we would now consider the leisurely pace of every 3-4 months. When we initially asked our sub-teams about moving to the faster WebRelease strategy, we heard screams of agony, "How do you expect us to ship more frequently? We're already breaking our necks!" We realized that our approach to change would need to be organic, credible, and revolutionary.

Concepts

Initially, there were two important questions. What were we trying to solve? How would success be measured? We came up with two simple answers:

1. The WebData team's goal was to enable faster adoption of our technology by Web and database developers.
2. We would measure success by tracking the adoption rates of the latest releases.

Simple? Well, not exactly. In addition to understanding the goals, framing and defining the problem space was necessary. This required some market research.

The first concept researched was "Web-time." The competition was delivering new features and previews at a staggering pace. They were clearly ahead of us. We knew we needed to be faster, but we also needed to decide how fast was fast enough. We learned that customers wanted to be early adopters, but they didn't want to be on an upgrade treadmill. They wanted time to understand and use new features, but they also wanted to avoid spending all of their time rewriting their code to keep pace. We needed to find a balance between satisfying the development community and protecting the customer from the chaos of constant integration.

Another concept was the "quality bar," the idea being that each product has a level of quality that is measurable. The level lies somewhere between bug-free and unusable. Early adopters are willing to accept some rough edges in return for a glimpse of newer technology, but they are unwilling to accept delivery of software that is significantly crippled due to coding errors.

"Feedback" was the third concept. A key component in empowering a development community is soliciting feedback and utilizing it. Throwing products onto a Web site is no longer enough to satisfy the technology-adopting customer. As seen with the enthusiasm of the "open source" movement, allowing free channels of communication invites the customer to have impact as well as a voice in the design and feature set of new programs. This openness also allows Microsoft to target the right set of features for the audience and provides a test bed of real-world environments. Our presence on the Web required places for new products, certainly. More importantly, our solution needed: defect lists, Q&A forums, bug submission forms, feature request aliases, developer chat, and in a nutshell openness.

Speed Bumps

When WebData initially began to tackle the myriad of problems between where we were and where we wanted to be, the consensus from the team was a resounding "You're nuts!" The way the team did business needed an overhaul. Due to a constant atmosphere of fire fighting, reorganization, and general lack of planning, the team was treading water; we were having difficulty moving forward. This section will touch on a few of the major structural and political speed bumps.

We had been originally organized around a premise of monolithic releases. At any given moment, the entire team was focused on a single major milestone. While this model has its benefits, it isn't suited for impatient Web customers. On the one hand, tracking progress was easy because only one date was significant. On the other hand, a hierarchy of management was entrenched in homegrown processes that slowed down the making of decisions. Given the rigidness of the organization, shortening the release cycles wouldn't allow enough time to fully test integration. There was a perception that we would throw quality out the window if we tightened up the schedule. The running opinion was that less time equals lower quality.

Along the same lines, management of the test and build labs was controlled by the functional teams. WebData was divided into two parts: one working on a mature code base, and the other on brand new functionality. To facilitate coordination between the two functional teams, labs had been created within each organization. This had led to a situation where each lab used different processes and tools. Also, because the focus of the functional teams was to ship, corners were cut on infrastructure that left the labs in a constant fire-fighting mode.

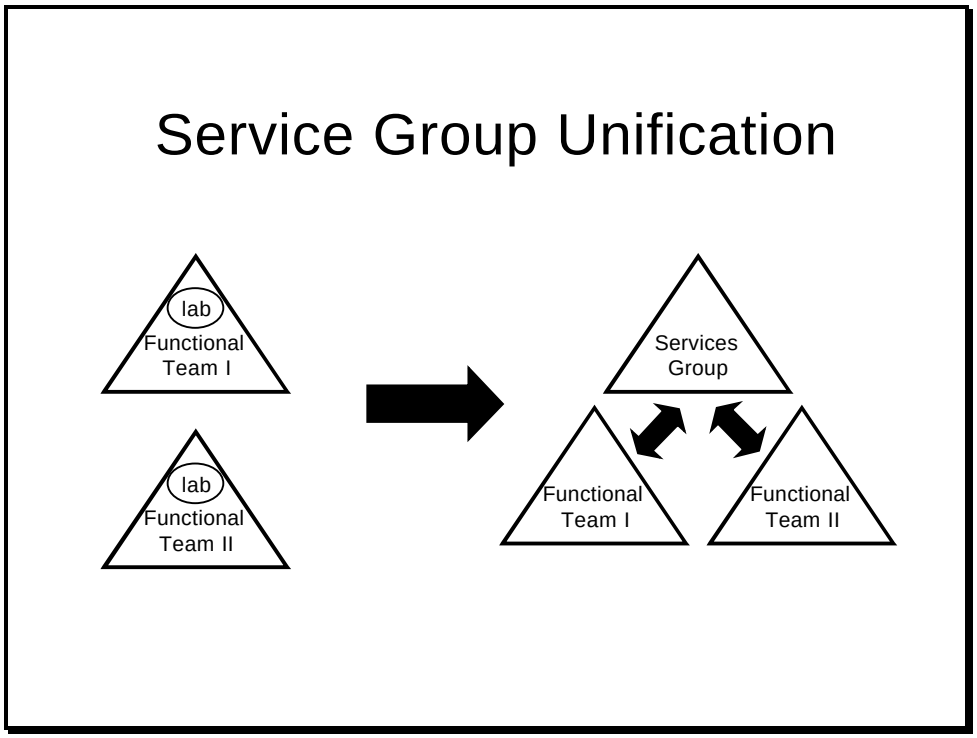
In general, the team wasn't quality assurance focused. Schedules and features were agreed upon without consideration of testing schedules. Planning phases were optimized out of the process because we didn't have any time to spare. Quality assurance was something done at the end of the project. Being the final group to catch the hot potato, testing was frequently blamed for the delays and truncated features. Many of the real and perceived roadblocks were typical of any traditionally structured development process. From time to time, all teams struggle with holding to promises. When randomizers dominate a project, without any coping mechanisms, teams tend to overcommit and underdeliver.

Building empires, establishing process solely for its own sake, and territoriality are universally maligned. In organizations where chaos is the norm, many teams fortify themselves and attempt to become impregnable from the chaos around them. These walls typically are built over several releases, and their reason for being is forgotten. WebData had plenty of good targets for our first attack on the fortifications. It was clear that we needed to do surgical reorganization. Our challenge was to identify and build solutions for the problems we could solve while learning to accept the remaining chaos in the system. Our approach was to put in place a structure that allowed WebData to deliver faster and with the same or higher levels of quality. We asked our team to think about the question, "What role did I play in allowing problems to persist?"

Nothing can refocus a team like new vision and leadership. WebRelease would be the buzzword to capture our structural, process, and cultural changes. WebRelease would be a way to release products. WebRelease would also be the way that we focused and redesigned our team.

Reorganization

The first order of business was to consolidate the labs into a single group. These labs were responsible for running a matrix of hardware, software, and test configurations generated by the functional groups. Originally, as mentioned earlier, the functional groups each had independent lab teams. This was wasteful on several levels. The move to a single lab structure allowed WebData to leverage spare bandwidth, unify processes, streamline test passes, and reduce overlapping responsibilities and staff. The lab became the engine of the testing organization. Its capacity dictated throughput, and it was critical to make it as efficient as possible. More importantly, it was now predictable.



This slide shows how WebData moved from a group with multiple functional teams, each having an lab, to a group with all of the services consolidated on a separate team.

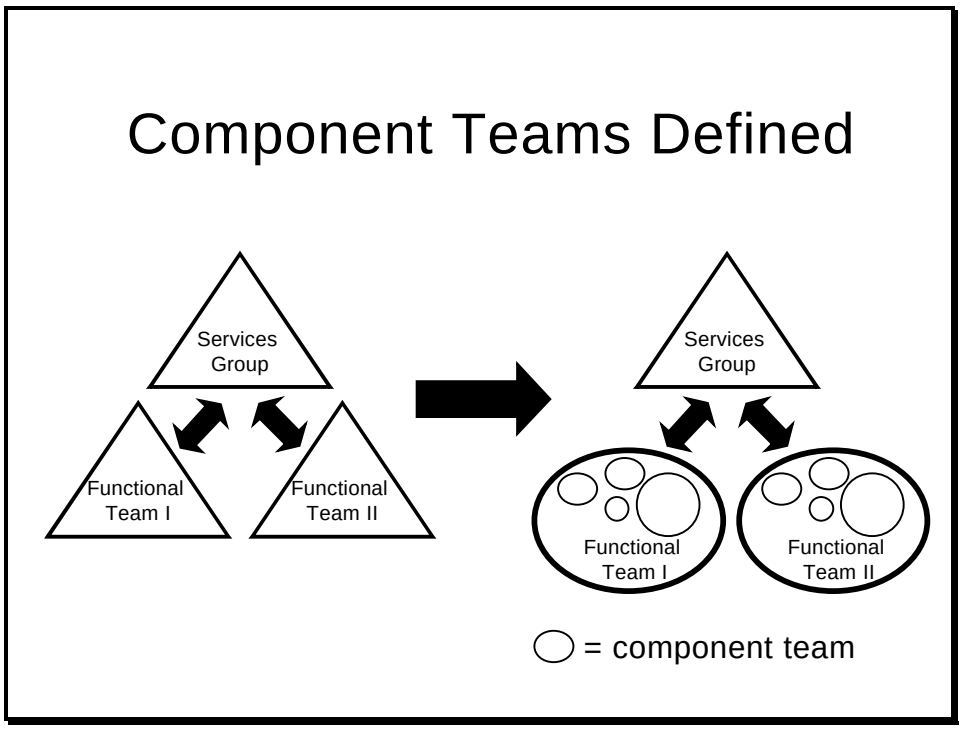
Consolidation of the labs went so well that it motivated the building of a Services group. This group's mission was to "provide a set of technologies and services that act as the foundation for an integrated set of business processes and solutions to help ship world class data platforms faster." The gains observed by having a single lab org were abstracted and applied to the build, tools, setup and release, and sustained engineering teams. As an autonomous entity, the Services group gained the power to plan and fix problems independent of the functional team's mission, schedule, and demands (the issues that historically prevented progress). The group's sole focus was to create optimal, unified, and reusable infrastructure. For example, build machines were now leveraged to compile multiple projects. In the past, because each team had a set of machines, many were sitting idle most of the time. The Services group was now poised to push change throughout the rest of WebData.

A microcosmic recurrence of WebData's lack of uniformity in process and methodology appeared for a brief time in the Services group. The team had taken on a collection of responsibilities and technologies that were neither uniform nor integrated. We solved this problem by removing duplication, unifying process, and investing heavily in tools. The multiple build process, for instance, was turned into a single generic process that could be applied to any project. This transition took time and a large investment in engineering. The new team had several advantages:

1. Lower learning curve—only one system to be understood and taught
2. Single system to maintain—lower number of points of failure
3. System advances had the effect of improving the entire process
4. Cross-pollination—better cooperation and collaboration across teams
5. Reduced duplication of effort and Uniformity in operations output

Within the functional teams, a different surgical reorganization was done. We took an approach opposite the consolidation done for the Services group. Instead of unification, we componentized the functional teams to a greater degree. We had seen that monolithic team structures slowed down decision making. By empowering smaller teams to make choices and trade-offs based on their respective product features, we saw that we could quicken the pace.

The risk of fracturing the groups in this way was that knowledge could become more compartmentalized and that coordination would be more difficult. To avoid this problem, we didn't structurally change the team's chain of command but created dotted line, or virtual, organizations.



The functional teams were reorganized into smaller component teams in order to increase responsiveness to customers and speed the development of new features.

Stabilization Phases

With the team reasonably organized, it was time to turn attention to process. With a number of new and less globally focused component teams came a need to impose some rules to avoid randomness. The project was divided into two phases:

1. Open Stabilization
2. Managed Stabilization

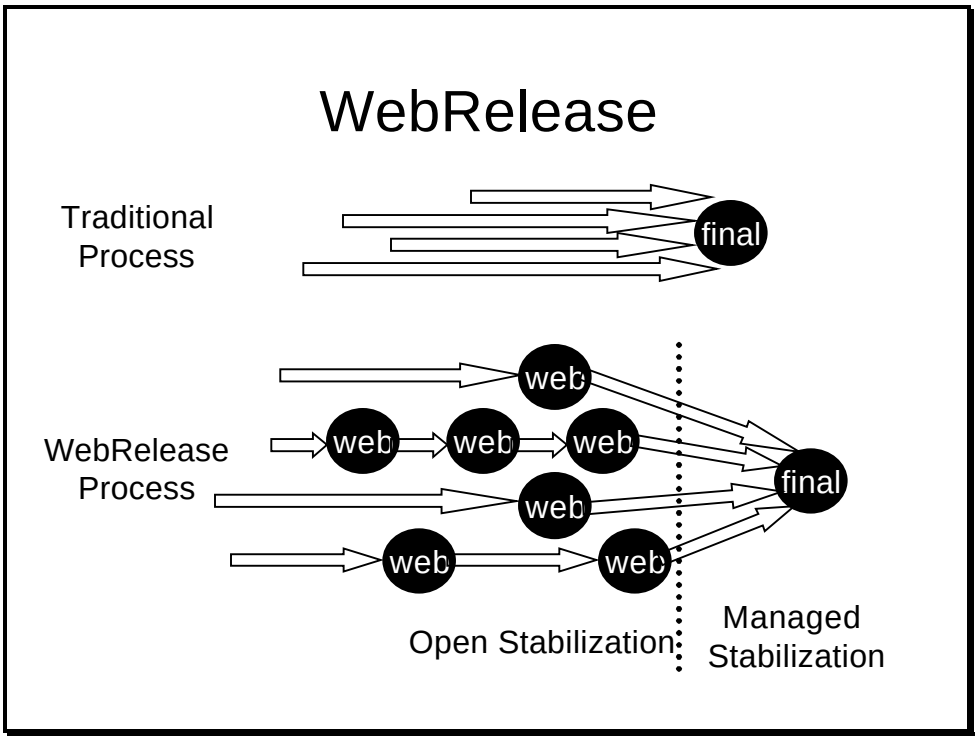
Open Stabilization

The Open Stabilization phase began when the coding was finished. Component teams were required to meet regularly to discuss the priority of bugs, the proper milestone for the fixes, the correct assignment, and appropriate resolution of all bugs within their areas of ownership. During this phase, the component teams were the sole decision makers and had nearly complete autonomy to take on or reject work items. These groups were assigned responsibility for the following areas:

1. Managing the bugs within a specific technology area.
2. Reviewing progress vs. the schedule.
3. Reviewing new feature requests within their control.

To provide some consistency, the Global Triage group was created as a team made up of senior managers. The primary role of this group was to ship a quality product and was therefore less focused on any individual area. The Global Triage group enforced a symbolic bar that rose with each passing day. The theory was that this would make it harder to make code changes, thus reducing churn and instability, and easier to focus only on showstopper problems. Providing this oversight was a critical part of the stabilization process.

To keep coordination and knowledge flowing between the component teams, the Global Triage group conducted regular feature reviews. These reviews would highlight blockages and assumptions. It was expected that each component team would deliver a snapshot of its progress including, functional code coverage, defect tracking statistics, an accounting of design change requests, and a general catharsis.



Traditional process had multiple teams working on interrelated projects, all converging into a single release and managed by one monolithic organization. WebRelease gives autonomy to the component teams to manage smaller product releases, eventually converging into a final release managed by the Global Triage group.

Managed Stabilization

In the Managed Stabilization phase, the component teams were asked to make recommendations, but their ability to make project-wide decisions was removed. The Global Triage group took control of the project and drove it to completion. The shift in ownership took place a few months prior to release. At that point, the bar became extremely high. Only a few highly reviewed and tested code changes were allowed. Most of the time was spent looking for holes in coverage and assessing the risks involved in accepting a code change. By limiting control during the managed phase and staying very focused, the team was still able to deliver large releases while satisfying the Web development community.

Ongoing Change

Along with structural and process changes, an ongoing improvement program was started. This initiative was simply called **the Plan**. It defined the way we prioritized and set attainable milestones that put WebData on the path of continual improvement. Central to **the Plan** was giving people permission to innovate, and it was made clear that the front-liners owned change.

We framed the challenges by focusing on the "six-pack": automation, development, growth, information, planning, and process. The team's suggestions were each placed in

one of these six buckets. Initially, the teams made hundreds of suggestions. These ideas were prioritized and delegated for research. In total, there were 100 prioritized sub-tasks. They were required to be actionable, clear, and concise. Following is a verbatim description of each central issue:

Automation
We are investing in building a strong tools team, with the mission to create a reliable and sustainable set of harnesses and automation. As we move forward, the functional, tools, and lab teams will be more closely woven together—sharing ideas and partnering to put in place a topflight set of tools and processes.

Development
Beefing and cleaning up existing tests is a priority. Tests need to be reviewed for coverage and usage. Where are our holes? Are we doing enough on high-priority features? We need to answer these questions.

Growth
If all we are doing is fire fighting, we aren't paying attention to a very important part of our team—growth. We need more moments of reflection in the process. We need clear definitions of our roles. We need to pay attention to people being burned out and tired of "doing the same old thing." At the same moment, we need to understand that we are a business and are expected to achieve our goals. A balance needs to be struck between growth and productivity.

Information
A sign of a good test team is its ability to communicate "Where we are." Testing owns driving the quality higher. Improving our organization will require other teams to make adjustments—for example, PMs writing better specs, or developers articulating design trade-offs. They need to know where to focus attention , and testing/services owns the advocacy role for the customers and the product quality. We need to develop a better way of visualizing and delivering status.

Planning
We are going to take a breath and make some changes. Management has been supportive of us reinventing ourselves. Let's hit the hard reset button. There may be some chaos in the short term, but having time to plan is critical and will be factored into all of our schedules moving forward.

Process
Drop the baggage you've been carrying. Why are you carrying that baggage? How can we make our lives easier and more manageable? Adding process means adding focus to our work.

Summary

WebData and WebRelease are still works in progress. The goal is to continually review and invent better ways of supporting the Web community while shipping high-quality bits faster. An explicit non-goal is to allow the fortifications to reenter the team culture.

When we started the process of reinventing ourselves, our initial step was to ask ourselves two questions: What were we trying to solve? How would success be measured? We agreed on several concepts to help us focus our effort: Web-time, quality bar, and feedback. Having a good plan wasn't enough; we had to make some organizational changes to break down historical fortifications such as monolithic team structure, territoriality, and lack of quality focus. The invention of the service team and creation of component teams were key to our success. The Services group was the "hammer" and enforcement branch of the team that helped unify process and resources. The Global Triage group served as the senior voice overseeing the whole process. The creation of component teams gave WebData more flexibility and encouraged the smaller virtual teams to make decisions and control their own release destiny. The final step was to shift our focus from the daily fire fighting to ongoing improvement with **the Plan**.

Our latest WebRelease can be found here:
http://www.msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp

#########
Microsoft, Windows, Office, SQL, and SQL Server are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries.
Other product and company names herein may be trademarks of their respective owners.

© 2000 Microsoft Corporation. All rights reserved.

QW2000 QuickStart 7Q

Mr. Brian Marick
Mr. James Bach
Mr. Cem Kaner

Evaluating Test Suites

BACK TO QW2000 PROGRAM

Key Points

- We give a systematic process for discovering how good a test suite is by examining the suite and questioning its author.
- We also describe likely properties of a good test suite. Different projects will attach different weights to those properties. An important result of evaluation is a clearer understanding of what matters to this project.
- A single evaluation is not sufficient. Instead, evaluating a test suite is part of a continuous conversation about the project's goals.

Presentation Abstract

If a test suite is used to judge something about the goodness of a software product, surely it itself must be good. And surely its goodness must be evaluated before its results can be trusted.

This paper is about how to do that.

We will first discuss how one might statically evaluate a finished test suite. As a part of that evaluation, we will begin to answer the question "what does it mean for a test suite to be good?" We'll illustrate the analysis with a specific example. The static evaluation procedure will include these elements (not all of which would be used in any given situation). Note: the final paper's list will doubtless differ in some ways.

1. Is the suite's purpose clearly understood? A test suite is created to serve some person, often a project manager wanting to know if the software is ready to ship, sometimes a developer wanting to know if it's time to move on. The customer of the suite has certain criteria. A suite whose creator doesn't know those criteria cannot be a good one.

2. In practice, the customer may have only vague criteria. In even the best case, a suite's creator should have refined the criteria through a risk analysis. Does the suite's creator have a good understanding of:

* The different classes of end users and their relative importance to the project?
* What kinds of failures matter most to those different end users?

* What kinds of failure-producing situations the product is likely to encounter?
* What characteristics of the product are shakiest?

A suite whose creator cannot answer these and related questions will be misdirected.

3. Is the suite's approach appropriate to the risks? Can the suite's creator describe (or has she already described) how the decision about what, where, and how to test was guided by the risks? Has the approach chosen been compared to any checklists of common test practices?

4. Were the right tests retained? Here, we distinguish between two different definitions of "the test suite". One definition is "all the tests run against the product up to the present time". The other is "all the tests that will be run again in the future". It often makes sense to discard tests if they are not expected to be likely to find bugs in the future. Can the test suite creator articulate the policy by which the tests were discarded or retained? Does it seem to sort tests into the right categories?

5. Are the retained tests maintainable? For those tests that are automated, do they avoid the automation trap: tests that are intended to be rerun, but are so sensitive to product change that the testers will spend all their time trying to keep old tests working, none of their time creating new tests? Do manual tests use elaborate scripts? Do these combine the worst aspects of automated tests (poor maintainability) with the labor-intensiveness of manual testing?

6. What does the suite not do? Here, we evaluate the suite against various types of coverage. Code coverage is one example, but there can be many others (such as checking whether every statement in the user documentation has been tested). More than one type of coverage is appropriate. A larger set will compensate for the weaknesses of any given type.

In all uses of coverage, we must take care to match the coverage against the test approach. Branch coverage, for example, says little about the goodness of load testing of multi-user systems.

You will note that this approach is highly subjective. We will not duck this issue. We will in fact confront the tendency to retreat to those few measures that are completely objective (such as code coverage), as we believe they do more harm than good.

About the Author

James Bach (http://www.satisfice.com) is founder and principal consultant of Satisfice, Inc. James cut his teeth as a programmer, tester, and SQA manager in Silicon Valley and the world of market-driven software development. He has worked at Apple, Borland, a couple of startups, and a couple of consulting companies. Through his models of Good Enough quality, exploratory testing, and heuristic test design, he focuses on helping individual software testers cope with the pressures of life in the trenches and answer the questions "What am I doing here? What should I do now?"

Brian Marick has worked in testing since 1981. A consultant since 1992, he concentrates on developer testing, the interface between developers and independent testers, the criteria for test evaluation, and helping teams and projects understand and manage the tradeoffs inherent in software assurance. He is the author of The Craft of Software Testing (1995) and was the first editor of Software Testing and Quality Engineering Magazine.

BACK TO QW2000 PROGRAM

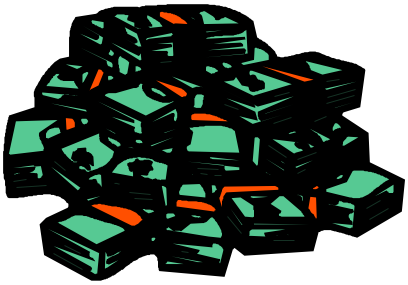
A Manager's Guide to Evaluating Test Suites

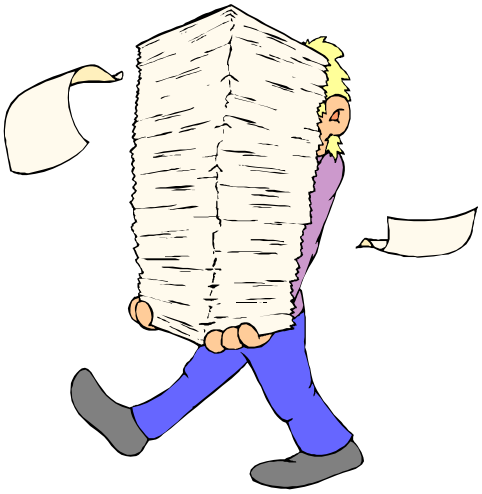
Brian Marick, Testing Foundations
James Bach, Satisfice, Inc.
Cem Kaner, kaner.com

www.testing.com, www.satisfice.com, www.kaner.com
marick@testing.com, james@satisfice.com, kaner@kaner.com

Copyright © 2000 by Brian Marick, James Bach, and Cem Kaner. All rights reserved.
Some images copyright www.arttoday.com

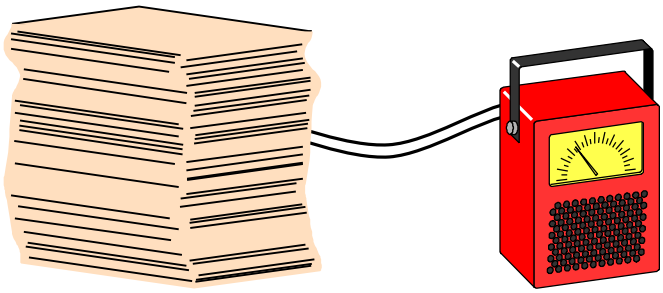
Your Problem





How Good Is It?

What You Want



Sorry

Objectivity

	Objective Methods	Subjective Methods
Different people get…	… the same answer	… different answers

Objectivity

	Objective Methods	Subjective Methods
Different people get…	… the same answer	… different answers
End result	HIGH chance of BAD answer	

For <u>this</u> task

Objectivity

	Objective Methods	Subjective Methods
Different people get…	… the same answer	… different answers
End result	HIGH chance of BAD answer	Opportunity for a USEFUL answer

Our job is to gain the skills and experience to be useful

An Evaluation Approach

- Evaluate the test suite from several broad perspectives
  - is the suite targeted toward what's important?
  - are maintainability and repeatability handled well?
  - what got overlooked?
- Primarily by asking questions of its developer

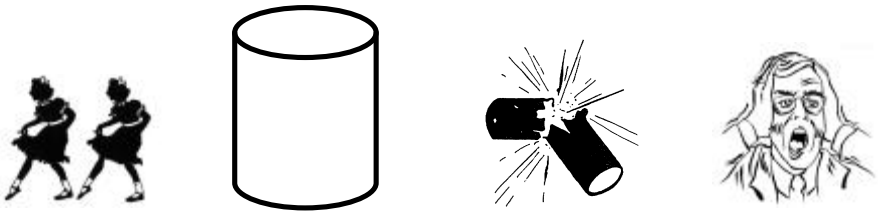
Perspectives

- ➢ Is the suite targeted toward what's important?
- Are maintainability and repeatability handled well?
- What got overlooked?

Step 1:
What Does the Tester Need to Know?

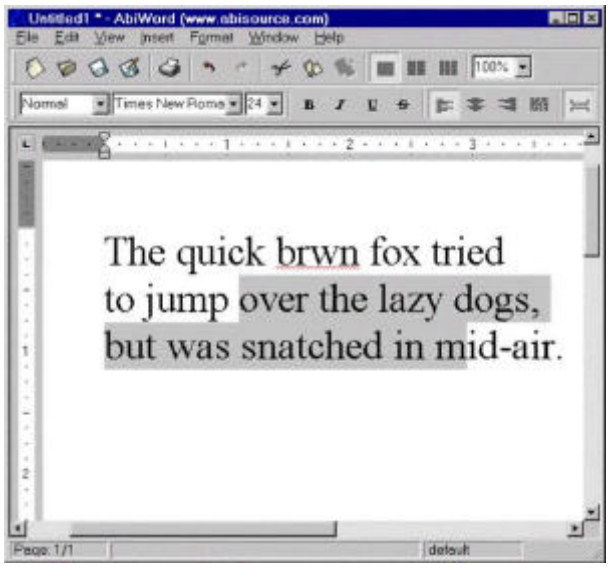
- Evaluate her <u>specific</u> understanding of how best to look for bugs
- James Bach's risk model

What Does It Mean for a Product to Have a Bug?



Threat → Product → Problem → Victim

By Way of Example



www.abisource.com

Victim

Who are the important classes of user?

The church secretary

- Consequences
  - concentrate on basic features
  - things should "just work"

Crossing the Chasm,
Geoffrey Moore

Problem

What types of failure matter most?

- Crashes, especially if data is lost
- Failure to import from Word
- Usability of undo

Threat

What failure-producing situations is the product likely to encounter?

Win95, Win98, WinNT, Win2000, BeOS, Linux, *BSD, …

- Consequence: much configuration testing
  - note: especially installation and first-invocation problems

Product

What areas in the product are shakiest?

Printing was a problem a while back…

(Sign of unstable area?)

Artifacts on on-screen layout…

(Manual testing)

Undo seems to be completely solid…

Step 2: Is the Suite's Approach Appropriate to the Risks?

We are favoring task-based tests over functional tests because…

(covering basic operations, usability)

We favor manual testing because…

(screen artifacts, usability)

… except that our core set of configuration tests…

(cross-platform emphasis)

Perspectives

- Is the suite targeted toward what's important?
- ➢ **Are maintainability and repeatability handled well?**
- What got overlooked?

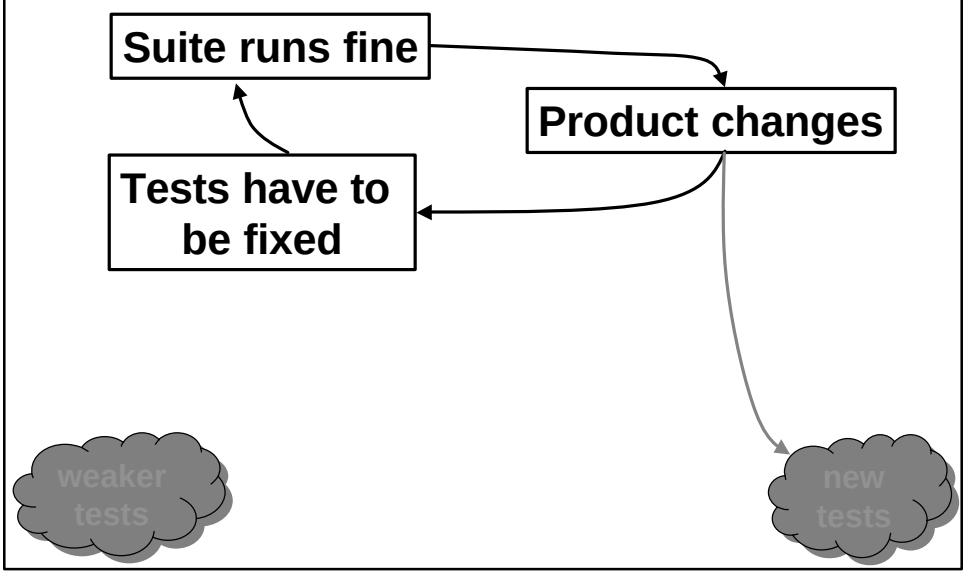
Repeatability Is a Tradeoff

Fewer unique threats

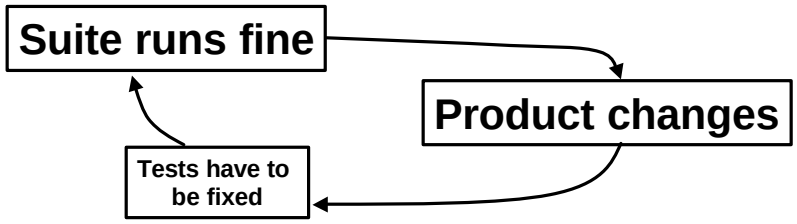
More re-execution

- **There should be unrepeated manual tests**
- **There should be automated tests**
- **Is there a good balance?**
- **Can the tester articulate her reasoning?**

How's the Maintainability?



Avoiding the Automation Trap



What was the approach?

Describe the last three changes you made to the suite

Perspectives

- Is the suite targeted toward what's important?
- Are maintainability and repeatability handled well?
- ➢ **What got overlooked?**

Look at a Sample of Tests In Detail

- Approach not followed?
- Gaps in approach?
- Gaps in tester's skill set?

You know… these tests just aren't gonna find enough bugs…

- Execute tests manually
- Watch the tester run tests

What Has the Suite Missed?

- Any suite misses bugs
- Bugs are found by
  - other project members
  - beta testers
  - customers

Use Missed Bugs

- Examine them and ask why they were missed
  - is the test approach flawed?
  - has it been implemented poorly?
- Do not over-react

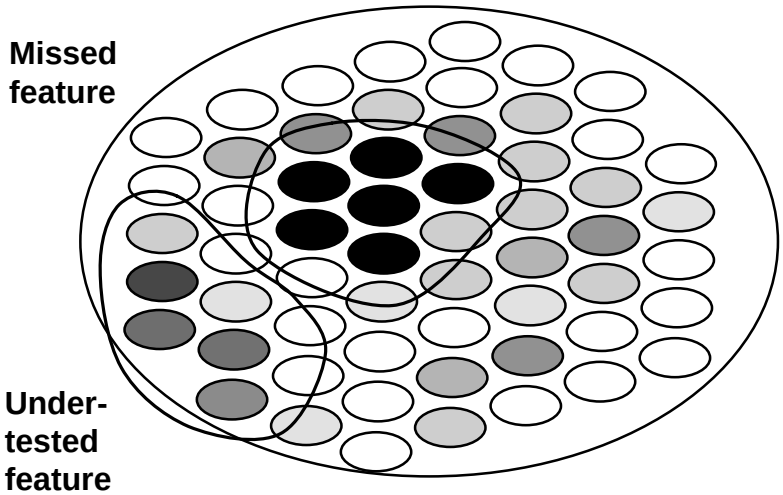
Coverage

- A simple form of code coverage

```
status = perform_operation();
if ( status == FATAL_ERROR )
    recover from error
perform normal, non-ERROR action
```

- Statement coverage measures which lines of code in the program have not been executed

Coarse Granularity Coverage



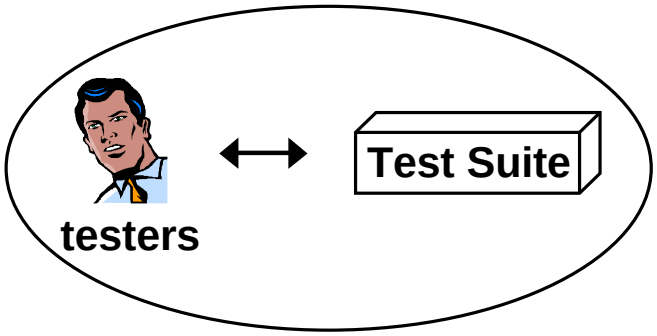
More on coverage in paper

Summary

- This approach is highly subjective
  - it relies on <u>your</u> skill and experience
- Important skills
  - testing
  - human communication and cooperation
  - good overall judgement

Bach's Commentary

The Testing Enchilada



Evaluating a test suite is not enough, if you want to understand the quality of your testing. Your "suite" may represent a fraction of the test process.

Bach's Commentary

- **Exploratory Testing:** Is there any testing you do that is not enshrined in a suite? If not, why not?
- **Reporting and Advocacy:** In what way do you report results and deliver value to your clients?
- **Tester Role:** In what way do testers add value to the suite, or make it work?
- **Test Cycling:** When is testing done? How is that coordinated with new builds? How long does it take?
- **Authorship:** How do you avoid test suite stagnation? Is the suite a work in progress, or a sacred relic?

Evaluating testing is not a simple, linear process.
It's an ongoing exploration.

Kaner's Commentary

Evaluation, eh?

What do you think you mean by "evaluation"?

Beware of Silly Metrics

Kaner's Commentary

- The best suite is the one that:
  - Finds the most bugs?
  - Covers the most code?
  - Represents the most common customer use patterns?
  - Is most automated?
  - Is most easy to explain to executives?
- Why?

LET'S FOCUS ON BUG COUNTS

Kaner's Commentary

Metrics:

- **Attribute:** goodness of the test suite
- **Measure:** bugs found during testing
- **Relationship:** what if we increased testing goodness by 20%? Can we do this without increasing bug count?
- **Side effects:** what if we increased the bug count by 20%? Can we do this without improving testing?
- **Purpose of measurement?** Discovery (education) or Motivation (management / performance standard)?
- **Purpose of testing?** Does this measure help us fulfill our testing mission? ***What is our testing mission?***

Kaner's Commentary

Models based on

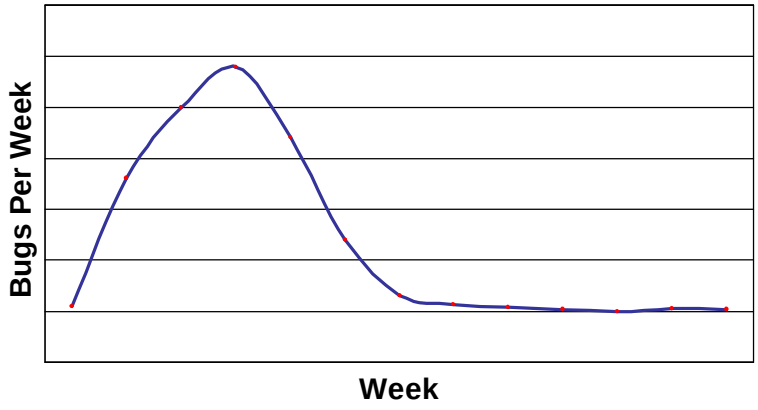
silly metrics

can lead us into

weaker testing.

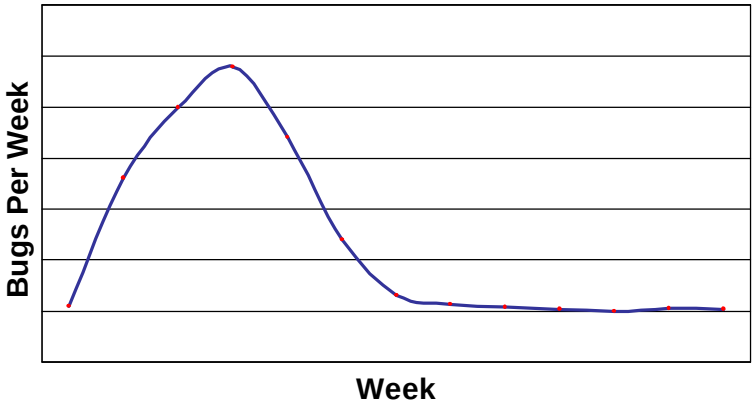
Kaner's Commentary

Reliability Curve?



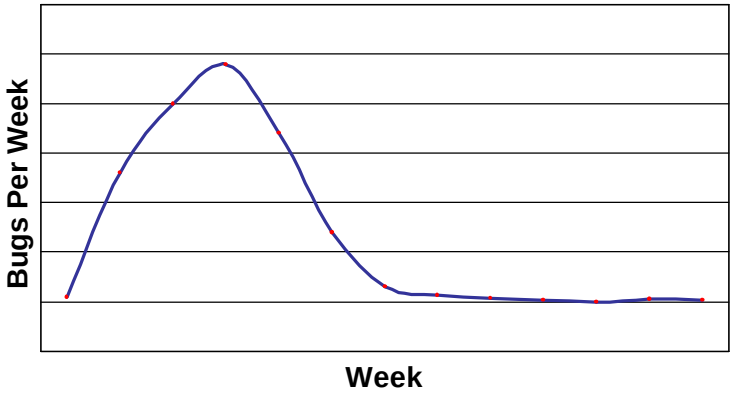
Kaner's Commentary

Immunization Curve?



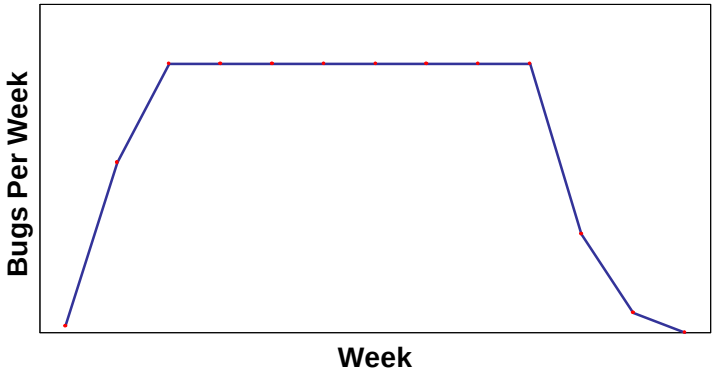
Kaner's Commentary

Test Case Power Curve?



Kaner's Commentary

Shouldn't We Strive For <u>This</u> ?



Kaner's Commentary

Conclusions:

- Goodness of testing is context dependent
- Evaluation of testing without evaluation of context will yield undesired side effects
- Models based on simplistic evaluations are misleading
- Test design, development, maintenance and extension are done by humans, for humans. The value of the work depends on how well it serves the involved or affected humans.

A Manager's Guide to Evaluating Test Suites

Brian Marick, Testing Foundations
James Bach, Satisfice Inc.
Cem Kaner, kaner.com

Whatever else testers do, they run tests. They execute programs and judge whether the results produced are correct or not. Collections of tests – test suites – *are expensive to create. It's fair to ask how good any given suite is. This paper describes ways we would go about answering that question.*

Our audience

There are two broad traditions of testing. One holds that the purpose of testing is to justify confidence in the program. Reliability modeling [Lyu96] is in that tradition. The other tradition focuses on producing valid bug reports. We will not argue here which tradition is correct. We merely state that we belong to the second tradition. This paper is addressed to people like us.

Evaluating a test suite one time

In this section, we'll assume that you have been presented with a test suite, the tester who created it, the bugs it found, and all relevant documentation. We'll present steps you could use to evaluate the suite. When detailed discussions of topics are readily available online, we only sketch them here.

Step 1: What is the suite's purpose?

Suppose you have two test suites, Suite1 and Suite2. Is Suite1 better if it finds more bugs? Not necessarily. Let's explore the issue by way of an example.

Suppose the product has five subsystems. For whatever reason, the testing team devotes almost all its effort to subsystem 1 and files 100 bug reports. The development team works diligently and fixes them all. The testing team can't find any bugs in the fixed subsystem, so the product is released. After release, we discover that the testing team did a good job on subsystem 1: no bugs are reported in it. 12 bugs are reported in each of the undertested subsystems. Here's a table representing this scenario:

Contact information: marick@testing.com, www.testing.com; james@satisfice.com, www.satisfice.com; kaner@kaner.com, www.kaner.com

Copyright © 2000 Brian Marick, James Bach, and Cem Kaner. All Rights Reserved. Permission granted to copy for personal use.

Some images copyright www.arttoday.com.

	Bug reports before release	*Bug reports after release*	
Subsystem 1	100	0	
Subsystem 2	0	12	
Subsystem 3	0	12	
Subsystem 4	0	12	
Subsystem 5	0	12	
	100	48	148

Here's another p ossible scenario. The testing team spreads its work more evenly among the subsystems. It finds only 50 of the bugs we know are in subsystem 1, but it also finds half of the bugs in the other subsystems. After release, all unfound bugs are reported by customers. We have the following situation.

	Bug reports before release	*Bug reports after release*	
Subsystem 1	50	50	
Subsystem 2	6	6	
Subsystem 3	6	6	
Subsystem 4	6	6	
Subsystem 5	6	6	
	74	74	148

In which scenario did the testing team do the better job? In the first one, they find more bugs. But there's something disconcerting about the notion that they did a better job by mostly ignoring most of the system.

We can illustrate why they in fact did a worse job by considering what the test manager can tell the project manager *before release* in each of the two scenarios.

In the first scenario, the test manager can say, "We tested subsystem 1 into submission. We found a lot of bugs, and they got fixed. We've tried really hard to break it again, and we think it's now solid. Unfortunately, we don't really know anything about the stability of the other subsystems." The team is presenting the project manager with a gamble: just what's going to happen in subsystems 2 through 6 after release? Will there be a few bu gs? Or will each subsystem have 100 bugs? Since we know the future of this project, we know

that the gamble to release would pay off – but the manager might not care to make that bet.

In the second scenario, the test manager can say, "We've done a pretty thorough pass over all the subsystems. We've found only a modest number of bugs in subsystems 2 through 5, so we predict few will be found there after release. But subsystem 1 is very shaky. We've found a lot of bugs, and we expect that there are still more to find. *What do you want to do?*" The team is presenting the project manager with information, not a gamble. It's unwelcome information – bad things will happen after release – but it's appropriate to the needs of the project manager.

The purpose of testing is to give good information to help the release decision. It's to let project managers know what will happen if they let the product go [Marick98c].[1]

The same applies when a tester helps a developer who needs to know whether it's safe to add new or changed code to the master source base [Marick98a]. Because that case is less common than the previous, we will assume for concreteness that the project manager is the customer of the test suite.

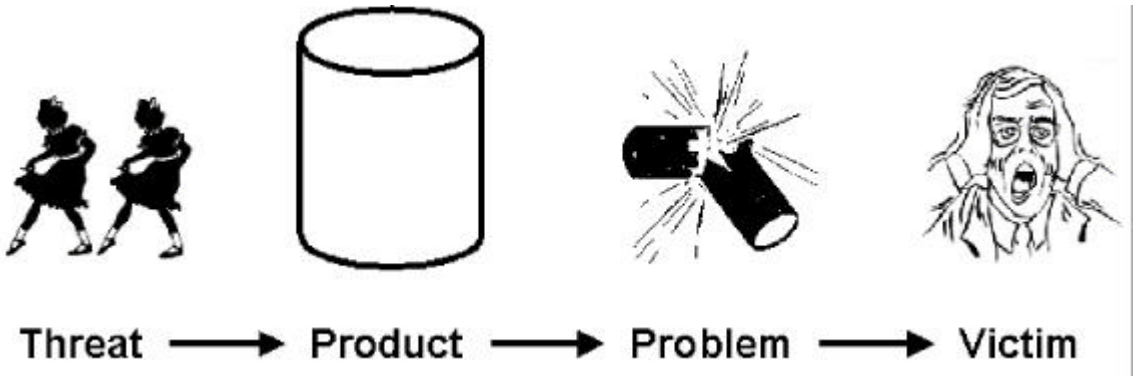
A test suite that provides information that the project manager already knows, and leaves unanswered questions she has, may be technically proficient, but it is not a good one.

If you are evaluating a test suite, you must therefore first evaluate the test suite creator's specific understanding of what the project manager needs to know. This is a matter of risk analysis. In this paper, we will analyze risk according to four factors: Product, Threat, Problem, and Victim. See also [Bach99a].

[1] Kaner is less sure of this assertion. Certainly, *sometimes* the purpose of testing is to help the project manager make a tough decision. But sometimes the project manager isn't looking to the testing group for the primary ship/no-ship release data. Instead, the company might just be hoping to find and fix the maximum number of defects in a fixed period of time. In that case, if the first group did enough testing of the five subsystems to realize that bugs were easy to find in subsystem 1 and hard to find in the others, then it followed the right strategy in focusing on subsystem 1. Alternatively, the company might have decided to ship the product at a certain date and be unwilling to change the decision *unless* the test group buries the company in open bug reports. In this case too, as long as the first subsystem is the richest source of bugs, the first group is acting appropriately.

Students of Kaner's have sometimes attacked this example as artificial. It is constructed to be simple in its presentation, and in that sense it is artificial. But every test group runs into the tradeoff that this example illustrates. Toward the end of the project, the test group is often under fundamentally conflicting pressures. They can operate in "assessment mode" and (like the second group) test broadly but find fewer bugs or they can operate in "bug-hunter mode" and (like the first group) maximize their bug finding rate at the expense of their assessment. Either role might be appropriate.

The following picture illustrates this style of risk analysis. What does it mean to say that a product has a bug? First, the product must be presented with some *Threat* – something in the input or environment that it cannot handle correctly. In this case, the threat is two cute but identical twins. (This corresponds to the common case of software being unable to handle duplicate inputs.) Next, the *Product* must perform its processing on that input or in the context of that environment. If it cannot handle the threat, it produces a *Problem* in the form of a bad result. But a problem means little or nothing unless there is some *Victim* who is damaged by it.



All four of these factors have to be considered in order to direct testing toward those bugs that are most likely to make it unwise to ship the product. As evaluators, we would ask questions probing whether the tester had considered them well.

For concreteness, let's suppose the tester was assigned to "black box" testing of the Tables feature in the Open Source word processor AbiWord (http://www.abiword.com). What might this tester's answers be?

Victim – who are the important classes of user?

The AbiWord developers consistently refer to their preferred user as "the church secretary" – someone less than a power user who has a strong preference for basic features that "just work". Therefore, a Tables test suite should concentrate more heavily on basic features at the expense of obscure ones. Everything on the Tables toolbar had better work in lots of circumstances. The subfeatures available through three levels of menu can fail much more often.

The AbiWord team's preferred user reminds us of the "target customer" in the marketing book *Crossing the Chasm* [Moore91] (chapter 4). There, Moore recommends creating 8 to 10 descriptions of users, giving them names, job descriptions, goals, and values. His reason?

> *Neither the names nor the descriptions of markets evoke any memorable images – they do not elicit the cooperation of one's intuitive faculties. We need to work with something that gives more clues about how to proceed. We need something that feels a lot more like real people. However, since we do not have real live customers as yet, we are just going to have to make them up. Then, once we have their images in mind, we can let them guide us to developing a truly responsive approach to their needs. (pp. 94-95)*

The same effect – engagement of the imagination – applies to testers as well as to marketers. We recommend Moore's approach.

Problem – what types of failure matter most?

Crashes would be bad for a church secretary, especially if they left documents unreadable by AbiWord. A failure to import Word tables is likely to be harmful – church secretaries exchange documents by email, so our AbiWord user will be often be using it to read Word attachments. When secretaries exchange documents, they probably cut and paste pieces from one document to another, so integration of tables with cut-and-paste code should work well.

Church secretaries are likely to be working with old equipment, so any performance problems will be amplified.

Has the tester looked at many church documents? Is there commonality across different types of churches, synagogues, mosques, and temples? If so, failures to handle commonalities – such as a particular preferred typeface – would be important problems to this class of victims.

Undo should work well with tables, because a non-expert user is likely to become confused if the actions that Undo undoes don't correspond well to the actions she thought she performed (as is likely if Tables are implemented as an afterthought, using underlying paragraph mechanisms [2]).

Finally, the testers should pay attention to the usability of the software. There is no usability team on this project, and the developers are power users, entirely unrepresentative of the canonical users. In order to properly prioritize their work, testers must develop an understanding of how the program is used. They should apply that knowledge to an evaluation of the program's design.

As these examples suggest, it can be hard to be sure you've found the important classes of ways in which the product might fail. Catalogs of failures, such as Appendix A in [Kaner93], can help.

[2] This is not the case with AbiWord, which doesn't even have tables as we write.

Threat – what kinds of failure-producing situations is the product likely to encounter?

One threat that an e-commerce application **must** contend with is activity from multiple simultaneous users. But testers of AbiWord needn't worry about that. However, AbiWord is highly cross-platform. It runs on Windows, BeOS, and Linux – indeed, this is one of its claims to fame. So a good test suite had better handle configuration issues well.

Product – what areas in the product are the shakiest?

Every product has certain things that are hard to get – and keep – right. As of late 1999, it seemed that printing might be one of those things for AbiWord (though it was really too early to say). More generally, it seemed difficult to get all layouts of the document looking right in all circumstances. In contrast, early users didn't seem to complain about undo bugs.

Step 2: What is the suite's approach?

A fine risk analysis doesn't assure a good test suite. Is the test suite's approach appropriate to the risks?

As an evaluator, we'd look for a written or oral description that went something like this:

> "Because the church secretary is so important, it seems we should favor a use-case (task-based) style of tests over strict functional tests.[3]
>
> "Further, the risk analysis affects our automation strategy. We've identified that usability is important. Moreover, some of the most persistent (but perhaps lower-impact) bugs are printer-layout or screen-layout bugs. Those argue for less automation than we might otherwise use. We need real humans executing real church secretary tasks and watching for usability problems. They should also watch for the kinds of layout problems that are notoriously difficult to detect with automation tools.
>
> "However, we have the problem of configuration tests, which are hard to do manually. It's bad enough to run a configuration test once, much less over and over again on different platforms. We have identified a subset of the test suite that can be automated and will be particularly sensitive to configuration dependencies. We believe it's sensitive because..."

The actual approach should be compared to a checklist of common practices, such as the following:

- "Stroke tests" are tests in which a tester checks every claim of fact and example in the manual.

[3] By use-case tests, we mean tests that behave like a user. For example, one such test might build a table, add one shading feature to it, print the page, use undo to remove the shading, add text to the table, print the page again, etc. In contrast, strict functional tests concentrate on exploring the possibilities of one feature (like shading) at the expense of creating the kinds of tables that real users create, in the way that real users create them.

- "Staked application testing" [Kaner/Marick00] is an attempt to use the application under test to duplicate the results of a competing application. To do this for AbiWord, the tester should find some actual church secretaries, get some real church bulletins built with Word, and attempt to recreate them exactly with AbiWord. (This is a test of more than tables.)

At the moment, we know of no thorough checklist. One is slowly being built at www.testingcraft.com, partially based on Appendix A from *Testing Computer Software* [Kaner93]. It would be a good idea to build an explicit checklist for your company.

In evaluating any approach, it's important to look at two things:

- are important things being done?
- are unimportant things **not** being done?

You need to check that the planned tasks are proportionate to both the risks and the project resources [Kaner96a].

Step 3: How's the maintainability?

For those tests that are automated, do they avoid the automation trap: tests that are intended to be rerun, but are so sensitive to product change that the testers will spend all their time trying to keep old tests working, none of their time creating new tests? This is a difficult matter of test suite design, and it's easy to go astray [Kaner97]. (See also the papers surveyed in [Marick99a].)

Don't just ask whether the suite was designed to be maintainable. Ask what's actually happened as the product evolved. How much time is spent upgrading the suite?

Do manual tests use elaborate scripts? These cost a fortune to develop, and they are expensive to maintain. After their initial use, they are unlikely to find additional defects. If scripts are used, they should straddle the boundary between the repeatability of scripting and the maintainability and benevolent side effects of less detailed test documentation [Marick97].

Has the tester chosen a good balance between manual and automated tests? [Marick98b]

Step 4: What do the details tell you?

Spend a little time looking at the details of the tests. Can you find gaps in the overall approach, in the tester's understanding of the project, or in her raw testing skills? There are two areas to look at:

Threats. How well do the specific tests implement the test approach? What test inputs were chosen? Do they cover all things that could affect the processing the program does: user inputs, machine configurations, contents of databases, network timeouts, and the like? Does the tester seem weak on particular test design techniques?

Result checking. A threat that provokes a problem does no one any good if the problem is overlooked. Here are some examples of things to check:

- Suppose the tester is doing purely exploratory testing ([Bach99c], [Pettichord99]). Does she only look for obvious problems ("the program crashed"), or does she notice small oddities like oddly slow operations? Does she then attempt to escalate oddities into major failures?
- Suppose the product provides a GUI front end to a bank. Does a test that removes €100 from an account with €100 in it only check that the ending balance on the screen is €0? Or does it also check whether the backend database has been updated? More generally, does the test check whether everything that should be affected has been?
- Suppose a test uninstalls a product. Does it check that the action leaves alone those things that should be unaffected? For example, does it check that components shared with other products are not removed? Does it check that all user data is left on disk?
- As noted above, automated tests need to be insulated from irrelevant changes to the product's interface. This is done by selectively filtering out information. Sometimes that leads to an inadvertent side effect: the tests become insulated from bugs as well. [Bach96] notes an extreme example: 3000 tests that always returned success, no matter what the product under test did! When reviewing automated tests, ask what kinds of bugs they might miss.

Step 5: What does the suite not do?

We'll first discuss code coverage. Code coverage tools add instrumentation to code. When the test suite executes, the instrumentation records how it exercises the program.

Our experience with such tools has been mixed. Some coverage tools are of poor quality. The most common problems in the weaker tools are that they are difficult to integrate into the build process, they fail to instrument some valid code that the compiler accepts (breaking the build), and they slow down execution to a ridiculous extent. Less common – but worse – is instrumentation that introduces a bug into the program. Their management of data is often crude. They typically assume a fully-automated test suite that is rerun against every build of the product, so are not good at merging data from several builds. The better tools are less prone to these problems.

All that having been said, we believe that code coverage can be a useful, but usually minor, part of test suite evaluation. We would use code coverage in two ways. The first is a quick look for major omissions; the other is a detailed look at critical code.

First, let's describe the major types of coverage that are measured by these tools.

Line coverage
: This type measures which code statements were executed.

Branch coverage
: Branch coverage measures which branches in the code were executed in which directions. It's possible to execute all statements without executing all branches, so branch coverage is stricter than line coverage.

Multicondition coverage

This type measures whether each Boolean condition in a branch has been evaluated to both true and false. It's a stricter criterion than branch coverage.

Multicondition is the best of these types, but line coverage is sufficient for our first purpose.

We would have a coverage tool summarize coverage at a fairly high level of granularity. Since AbiWord is C++ code, we would probably use the class level, which roughly corresponds to file-by-file summaries. We would look at those classes which had low or no coverage, and we would ask "why not?". Was it an intentional omission in the test (probably due to lack of time), or a mistake?

Next, if the importance of the code under test warranted, we would look at a sample of missed lines, branches, or conditions. (We would also like to use coverage that checks whether a loop has been executed zero, one, and many times, and whether boundaries have been checked. But the only coverage tool that we know measures those [Marick92] doesn't work on C++.)

Detailed code coverage is valuable when omitted coverage can be related back to the test design process to yield useful insights [Marick99b]. For example, any branch that has never been taken corresponds to some expected condition in the product's input or environment that has never occurred. Why not? Is it a natural consequence of the test approach? (Perhaps it corresponds to a test that couldn't be run because of lack of time.) Is it a mistake in test implementation? Is it a test design oversight? Is it symptomatic of pervasive oversights – for example, has error handling been systematically undertested?

Note that code coverage is less useful for some kinds of testing than others. For example, knowing that a line of code has been executed does not tell you whether it's been executed in a realistic user task. So coverage sheds less light on use-case style testing than on functional testing.

Code coverage is not the only type of coverage. Any systematic and tractable method of enumerating the ways a test suite exercises the code can be used in the same way. For example, Marick learned a style of documentation coverage from Keith Stobie. You begin with a highlighter and user documentation. You highlight every sentence in the documentation that has a test. At the end, there should be a justification for each un-highlighted sentence. ("Other things were more important" can be a good justification.)

[Kaner96b] gives a list of many other types of coverage.

Step 6: What has the suite missed?

Any test suite will miss some bugs. They might be discovered by other project members, by beta testers, or (after release) by end users. We think it is a good practice to examine those bug reports and ask questions. What do they tell you about how well the test approach has been implemented? Do they suggest flaws in the approach?

It's best to look at bugs that have been fixed. You can understand not only the observed failure, but also something of the underlying cause.

In a conversation on this topic on the SWTEST-DISCUSS mailing list (swtest-discuss-request@rstcorp.com), Jon Hagar warned that you should be careful not to overreact. Some bugs just get missed. Frantically changing plans in response to every serious missed bugs will lead to unproductive thrashing. We echo his comments.

Summary

This evaluation approach is highly subjective. In particular, it depends on the skill and experience of the evaluator. But an evaluator who is less good than the creator of the test suite can still produce value **if** she has good skills at human communication and cooperation, and good overall judgment. The worst thing is an evaluator who - like many drawn to evaluation - combines a lack of skill and experience with a dogmatic lack of knowledge of her lack of knowledge.

Although subjective, we think this method is preferable to an overly narrow objective one. Like it or not, objective rules are easy to misuse except in extremely well understood situations, such as counter work at a fast food restaurant. Ironically, we would only trust objective testing criteria in the hands of someone we had subjectively decided knows better than we do.

This approach is also expensive, if done in full. But it needn't be. The effort spent in evaluation should be tailored to the risk posed by a weak test suite. If the risk is low, a quick spot-check would suffice: an hour or so of interviewing about the test suite's purpose and approach, a brief check of maintainability, a small sample of bug reports and coverage.

What's missing?

This paper is incomplete in an important way. It assumes that evaluation is a "point activity". You as evaluator swoop in, take a snapshot of the knowledge of the tester and test suite, make your judgment, then ride off into the sunset.

Real projects don't work that way. At least, they shouldn't. A project begins with radically incomplete and incorrect answers to <u>all</u> of the questions we asked in this paper. Over time, the answers get better. For example, a project's idea of the canonical user will change.

If you evaluate once, when should that be? At the end of the project, when you can make the most accurate judgment? – but when it's too late to affect anything? Or early, when you can make a difference, but perhaps a difference based on bad data?

The answer, of course, is that you shouldn't evaluate once. The evaluation process should be continuous throughout the project. These questions should be constantly asked, to constantly refine your understanding of what a good (appropriate) test suite for this project would be.

Moreover, any project is in effect a conversation among product members (and, indeed, among all stakeholders[4]). By asking these questions, and others, you not only learn the answers, you also change them. That's an important service. For more, see [Bach97] and [Bach99b].

Appendix A: Approaches we reject

Error seeding and mutation testing are sometimes advocated as ways of evaluating a test suite. Here we explain why we reject them.

Background

A bug report describes what the tester believes to be a *failure* in the program. A failure is a difference between what a program did in some execution and what it should have done. The question of whether a particular result is really a failure is sometimes cause for lively debate.

A failure is caused by an underlying *fault*. In a program, a fault can be defined as the code that's changed to resolve the bug report. A single fault may cause multiple failures, and it may not be possible to tell that two failures are caused by the same fault until the fault is discovered.

Since there may be several reasonable fixes for a bug, a single failure might be said to be caused by multiple faults. What the fault *really* is can also be cause for great debate.

Specifications and requirements documents may also contain faults. If so, they describe or require results that are wrong from the perspective of a reasonable user. It is quite possible for the code to be right and the requirements document to be wrong. The fault in such an "upstream" document can be defined as the text that's changed to resolve a bug report. In the common case where the documents don't exist or are not updated, the fault is whatever you'd write down if you wrote down those kinds of things.

Error seeding

Let's define a *perfectly effective test suite* as one that reveals at least one failure for every fault in the program.

Given the background, this definition has some immediate problems:

1. What constitutes a failure is debatable.
2. The set of faults is debatable.
3. Some faults are not in the program.

[4] See *The Cluetrain Manifesto* [Locke00]: "A powerful global conversation has begun. Through the Internet, people are discovering and inventing new ways to share relevant knowledge with blinding speed. As a direct result, markets are getting smarter—and getting smarter faster than most companies… These markets are conversations… "

However, these are secondary issues. The fundamental question is how you would measure how close a test suite comes to perfection. You cannot simply say:

Effectiveness = number of faults found by suite / number of faults present

You don't know – and cannot know – the number of faults present in the code.

You could estimate *number of faults present* by adding the number of faults found in, say, the first six months after release to *number of faults found by suite*. This estimate of effectiveness comes a bit late. We'd like an estimate that would let us do something useful before release.

One alternative estimate is *error seeding* [Mills70]. In our terminology, it would be called "fault seeding". The process is this:

1. Seed, or inject, *N* representative faults into the program. For example, one seeded fault might be the removal of code that checks whether the network has gone down.
2. Test. You'll find some fraction, *some-of-N*, of the *N* seeded faults.
3. You probably found roughly the same proportion of the unseeded "native" faults. The estimated effectiveness is *(some-of-N / N)*.

The problem is that it is surprisingly difficult to inject representative faults. We lack a useable taxonomy of the different kinds of faults, knowledge of their relative chances of occurring, and an understanding of how the placement of faults in a program affects their visibility as failures.[5] If we seed the wrong proportions of the wrong kinds of faults in the wrong places in the program, our estimate may be wildly off.

We don't deny that fault seeding can provide some insight into a test suite. But does it provide more than coverage does? Statement coverage tells you that any fault seeded into an uncovered line will not be caught. Missed multicondition coverage tells you that a test suite could not have caught certain Boolean operator faults. Generally, each type of coverage can be viewed as targeting a particular class of seeded fault [Marick95] without actually doing the seeding. To accept fault seeding, we require experimental evidence that seeding additional classes of faults provides additional value.

Mutation testing

A variant of fault seeding is called *mutation testing* [DeMillo78]. It attempts to avoid the problems just cited through brute force and one key assumption. Mutation testing makes use of *one token faults*. A one token fault is, for example, changing a "+" to a " -", a "<" to a "<=", or the variable "i" to the variable "j". These faults are injected everywhere they can be, each one in a new copy of the program. This avoids the issue of deciding where to place faults and whether bad placement affects visibility.

[5] See, however, Voas's testability models for work on understanding the placement of faults. [Voas92] is an introduction. See also papers at the Reliable Software Technologies web site (www.rstcorp.com).

It's still the case that not all faults in programs are one-token faults. For example, the fault of omitting a network error check is a multiple-token fault. Mutation testing's *coupling assumption* asserts that a test suite adequate to find all one-token faults is also adequate to find all (or almost all) faults of all kinds. That given, it would perhaps be reasonable to say that a test suite adequate to finding 50% of one-token faults will probably also find 50% of all faults.

Mutation testing has problems that we believe makes it unsuitable for commercial use.

The coupling assumption is one that must be tested by experiment. To our knowledge, that has not been done, except for the simple case of pairs of one token faults [Offutt92]. We suspect that the coupling assumption is not true. A key problem is *faults of omission*, which are faults fixed by adding code. For example, a program that fails to check for network outages is fixed by adding code to do the check. Mutation testing – which concentrates on checking for variant implementations of code that's present – would seem to have little leverage for code that's not present. Since faults of omission represent a large fraction of faults that escape test suites (see [Marick00] for some numbers), that lack of leverage is worrisome.

Still, our approach in this paper was to recommend a collection of approaches, each making up for some of the weaknesses of the rest. Why shouldn't mutation testing be included in that collection?

Mutation testing requires tools to inject the faults. As far as we know, no commercially relevant mutation testing tools are available. One of us wrote a tool that provides a variant of mutation testing for C programs [Marick92], but it is available only for C programs and works only on old versions of Unix. Moreover, his experience with that tool [Marick91] left him unconvinced that mutation testing provides value over code coverage. Examining undiscovered mutation faults did not lead him to discover omissions in his test suite. Experiments or case studies applying mutation testing to large-scale software over an extended period of time might change his mind.

Acknowledgements

Boris Beizer prompted this paper by asking this question in an exchange with Marick on the SWTEST-DISCUSS mailing list (swtest-discuss-request@rstcorp.com):

> ASSERT: A good tester is one that produces a good test suite -- a suite that has a high probability of finding bugs if there are any to find. [...]
>
> QUESTION: What objective method do you propose to use to measure the effectiveness of a test suite?

As you've seen, we have not provided an objective method (that is, one that would yield the same results no matter who applied it). We do not believe that a useful objective method is possible. Still, it was a good and fair question.

References

[Bach96]
James Bach, "Test Automation Snake Oil,", *Windows Tech Journal*, October 1996. http://www.satisfice.com/articles/test_automation_snake_oil.pdf.

[Bach97]
James Bach, "Good Enough Quality: Beyond the Buzzword" (Software Realities column), *IEEE Computer*, August 1997. http://www.satisfice.com/articles/good_enough_quality.pdf

[Bach99a]
James Bach, "Risk-based Testing", *Software Testing and Quality Engineering Magazine*, Vol. 1, No. 6, November/December 1999. http://www.stqemagazine.com/featured.asp?stamp=1129125440

[Bach99b]
James Bach, "What Software Reality is Really About" (Software Realities column), *IEEE Computer*, December 1999. http://www.satisfice.com/articles/software_reality.pdf

[Bach99c]
James Bach, "General Functionality and Stability Test Procedure for *Certified for Microsoft Windows* Logo". (A description of an exploratory testing procedure.) http://www.satisfice.com/tools/procedure.pdf

[DeMillo78]
R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *IEEE Computer*, Vol 11, no. 4, April 1978.

[Kaner93]
Cem Kaner, Jack Falk, and Hung Quoc Nguyen, *Testing Computer Software*, 2nd edition, 1993.

[Kaner96a]
Cem Kaner, "Negotiating Testing Resources: A Collaborative Approach," a position paper for the panel session on "How to Save Time and Money in Testing", in *Proceedings of the Ninth International Quality Week* (Software Research, San Francisco, CA), 1996. (http://www.kaner.com/negotiate.htm)

[Kaner96b]
Cem Kaner, "Software Negligence and Testing Coverage," *STAR 96 Proceedings*, May 1996. (http://www.kaner.com/coverage.htm)

[Kaner97]
Cem Kaner, "Improving the Maintainability of Automated Test Suites," in *Proceedings of the Tenth International Quality Week*, May 1997. http://www.kaner.com/lawst1.htm

[Kaner/Marick00]
Brian Marick wrote a description of Kaner's Staked Application Testing at this URL: http://www.testingcraft.com/staked-application-testing.html.

[Locke00]
Christopher Locke, Rick Levine, Doc Searls, and David Weinberger, *The*

Cluetrain Manifesto: The End of Business as Usual, Perseus, 2000. Much material at http://www.cluetrain.com.

[Lyu96]
Michael Lyu, *The Handbook of Software Reliability Engineering*, IEEE Computer Press, 1996.

[Marick91]
Brian Marick, "Experience with the Cost of Different Coverage Goals for Testing", *Pacific Northwest Software Quality Conference*, October 1991. http://www.testing.com/writings/experience.pdf.

[Marick92]
Brian Marick, The Generic Coverage Tool (GCT). Source and user documentation available at ftp://cs.uiuc.edu/pub/misc/testing/gct.1.4/. See also ftp://cs.uiuc.edu/pub/misc/testing/gct2.0Beta/. Brian Marick **does not support** this software any more.

[Marick95]
Brian Marick, *The Craft of Software Testing*, 1995.

[Marick97]
Brian Marick, "Classic Testing Mistakes", *STAR Conference*, May 1997. http://www.testing.com/writings/classic/mistakes.html.

[Marick98a]
Brian Marick, "Working Effectively With Developers", *STAR West Conference*, October, 1998. http://www.testing.com/writings/effective.pdf.

[Marick98b]
Brian Marick, "When Should a Test be Automated?" *Proceedings of International Quality Week*, May, 1998. http://www.testing.com/writings/automate.pdf.

[Marick98c]
Brian Marick, "The Testing Team's Motto". http://www.testing.com/writings/purpose-of-testing.htm.

[Marick99a]
Brian Marick, "Web Watch: Automating Testing", *Software Testing and Quality Engineering Magazine*, Vol. 1, Num. 5, Sep/Oct 1999. http://www.stqemagazine.com/webinfo_detail.asp?id=102

[Marick99b]
Brian Marick, "How to Misuse Code Coverage", *International Conference and Exposition on Testing Computer Software*, June 1999. http://www.testing.com/writings/coverage.pdf.

[Mills70]
Harlan Mills, "On the Statistical Validation of Computer Programs". Reprinted in Harlan Mills, *Software Productivity*, 1983.

[Moore91]
Geoffrey A. Moore, *Crossing the Chasm*, 1991.

[Offutt92]
A. Jefferson Offutt, "Investigations of the Software Testing Coupling Effect," *ACM Transactions on Software Engineering and Methodology*, Vol. 1, No. 1, 1992.

[Pettichord99]
Bret Pettichord, "An Exploratory Testing Workshop Report," July 1999. http://www.testingcraft.com/exploratory-pettichord.html

[Voas92]
J. Voas, "PIE: A Dynamic Failure-based Technique," *IEEE Transactions on Software Engineering*, August 1992.

QW2000 Vendor Technical Paper 7V2

Bruce Katz
(Rational Software)

Test Early, Test Often: Applying today's best testing practices and techniques to the full software development lifecycle.

BACK TO QW2000 PROGRAM

Key Points

- Overview of an iterative software development process incorporating the best practices for testing - testing early, testing often, testing throughout the entire software development life cycle.
- Examples of the type of tests and test technologies used throughout the development life cycle in support of the best practice - test early, test often.
- Guidelines for implementing the best practices of testing early and often.

About the Author

Bruce R. Katz is a Sr. Test Consultant and Process Engineer with Rational Software. He is responsible for researching and incorporating the best testing practices in the Rational Unified Process. Bruce has over fifteen years experience in software testing and is a Certified Quality Analyst (CQA), by the Quality Assurance Institute (QAI).

BACK TO QW2000 PROGRAM

Rational
the e-development company™

Test Early Test Often:

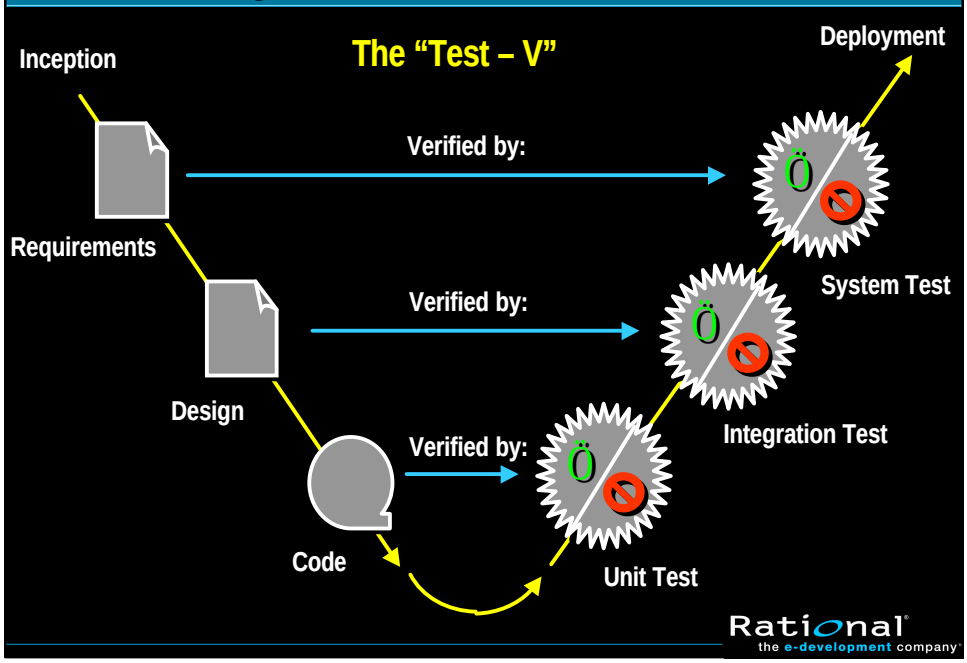
Applying today's best testing practices and techniques to the full software development lifecyle

Bruce R. Katz, CQA
Sr. Test Consultant, Process Engineer
Rational Software Corporation

Agenda

- Best Testing Practices: Past
- Best Testing Practices: Present and Future
- Sample Roadmap of Test Early, Test Often
- Summary and Q and A

Best Testing Practices: Past



Best Testing Practices: Past

Assumption	Reality	Challenge
◆Everything going well	◆Insufficient, timely, objective, data	◆Test earlier in lifecycle ◆Verify quality constantly
◆Testing is progressing and defects are being discovered	◆Requirements change ◆Not testing right things ◆Not testing properly	◆Increase test coverage ◆Test the right things ◆Test appropriately
◆Everyone working hard	◆Scrap & rework ◆No sharing or reuse	◆Maximize efficiency & effectiveness ◆Increase reuse
◆Requirements well specified	◆Requirements not well specified, understood, or interpreted	◆Improve management of requirements

Rational
the e-development company

Best Testing Practices: Present and Future

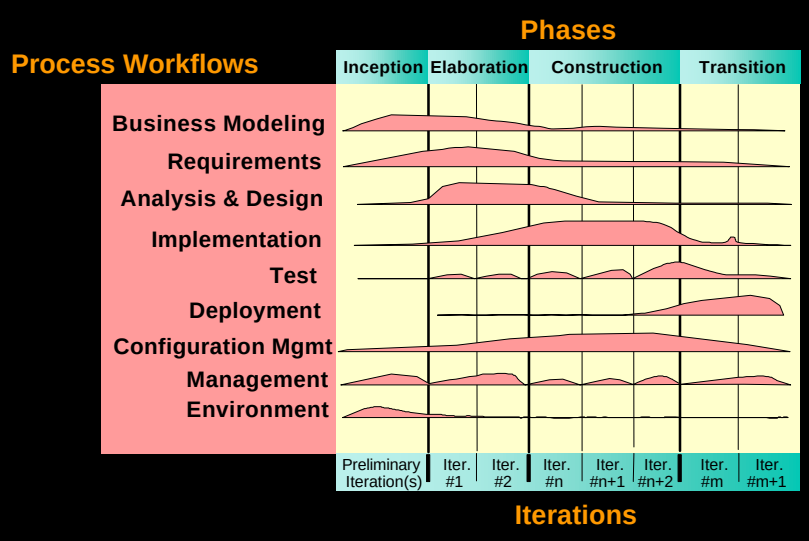
Test Early, Test Often (TETO)

Tenets:

- Test all Quality Dimensions
  - Reliability
  - Function
  - Performance
- Design for Testability
  - Design and implementation artifacts are used by all
  - Test built into / from the design and implementation artifacts
- Maximize Efficiency and Effectiveness
  - Managed, repeatable, well documented and communicated process
  - Automation
    - use tools and technology
    - implement throughout the lifecycle

Rational® the e-development company™

Rational Unified Process

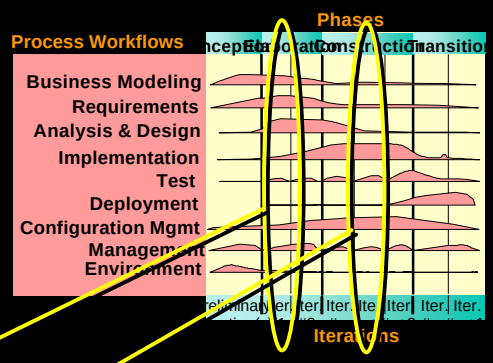


Rational® the e-development company™

Sample Roadmap

E-business application:

- Telephone Service Reps (TSRs)
- Transfer funds between two accounts
- Supplemental Requirements include:
  - Performance (sub-second)
  - Scalability (> 1,000 TSRs)



Architecture

Application

Construction

Rational
the e-development company

Elaboration: Architectural Iteration

Goal of Iteration: Architecturally significant risks

Example: Deployment Architecture

Obstacles / Challenges:

- Which architecture provides best performance and scalability?
- How to test?
- Is effort justified to create and test the test harness?

Solution:

- Rational Rose, Performance Architect, and PerformanceStudio

Rational
the e-development company

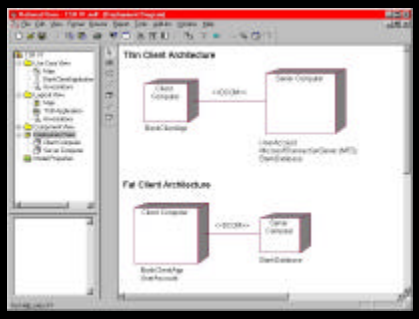
Elaboration: Architectural Iteration

Steps:

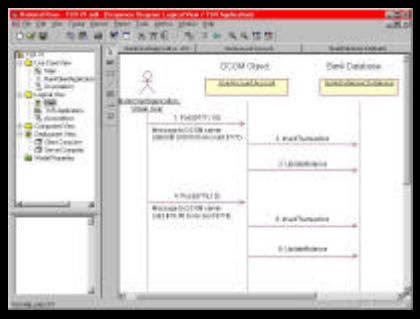
- Create Design and Implementation models
  Rational Rose
- Generate test scripts (automatically) from models
  Rational Performance Architect
- Modify, enhance, and create additional test artifacts
  Rational PerformanceStudio
- Instrument code for code-based test coverage (optional)
  Rational Purify, Quantify, PureCoverage
- Execute and Evaluate tests
  Rational PerformanceStudio, Purify, Quantify, and PureCoverage

Rational the e-development company

Elaboration: Architectural Iteration



Deployment Diagram



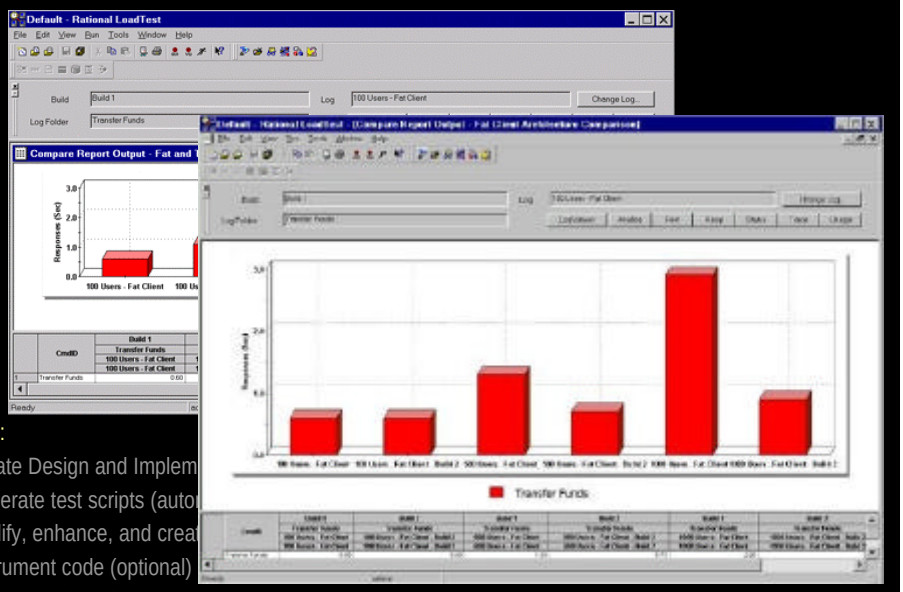
Sequence Diagram

Steps:

- Create Design and Implementation models
- Generate test scripts (automatically) from models
- Modify, enhance, and create additional test artifacts
- Instrument code (optional)
- Execute and Evaluate tests

Rational the e-development company

Elaboration: Architectural Iteration



Steps:

- Create Design and Implem
- Generate test scripts (auto
- Modify, enhance, and creat
- Instrument code (optional)
- Execute and Evaluate tests

Rational the e-development company

Elaboration: Architectural Iteration

Status Report:

TETO Tenet:	Actions:	Results:
◆Test all dimensions	◆Architectural assumptions tested	◆Performance & scalability issues addressed early
◆Design for testability	◆Test considerations included in models ◆Test scripts & test coverage based on models	◆Maximized test coverage ◆Provided objective test coverage
◆Maximize efficiency & effectiveness	◆Generated & executed test scripts automatically	◆Minimized effort ◆Maximized reuse

Rational the e-development company

Construction: Development Iteration

Goal of Iteration: Remaining product requirements addressed

Example: Developing component(s)

Obstacles / Challenges:

- What is the reliability of components / subsystems / application?
- How to test? In isolation? As integrated subsystems?
- Is effort to create and test reusable test harness justified?

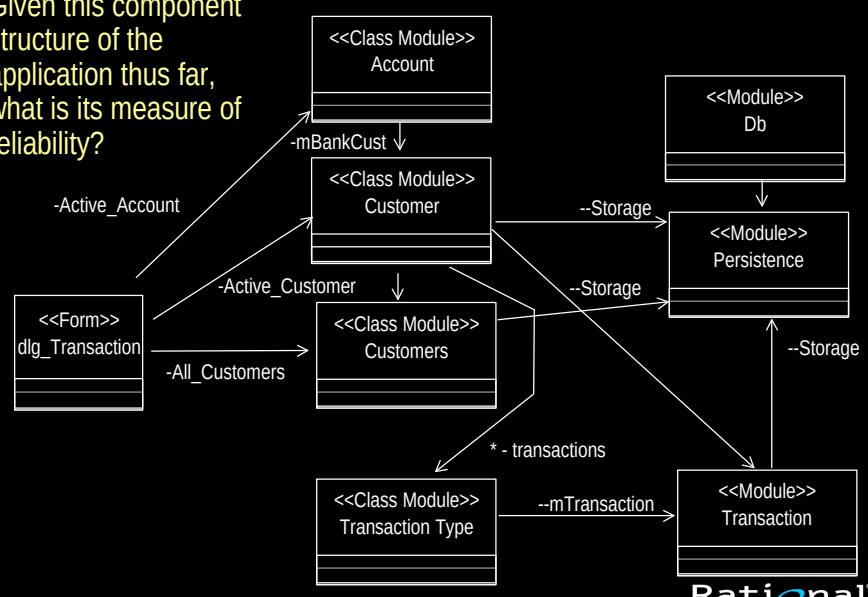
Solution:

- Rational Rose and Rational ComponentTest

Rational the e-development company

Construction: Development Iteration

Given this component structure of the application thus far, what is its measure of reliability?



Rational the e-development company

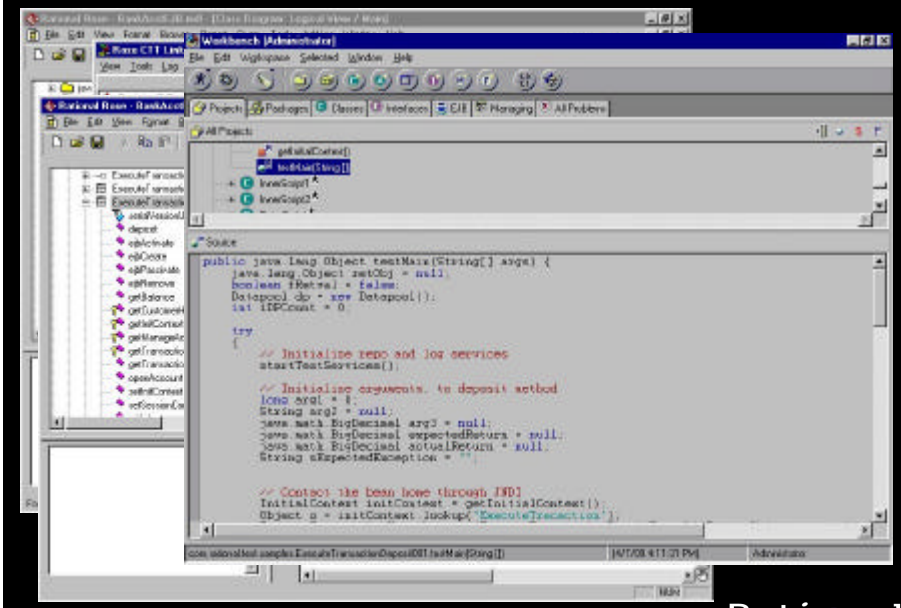
Construction: Development Iteration

Steps:

- Design and Implement components
  Rational Rose
- Instrument model (for model-based test coverage)
- Generate Test Drivers and Stubs
  Rational ComponentTest
- Modify / Enhance Test Drivers and Stubs
- Execute and Evaluate tests
  Rational ComponentTest

Rational
the e-development company

Construction: Development Iteration



Rational
the e-development company

Construction: Development Iteration

Status Report:

TETO Tenet:	Actions:	Results:
◆Test all dimensions	◆Tested Individual components ◆Tested integrated subsystems	◆Reliability & functional defects addressed early
◆Design for testability	◆Test considerations included in models ◆Test scripts & test coverage based on models	◆Maximized test coverage ◆Provided objective test coverage
◆Maximize efficiency & effectiveness	◆Generated & executed test scripts automatically	◆Minimized effort ◆Maximized reuse

Rational® the e-development company™

Construction: Development Iteration

Goal of Iteration: Remaining product requirements addressed

Example: Testing application via the application-user-interface

Obstacles / Challenges:

- How complete is testing? Navigation? Data entry?
- Can / how is testing automated?
- What impact does changing the UI have on test scripts?

Solution:

- Rational TestFactory

Rational® the e-development company™

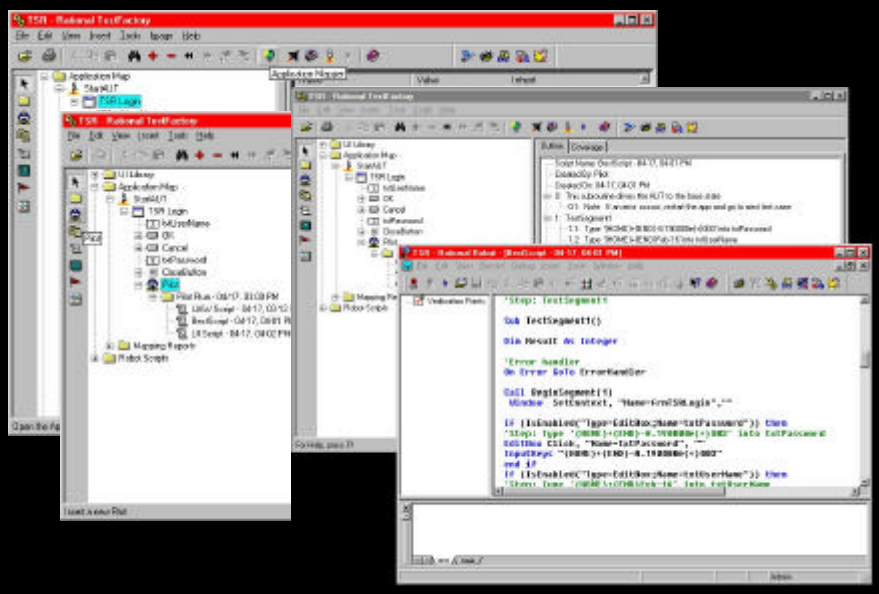
Construction: Development Iteration

Steps: (all performed using Rational TestFactory)

- Instrument Code (opt., code-based test coverage)
- Map Application User Interface
- Generate Test Scripts
- Execute and Evaluate Tests

Rational
the e-development company™

Construction: Development Iteration



Rational
the e-development company™

Construction: Development Iteration

Status Report:

TETO Tenet:	Actions:	Results:
◆Test all dimensions	◆Application navigation and input tested via application user interface	◆Reliability & functional defects addressed effectively
◆Design for testability	◆Test scripts & test coverage based on application user interface	◆Maximized test coverage ◆Provided objective test coverage
◆Maximize efficiency & effectiveness	◆Generated and executed test scripts automatically	◆Minimized effort ◆Maximized coverage & reuse, especially when UI changes

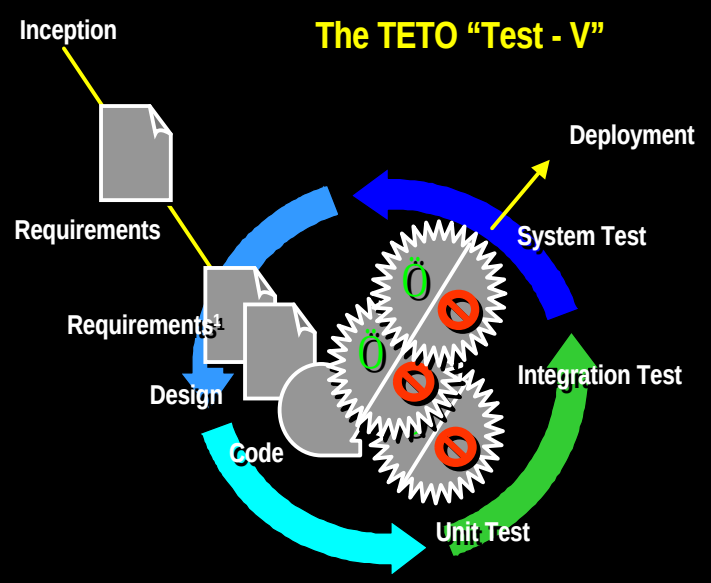
Rational® the e-development company™

Test Early, Test Often: Summary

Challenge	TETO
◆Test earlier in lifecycle	◆Enables testing as early as design
◆Increase test coverage ◆Test the right things ◆Test appropriately	◆Objective test coverage ◆Design and Implementation used as basis for test implementation and execution
◆Maximize efficiency & effectiveness ◆Increase reuse	◆Automation: ▪ Reduced manual effort ▪ Eliminated misinterpretation ▪ Enabled more complete and objective test coverage ▪ Reduced rework and modification
◆Improve management of requirements	◆Enabled earlier understanding and validation of requirements

Rational® the e-development company™

Test Early, Test Often: Summary



Rational
the e-development company™

Test Early Test Often:

Q and A

Bruce R. Katz, CQA
Sr. Test Consultant, Process Engineer
Rational Software Corporation

QW2000 Paper 7T2

Mr. Charles White
(Segue Software, Inc.)

Functional Testing of CORBA Based Systems in Java

BACK TO QW2000 PROGRAM

Key Points

- CORBA and EJB Objects in E-Business
- Testing for Success in Distributed Systems
- Understanding Middleware

Presentation Abstract

Three-tier applications are becoming the norm in the development world. The components in the middle-tier, often refered to as business objects, are frequently based on standard middleware such as CORBA or EJB. This paper discusses testing the middle-tier or business objects in a distributed application. There are two kinds of testing that can be done on business objects, structural and functional. Structural testing is internal testing of a single component. Functional testing is unit testing of one or more components. By the time a distributed, multi-tier application gets to the QA group, it is often no longer possible to do structural testing. However, with the rise of standard middleware, the testing of business objects can be done through the exposed business object interfaces. SilkPilot is a Java application which can do functional testing of the business objects in a distributed application. With support for testing both CORBA and EJB servers, SilkPilot can reduce your time to market by greatly accelerating your testing cycles.

This talk discusses

- Structural and functional testing on business objects at different stages of the product life-cycle
- How using standard middleware helps improve testability of business objects
- What CORBA and EJB are and how they are used.
- How SilkPilot works with CORBA and EJB to test business objects
- How SilkPilot can save time, protect your investments, improve productivity and increase reliability of depolyed applications.

About the Author

Charles White is the CTO for the Distributed Development Group at Segue Software. Charles was a founder of Black & White Software (since acquired by Segue), one of the first companies to venture

into the CORBA tools space.

Charles has worked architecting and implementing development tools for Unigraphics, Daisy Systems, Siemens-Nixdorf, Visual Edge Software, Black & White and Segue for the the past 20 years. Charles has published numerous technical papers, and has been a speaker at numerous conferences on user interfaces and distributed objects. Charles can be reached at chas@segue.com.

BACK TO QW2000 PROGRAM

QW2000
San Francisco
May 2000

segue

Functional Testing of CORBA and EJB Business Objects

Charles White
Segue Software, Inc.

chas@segue.com
http://www.segue.com

segue

Presentation Overview

- What is Functional Testing?
- The Challenge: Multi-tiered Applications
- Distribution in Heterogeneous Environments
- EJB in Heterogeneous Environments
- Functional Testing of CORBA & EJB Objects
- SilkPilot - A Generic Functional Testing Tool

2

How is Software Tested?

- Functional Testing (Black-box testing)
  - Goal: Prove that software conforms to its specification
  - Test scenarios derived from functional specification
- Structural Testing (White-box testing)
  - Goal: Try to find cases that will break the software because of conditions the programmer did not expect
  - Test scenarios derived by examining the source code

3

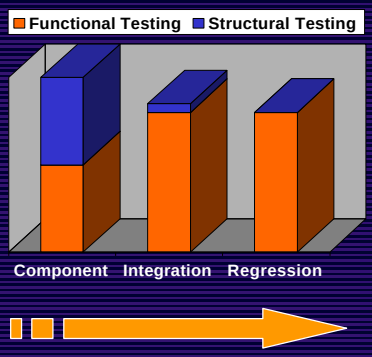
When is Software Tested?

- Component Testing
  - Goal: Find errors in new components
  - During development
- Integration Testing
  - Goal: Find errors in interactions and interfaces of new, untested modules
  - During development and maintenance
- Regression Testing
  - Goal: Re-establish the confidence that the software will continue to function after modifications
  - During maintenance

4

Structural vs. Functional Testing

- Importance and amount of required Structural Testing decreases
- Importance and amount of required Functional Testing increases



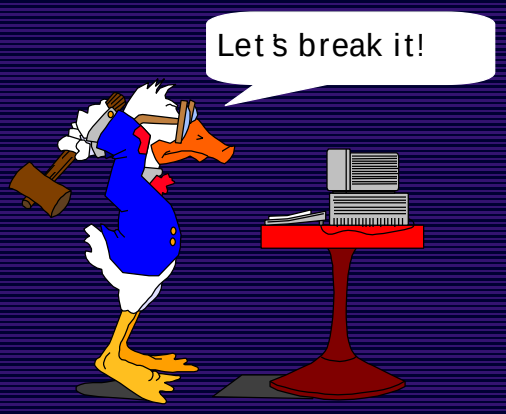
5

How to Test Non-Distributed Applications?

- UI level
  - Tester simulates the end user, tries to break application
  - Tool support for record/playback, e.g. SilkTest
  - Completely black-box, so origin of errors is usually unknown
- Object/Module level
  - Developer writes specialized test drivers
  - Objects/Modules need to be linked with the drivers
  - If modules do not have public interfaces, testing on this level is usually done during development by the developer only

6

Test Like This!



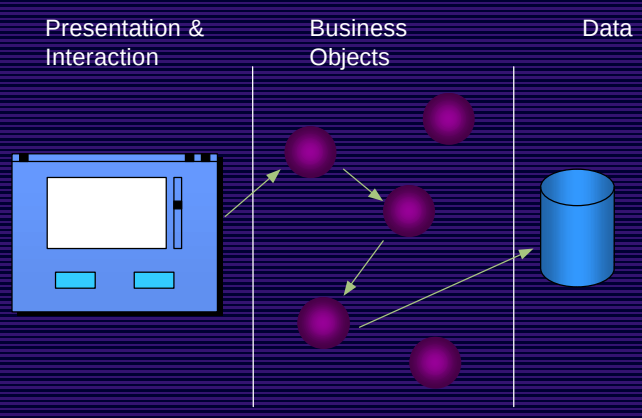
7

Presentation Overview

- What is Functional Testing?
- The Challenge: Multi-tiered Applications
- CORBA in Heterogeneous Environments
- EJB in Heterogeneous Environments
- Functional Testing of CORBA & EJB Objects
- SilkPilot - A Generic Functional Testing Tool

8

Multi-tiered Applications



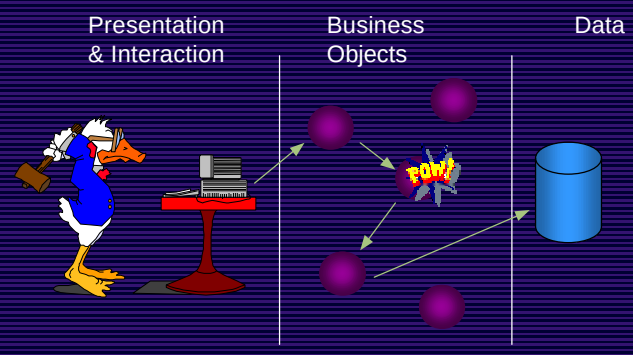
9

Business Objects

- Objects can be anywhere
- Clients have transparent access to server
- No language or platform restrictions
- Provide basis for delivering component-based applications

10

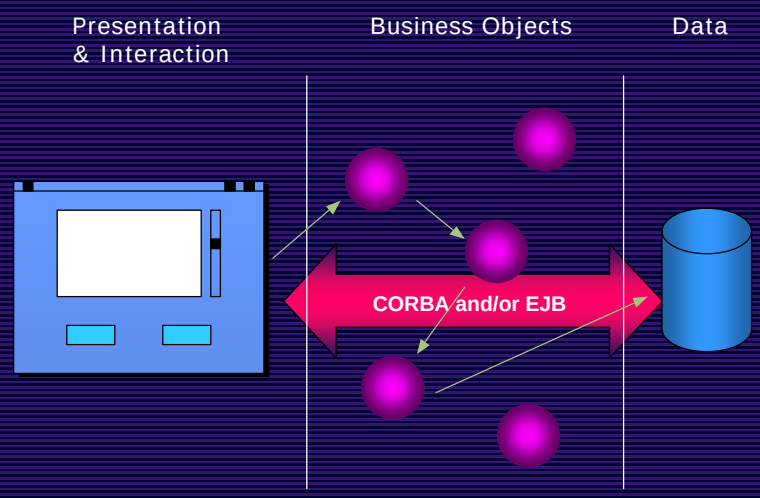
Testing Multi-tiered Applications



Complexity: Many system components
Coordination: Many participants in creation and deployment
Change: Rapid delivery cycles

11

Solution: Use Standard Middleware!



12

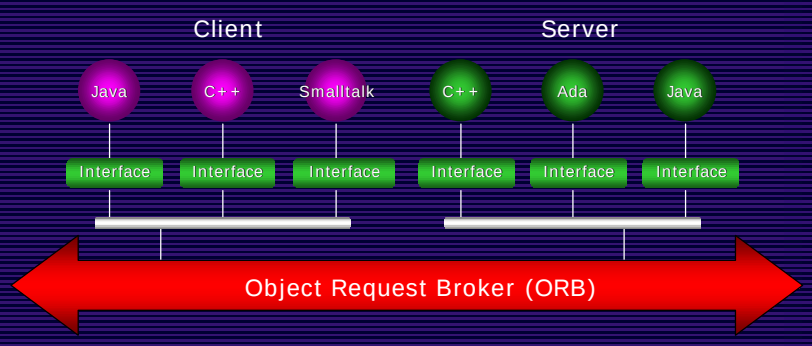
Presentation Overview

- What is Functional Testing?
- The Challenge: Multi-tiered Applications
- CORBA in Heterogeneous Environments
- EJB in Heterogeneous Environments
- Functional Testing of CORBA & EJB Objects
- SilkPilot - A Generic Functional Testing Tool

13

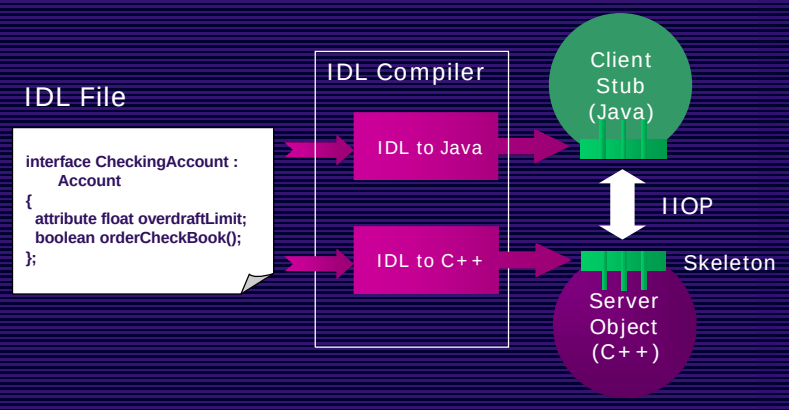
CORBA - What is it?

- Software bus for client-server object distribution in heterogeneous environments
- Object interfaces specified independent of implementation
- ORB mediates between clients and servers



14

IDL - Interface Definition Language



IIOP Protocol Ensures Interoperability

Internet Inter-ORB Protocol (IIOP):

- CORBA wire protocol
- Permits CORBA clients and servers to utilize different ORBs
- On top of TCP/IP

Presentation Overview

- What is Functional Testing?
- The Challenge: Multi-tiered Applications
- Object Distribution in Heterogeneous Environments
- EJB in Heterogeneous Environments
- Functional Testing of CORBA & EJB Objects
- SilkPilot - A Generic Functional Testing Tool

17

Enterprise Java Beans

- Reusable server components
- Beans reside in the EJB container
- Container communicates with the client on behalf of EJB
- Transaction/Security/Persistence are automatically provided by the container

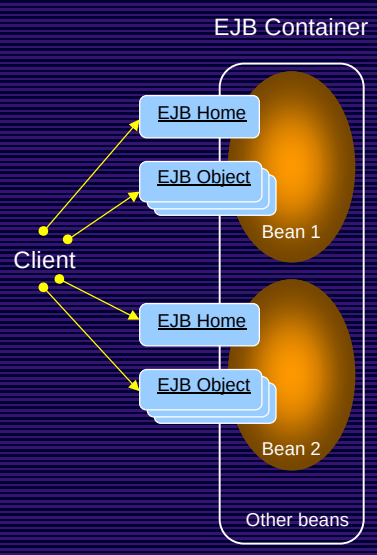
18

Enterprise Java Beans

- Session Beans
  - may or may not have a state
  - disappear when server crashes
  - new instance for each client
- Entity Beans
  - represent data in the database
  - survive server crashes
  - one instance for many clients
  - found by primary key

19

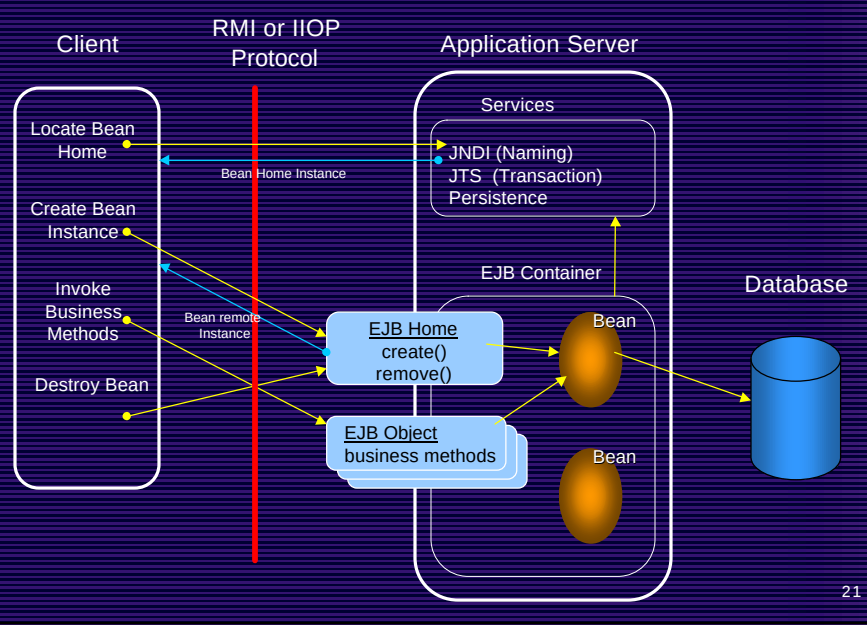
Client View of the bean



- Client can not see the bean
- Home interface - EJBHome
  - create methods
  - finder methods
  - remove methods
- Remote interface - EJBObject
  - business methods

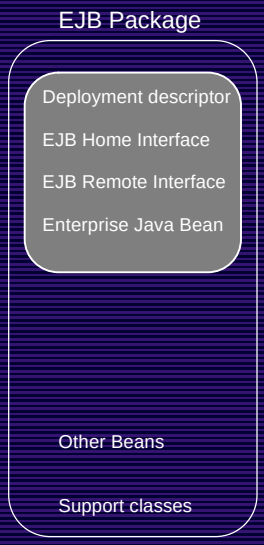
20

Communication Scenario



21

EJB Deployment in Container



- Deployment descriptor
  - EJB Classes
  - bean type
  - policies
- Home and remote interfaces
- Support classes

22

EJB Deployment in Container

- Container analyzes EJB package
  - read deployment descriptor
  - generate implementations for home and remote interfaces and client stubs
- Bean home object is registered with the naming service
- Home and remote interfaces are available to the client
- Wait for connection from the client

23

Container Contract

- How EJB container manipulates the bean?
- Bean implements methods required for its management
  - ejbCreate()
  - ejbFind()
  - some other methods
- Bean contains business logic methods
  - bean should not implement remote interface
  - container uses Java reflection to redirect calls to the bean

24

Common Misunderstanding

EJB, RMI and CORBA are not mutually exclusive

- RMI over IIOP allows transparent mapping of RMI to CORBA
- EJB client may use EJB API over IIOP or use CORBA mapping of EJB

25

EJB CORBA Clients

- EJB/CORBA Client
  - uses EJB Java API
  - JNDI to locate objects
  - RMI over IIOP to invoke methods
  - JTS for transactions
  - use of IDL is implicit to programmer
- Plain CORBA Client
  - written in any language that supports CORBA
  - COS Naming to locate objects
  - CORBA IDL to invoke methods
  - CORBA OTS for transactions
  - explicit use of IDL

26

Advantages of EJB

- Established roles for application development
- Automatic transaction management
- Distributed transaction support
- Portability
- Scalability
- Integration with CORBA
- Vendor enhancements

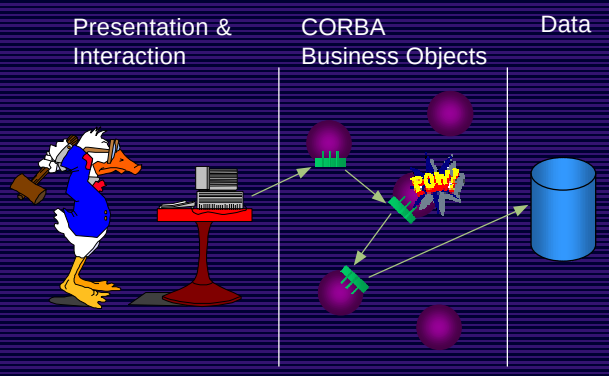
27

Presentation Overview

- What is Functional Testing?
- The Challenge: Multi-tiered Applications
- Object Distribution in Heterogeneous Environments
- EJB in Heterogeneous Environments
- Functional Testing of CORBA & EJB Objects
- SilkPilot - A Generic Functional Testing Tool

28

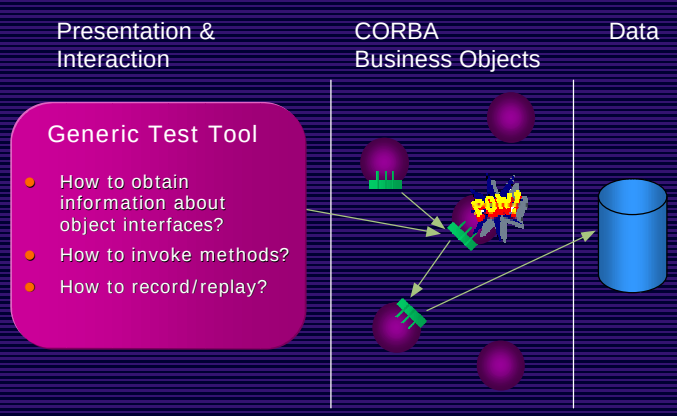
Testing Multi-tiered CORBA Applications



- Can we test at the object interface level, in the real world, independent from the GUI?
- YES!

29

Functional Testing of CORBA Objects

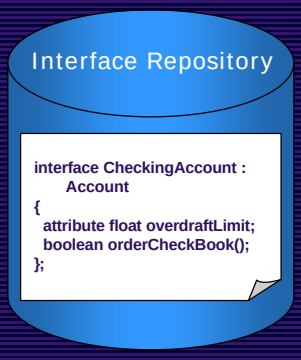


Generic Test Tool

- How to obtain information about object interfaces?
- How to invoke methods?
- How to record/replay?

30

Obtain Information about Object Interfaces



CORBA Interface Repository (IFR)

- Provides information about objects at runtime
- Programming Interface for Interface Definition Language (IDL)
- API defined by CORBA Specification
- Full READ/WRITE access to IDL
- Implemented as standalone CORBA server

31

How to Invoke Methods?

Dynamic Invocation Interface (DII)

- API defined by CORBA Specification
- "Lower level" interface
- Provides fully dynamic access to remote objects
- No compiled IDL stubs in client!

32

How to Record/Replay?

- Save CORBA IIOP request/replies using a suitable external data representation
- Test tool can interpret external data and replay a session
- External data can be used to generate standalone test clients

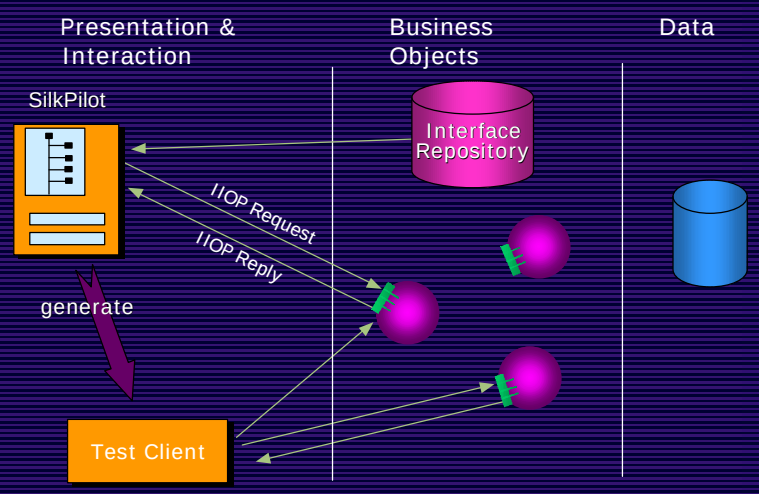
33

Presentation Overview

- What is Functional Testing?
- The Challenge: Multi-tiered Applications
- CORBA in Heterogeneous Environments
- EJB in Heterogeneous Environments
- Functional Testing of CORBA & EJB Objects
- SilkPilot - A Generic Functional Testing Tool

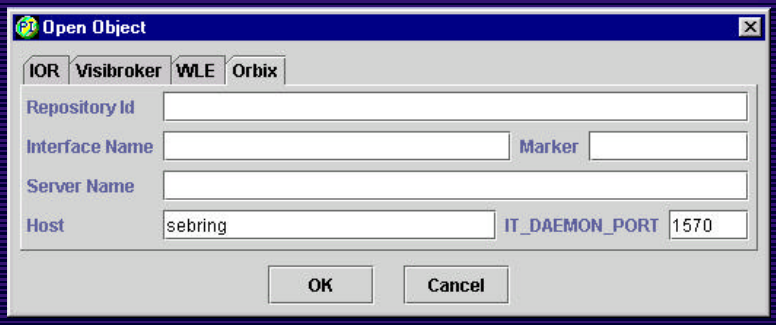
34

SilkPilot - A Generic CORBA Testing Tool



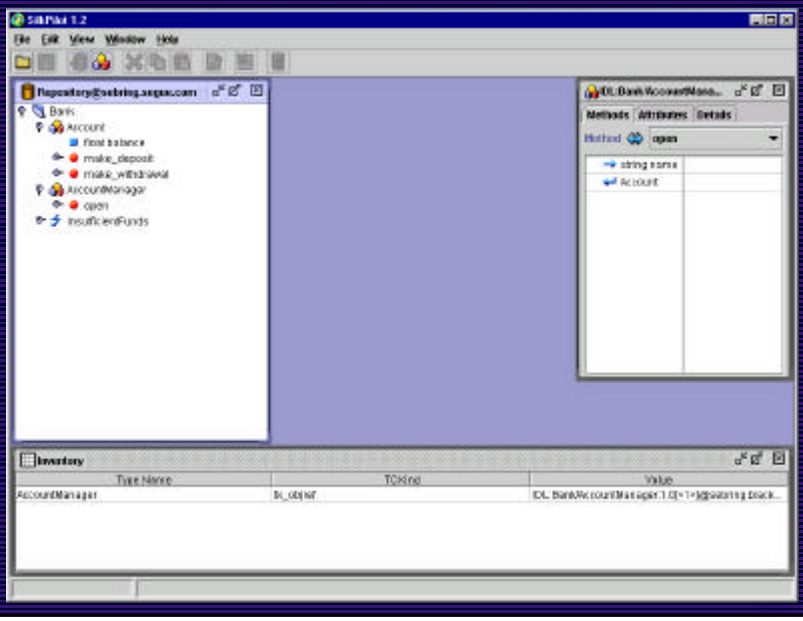
35

SilkPilot - Open Object



36

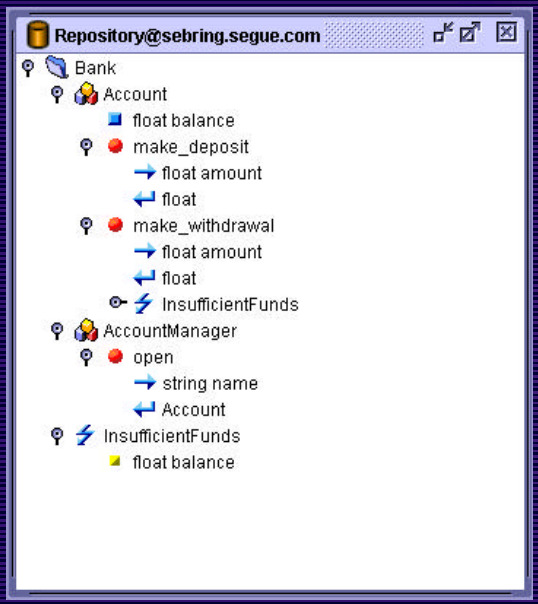
SilkPilot - Main Window



37

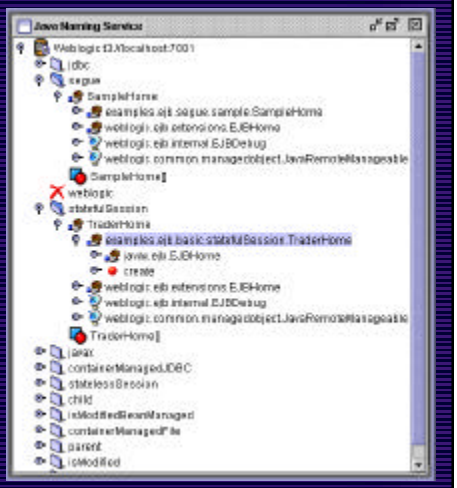
SilkPilot - IFR Browser

- CORBA IFR browser
  - Use built-in or ORB supplied IFR
  - visual representation of IDL objects

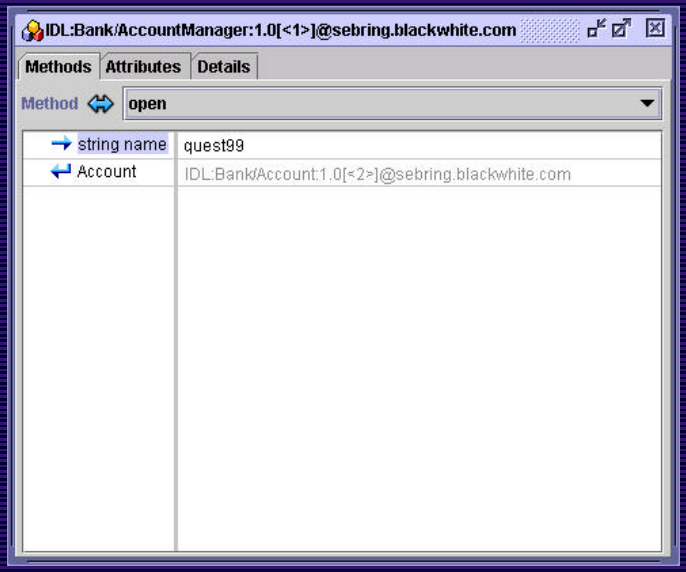


SilkPilot for EJB

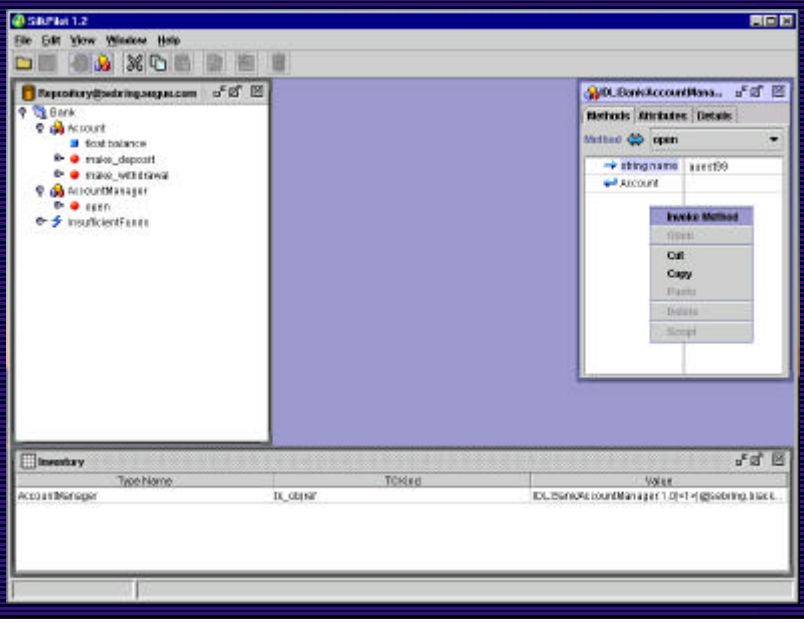
- JNDI browser
  - browse any JNDI compliant service
  - visual representation of EJB and RMI objects
  - narrow to any base interface



SilkPilot - Object Window

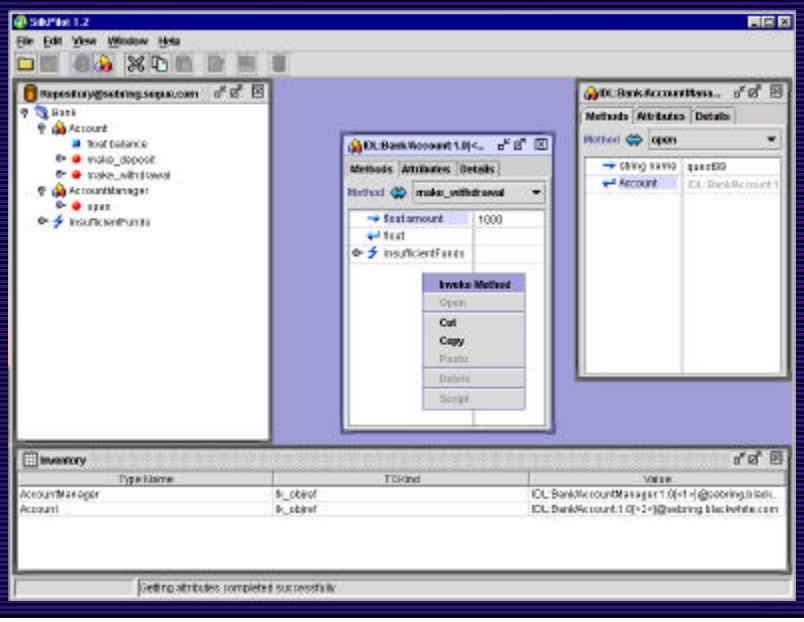


SilkPilot - Create Account



41

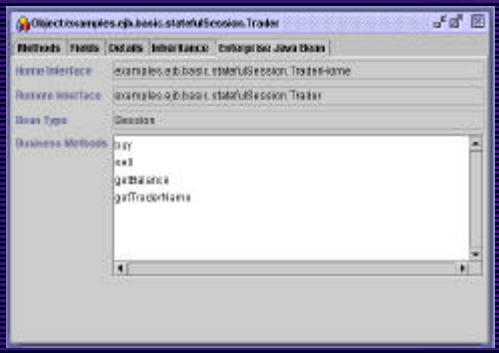
SilkPilot - Make Withdrawal (1)



42

SilkPilot for EJB

- EJB Metadata
  - home and remote interfaces
  - bean type
  - business methods



43

Test Client Code Generation

- Generates Java Client
- All steps done in SilkPilot can be put into code
- Generates Makefile to build the client
- Generates script to run the client after building it

44

Summary of Features

- Saves Time and Money
  - No need to manually build custom test programs
- Investment Protection
  - SilkPilot is built on standards - CORBA and EJB
- Improve Productivity
  - Intuitive graphical user interface
  - Generated client code can be reused
- Maximize successful development
  - Easily test all components within a distributed system

45

Thanks for Coming

< the end >

QW2000 Paper 7A2

Mr. Scott Trappe
(Reasoning Inc.)

Find the Defects that Traditional Testing Misses with Automated Software Inspection Services

BACK TO QW2000 PROGRAM

Key Points

- Automated software inspection -- new techniques that improve software quality
- Inpsection tools vs. services -- taking advantage of the oursourcing trend
- ROI for automated software inspection -- amazing returns for little effort, but what's the catch?

Presentation Abstract

Producing high quality software for increasingly complex applications is a constant challenge. Due to the constraints placed on osftware development schedules and budgets, traditional testing techniques have limited ability to catch all critical defects prior to software release. Automated Software Inspection Services are a new and largely untapped method of finding defect soften missed by tranditional testing. This session explores the potential of these methods a s a cost-effective and automated means of improving quality without taxing in-house resources.

About the Author

Scott Trappe is a senior VP of Operations for Reasoning, Inc. He is considered an expert in Automated Source Code Analysis methods, having contributed numerous articles on the subject to software development publications as well as having hosted presentations at industry related events. Previously he worked at Intrinsa, where he held the dual role of VP of Engineering and Marketing. Mr. Trappe has also worked at Netopia; Telekom in Malaysia; Operations Control System; and Tektronix. He holds an MBA from the University of Californig, Berkeley and a B.S. from the University of Arizona.

BACK TO QW2000 PROGRAM

Find Defects that Traditional Testing Misses with Automated Software Inspection Services

Scott Trappe
Senior Vice President & General Manager
Reasoning, Inc.

© 2000 Reasoning, Inc.

Problem: The Software Paradox

- Software is more complex and critical than ever -- but programmers have less time to develop and test it

➔ ***THIS IMPACTS SOFTWARE QUALITY***

- Software failure results in $85B revenue loss per year
- New e-commerce software needs continuous operation, failure means loss of business
- Need to deploy software solutions quickly
- Need more robustness; used by uneducated users

➔ ***THIS DEMANDS MORE SOFTWARE QUALITY***

© 2000 Reasoning, Inc.

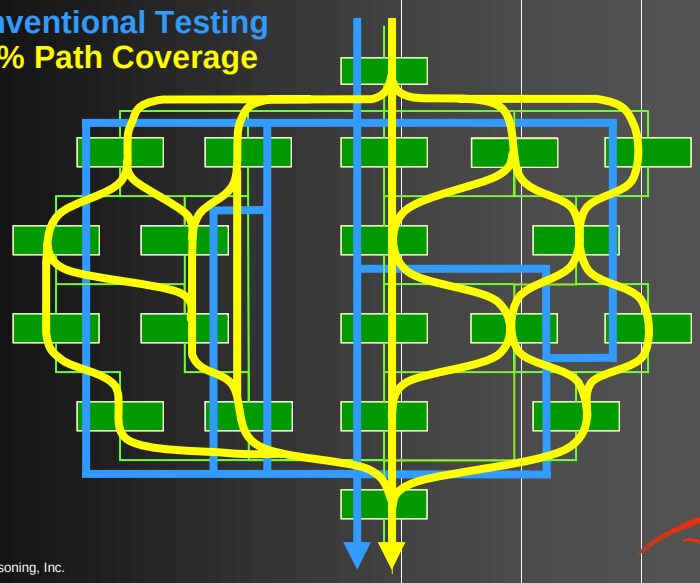
Testing Alone Is Not Enough

- Even the most rigorous testing performed by U.S. companies finds only 85% of all software defects.
  [*Capers Jones, 1997*]
- Most testing programs only test 30% to 40% of an application. [*The Standish Group, 1998*]
- "Testing is never finished, only abandoned"
  [*Encyclopedia of Software Engineering*]
- Keeping test scripts and test data current is difficult

© 2000 Reasoning, Inc.

Code Coverage vs. Path Coverage

Conventional Testing
100% Path Coverage

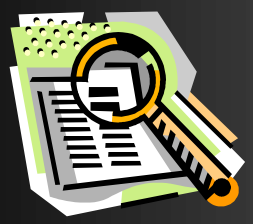


© 2000 Reasoning, Inc.

Solution: Software Inspection

"Software inspection is by far the single most effective technique for removing defects in software."

Capers Jones



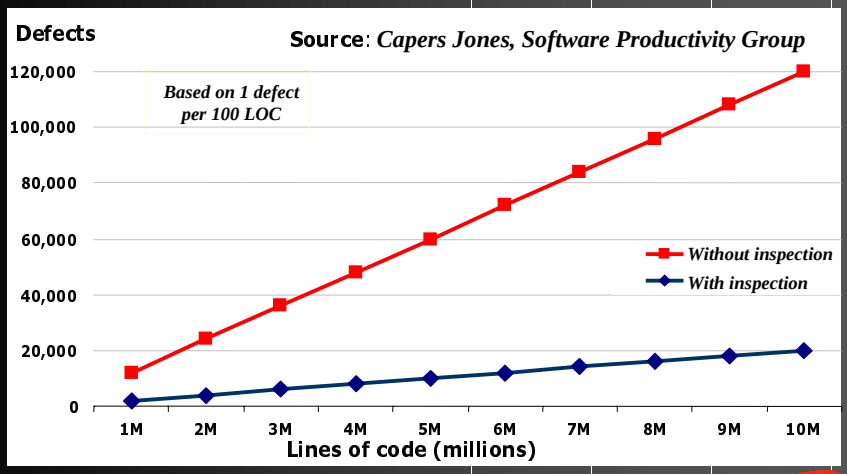
© 2000 Reasoning, Inc.

Software Inspection Overview

- Origin: Process for conducting Inspections developed in the 1970s by Michael Fagan of IBM
- Definition: Inspection is a form of *static* testing that analyzes source code to find programmatic errors
- Impact: Over half of all defects in released software applications can be found through Inspection
- Process: Developer or team of developers perform a "structured walk-through" of the source code. They look for structural errors that can lead to program failure

© 2000 Reasoning, Inc.

Software Inspections Find Serious Defects Testing Misses



© 2000 Reasoning, Inc.

Software Inspection Complements Testing

- Inspection plus testing can find 95% of all software defects [*Capers Jones, 1999*]
- Inspection identifies structural errors, allowing testers to focus on functional errors
- Inspection finds defects earlier and faster, helping to reduce development overruns that may affect testing schedules
- With 100% code coverage, Inspection can analyze code segments not covered by traditional testing methods

© 2000 Reasoning, Inc.

Software Inspection Reduces Costs

- Inspection finds up to 65% of all defects *prior to testing*, thus reducing test time and cost.[1]
- The cost of fixing a defect in testing is 10 times the cost of finding a defect prior to testing.[2]
- Inspections save 60% of rework costs.[3]

1. Capers Jones, Software Productivity Research, Inc.
2, 3. Robert B. Grady, *Successful Software Process Improvement,* Prentice Hall PTR, 1997.

9

© 2000 Reasoning, Inc.

Drawback: Manual Software Inspection Is Expensive

- Manual Inspection requires experienced software engineers
- Inspection is time-consuming
- No consistent, repeatable results
- Opportunity cost: time engineers spend doing inspections takes away from actual development

10

© 2000 Reasoning, Inc.

Alternative:
Automated Software Inspection

© 2000 Reasoning, Inc.

Automated Software Inspection (ASI) Advantages

- Quick feedback *during* the project
- No burden on engineering staff
- Finds defects missed by conventional testing
- Pinpoints where the defect is in the code
- Repeatable, consistent results
- No test cases required
- Non-intrusive
  - **No impact on size or performance of your software**
  - **Works on incomplete (non-running) code**
  - **No code changes required**

© 2000 Reasoning, Inc.

Example: Embedded Systems

- Embedded Systems are:
  - Difficult to Test
  - Difficult to Debug
  - Failures are Expensive/Catastrophic
- Automated Software Inspection:
  - No target environment required
  - Validates error-handling code
  - Does not affect real-time performance
  - Target-processor independent

© 2000 Reasoning, Inc.

One Company's Experience with ASI

.We also have a special advanced source code analyzer that we use… [in the development of Windows 2000]. We love this tool. It helps us improve our code for readability and it can find bugs that our testing may not find..

Jim Allchin
Group Vice President, Platforms Group
Microsoft Corporation

February 15, 2000

© 2000 Reasoning, Inc.

Product-based ASI has Disadvantages

- Costly time-commitment to evaluate, purchase, install, learn, configure, and maintain product
- High “false-positive” rate
- Requires development resource to run product and filter results
- Example: PC Lint

© 2000 Reasoning, Inc.

Better Approach: Automated Software Inspection Services

- Finds defects without taxing in-house resources
- Fast (results returned in under 1 week)
- Customer only sees Critical Defects, no “noise”
- Repeatable - as often as required
- Utilizes advanced technology not available as a product
- Example - InstantQA[sm] Services from Reasoning

© 2000 Reasoning, Inc.

Trend:
Market is Shifting to Outsourced Software Inspection

"Independent Source Code Verification (ISCV) will be a strong services growth market during the next five years and will become an established part of the IT services landscape." *

GartnerGroup INTERACTIVE

*Source: The GartnerGroup-ISCV: An Independent Service to Aid Software QA, June 29, 1999

17

© 2000 Reasoning, Inc.

Examples of Defects ASI Services can Find

C/C++	Cobol
Dynamic Memory Issues	Initialization
• Memory Leak	• Data Element Used but Not Set
• Bad Deallocation	Interface
Initialization	• Unchecked Return-Code
• Variable Always Uninitialized	• Parameter Mismatches
• Variable Conditionally Uninitialized	• CICS Length Mismatches
Pointer Management	Dead Code
• Null Pointer Dereference	Fragility
• Out of Bounds Array Read/Write	• Deeply Nested If Statements

18

© 2000 Reasoning, Inc.

Detailed Example - Initialization Error

```
int len;
 if ((to - from) > 86400) {
   now = localtime(&from);
   len = strftime(__internatl_OdateToStr, 64, "%m/%d/%Y - ", now);
   now = localtime(&to);
   len = strftime(&__internatl_OdateToStr[len], 64, "%m/%d/%Y", now);
 }
 else {
   now = localtime(&from);
   len = strftime(&__internatl_OdateToStr[len], 64, "%m/%d/%Y", now);
 }
```

Variable Declared

Variable Set

Variable Used
But Not Set

19

© 2000 Reasoning, Inc.

ASI Services Return on Investment

- Three Scenarios
  - ASI Service vs. Manual Inspection
  - ASI Service vs. Conventional Testing
  - Improving Defect Removal Rates
- Assumptions
  - Application is 500 KLOC
  - ASI Service finds 1 critical defect per KLOC: 500 defects
  - ASI Service cost: $.10/line × 500 KLOC = $50,000
  - Results returned within 2 weeks

20

© 2000 Reasoning, Inc.

ASI Service vs. Manual Inspection

- Assumptions:
  - Cost of software engineer is $10,000 per month
  - Software engineer works on average 200 hours/month
- Cost for Manual Inspection:
  - Engineer will require an average of 5 hours to find a defect[1]
  - 5 hrs ÷ 200 hrs/month × $10,000/month = $250 Defect
  - $250 × 500 defects = $125,000 and 12 months
- ROI: ASI Service finds the same number of defects:
  - for $75,000 less
  - in 2 weeks, not 12 months

1. [Watts Humphrey 1999, Software Engineering Institute]

© 2000 Reasoning, Inc.

ASI Services vs. Conventional Testing

- Assumptions:
  - Cost of QA engineer is $10,000 per month
  - QA engineer works on average 200 hours/month
- Cost for Conventional Testing:
  - QA Engineer will find an average of 8 defects/month[1]
  - $10,000/month ÷ 8 defects/month = $1,250/Defect
  - $1,250 × 500 defects = $625,000 and 5 man-years
- ROI: ASI Service finds the same number of defects:
  - for $575,000 less
  - in 2 weeks, not 5 man-years

1. [Gartner Group]

© 2000 Reasoning, Inc.

Improving Defect Removal Rate

- Assumptions
  - $15,000 avg. cost to find a defect after deployment[1]
  - 5,000 bugs introduced before any testing or inspection[2]
- "Best in class" Defect Removal Rate[2]: 85%
  - With a 15% miss rate (1 – 85%) x 5,000 = 750 bugs will make it to the field
- Assume ASI Service boosts DRR 5%: from 85% to 90%:
  - Eliminates 250 defects
  - Savings: 250 * $15,000 – $50,000 = $3,700,000

1 [The Cutter Consortium]
2 [Capers Jones]

© 2000 Reasoning, Inc.

The Worst Case

For a 500 KLOC application and assuming $15,000 average cost to find a defect after deployment[1]…

If ASI Service finds ≥ 4 defects that would have missed detection through other testing, this more than recovers the cost of the inspection ($50,000).

1 [The Cutter Consortium]

© 2000 Reasoning, Inc.

Summary

- Traditional testing has limited ability to find defects in increasingly complex software applications
- Software Inspection can augment testing by finding more defects earlier and at less cost
- The limitations of manual or product-based software inspections can be eliminated by using Automated Software Inspection Services
- Automated Software Inspection Services have a compelling Return on Investment

© 2000 Reasoning, Inc.

QW2000 Paper 7W2

Ms. Andrea MacIntosh & Mr. Wolfgang Strigel (QA Labs Inc.)

"The Living Creature" - Testing Web Applications

BACK TO QW2000 PROGRAM

Key Points

- Web application testing
- Web application architecture
- Web user
- E-business

Presentation Abstract

While few people in the software development industry doubt the need or at least the advantage of including testing and quality assurance in development projects, it has taken many years, many projects, and many failures to get to this point. Now we are entering the age of web applications, a new era, and one that somehow doesn't seem like "real software development": these projects are fast, faster than any application software project, moving in Internet-time. Perhaps it is the speed of creation and deployment, the ease of publishing components, or the perceived lightweightness of these components ("it's just some JavaScript") that allows many to believe that testing can be omitted, or done on the fly by the development staff. But with the fast web-time environment comes a new type of customer - one who can, in the blink of an eye, browse away from your site if it's too slow, hard to use or doesn't work the first and every time.

Now, web application companies are seeing the importance of adding testing to their project schedules, and many are finding it to be a very difficult task. Why? One would think that a web application project could be treated much the same as a software application development project, and thus could be subject to many of the software engineering practices developed over the last 30+ years. But herein lies the problem: the initial assumption that a web application project is similar (if not identical) to a software application project is a false one.

Web applications target a different type of user, who has different needs and wants than the traditional desktop application user. Web applications also move at a completely different pace and move in a different timeframe than off-the-shelf software, a far faster one with shorter and shorter schedules. While traditional applications only had to be compatible with several versions of an operating system, web applications need to run on a wide spectrum of browser versions and cope with complex browser

incompatibilities that change much quicker than the traditional operating system release cycles. Web applications also have the extra burden of a wide variety of external marketing pressures, including shortened time-to-market, advertising revenues, and branding. Add to this situation a fundamentally different application architecture, one that follows more closely a business transaction model than any other, and you are facing quite a different beast than your traditional desktop application.

This paper will describe our experience in testing web applications and give some practical hints for the "do's and don'ts" in web testing.

About the Authors

Wolfgang B. Strigel is the founder and President of the Software Productivity Center Inc. (SPC), and of QA Labs Inc. The SPC is an international resource centre with products and services for software development organizations. QA Labs offers services in quality assurance and software testing, including QA outsourcing, consulting and training. Previously, he was a Vice President at MacDonald Dettwiler, a Canadian aerospace company, where he was responsible for the engineering department.

Mr. Strigel is a member of several academic advisory boards. He also serves on the board of several private corporations and the Japanese Software Productivity Society. Mr. Strigel is a member of the Editorial Advisory Board of IEEE Software Magazine, a member of TIA, CIPS, IEEE, ACM, PMI, ASQC and ISO standards council. He is an expert advisor to Giga Information Group. He is a frequent speaker at international conferences. Mr. Strigel received a B.Sc. (Munich, Germany) and M.Sc. (McGill University, Montreal, Canada) in Computer Science, and a MBA (SFU, Vancouver, Canada).

Andrea MacIntosh is one of the founding partners of QA Labs Inc., where she focuses on Quality processes, training and consulting. Her areas of technical expertise include Java, XML, web applications, Macintosh and Linux operating systems, file formats, and ISO 9000 series implementation. Previously she was the Director of Quality Assurance for Paradigm Development Corporation, where she worked on projects for Microsoft, Adobe, Corel, Inso, and Disney Interactive.

Ms. MacIntosh's formal education is in the field of Physics (B.Sc. UBC) where she had the opportunity to perform research in the area of non-linear dynamics of water turbulence and human cardiac patterns.

BACK TO QW2000 PROGRAM

The Living Creature

Testing Web Applications

Andrea MacIntosh and Wolfgang Strigel
QA Labs Inc.

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Outline

- Web Application Architecture
- Development Lifecycle of a Web Application
- Quality In The Web Space
- Testing In The Web Space

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Web Application Architecture

<u>Basic Workflow</u>

Browser

- ☒

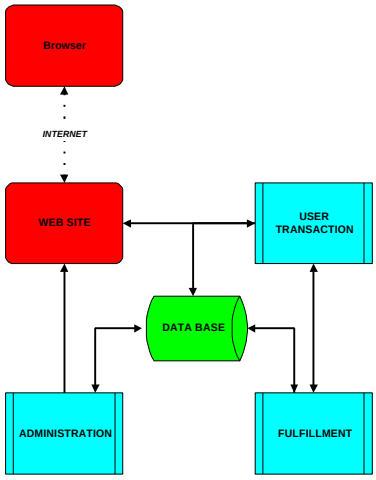
Web Server

- ☒

Application Server(s)

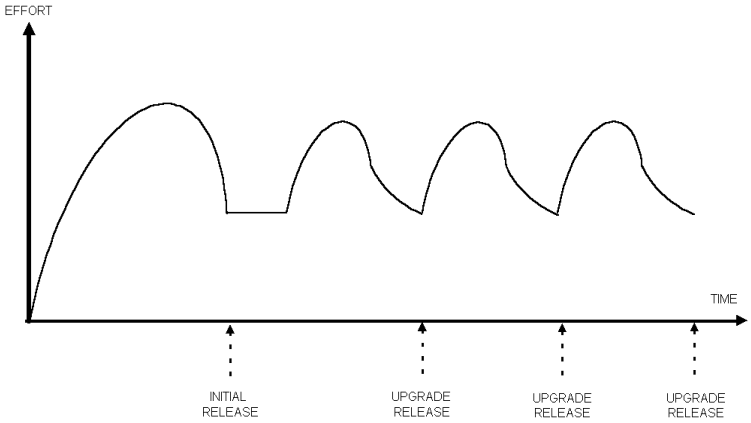
- ☒

Database



Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Development Lifecycle of a Web Application



Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Development Lifecycle

Requirements

- More to do with marketing, art, branding, and advertising than providing a solid set of core functionality
- Requirements for the initial release captured in a document and changes ("CRs") subject to rigorous scope management

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Development Lifecycle

Implementation

- Traditional Thinking: *"Prototypes are thrown away, never shipped"*
- New Web Thinking: *"Evolutionary development processes"*
  - High risk of insufficient testing before the site is launched

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Quality In The Web Space

- Reliability
- Recoverability
- Security
- Usability
- Performance

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Reliability

- Opinion of the authors that reliability of web applications is a <u>user-defined quality factor</u>
  - the definition of "reliable" as it applies to web applications is a subjective one, and that the <u>user</u> is the creator and modifier of any such definition

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Recoverability

- Recoverability includes
  - fail-safe switchover
  - re-synchronization with all connected systems (warehousing systems, payment fulfillment operations, etc.)
  - data validation to ensure that data has not been lost or become corrupted

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Security

- Some methods of reducing these kinds of risks include:
  - Researching current security issues in third-party products you plan on using - this may affect your decision to use that product

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Security

– Reviewing your design to try to remove as many potential security issues as possible, including examining architectural designs

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Security

– Writing "defensive" code, as demonstrated with the rules for good defensive Java programming listed in Chapter 7 of Securing Java, by Gary McGraw and Edward Felten.

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Security

- Creating coding standards that reflect some of these "defensive" techniques
- Performing code inspections on the high-risk components of the web application.

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Usability

- UI ≠ Usability
  - The UI is the browser
  - The UI is not images, logos
  - Most web applications exhibit poor usability
- Techniques
  - Pilot or focus groups
  - Computer Human Interface performance factors

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Performance

- Conduct some of the early performance testing on a minimum configuration machine
- Include connecting via modem (many users will have connection speeds of 28.8k)
- Automate performance testing so it is more accurate and can be run on a regular basis

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Testing In The Web Space

- What's different than testing any other application?
  - The Test Environment
  - Platforms and Browsers
  - Back to Front Test Approaches
  - Focusing Test Efforts

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

The Test Environment

- Your test team shouldn't crash your site -- they should crash a separate test server.
- Minimum Set-Up:
  - A development server used for the development team.
  - A staging server that is periodically updated with a new release, used as the test platform.
  - A production server that hosts the live site. Nobody, except the appointed web master is allowed to modify this server.

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Platforms and Browsers

- Testing schedules are aggravated by browser and operating system compatibility testing
- Separate the configurations into 2 groups: a core set of "primary" configurations, on which the majority of your testing will be completed; and a set of "secondary" configurations, on which security, performance, system, and acceptance testing will be performed

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Testing Back-to-Front

- Focus on testing a web application not from the client-side (browser), but from the server-side
- Scripting is your best tool here
  - Perl LWP or Win32::Internet libraries

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Focusing Your Test Efforts

- Develop a test plan which is as complete as possible and focuses on a Back-to-Front approach
- Structure the plan by functional component
- Prioritize components using the associated risk of failure as a guide
- Start testing the high priority items and work your way down to lower priorities until you run out of time.

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

Conclusion

Copyright © 2000 QA Labs Inc.
All Rights Reserved.

"The Living Creature" - Testing Web Applications

Table of Contents

Web Application Architecture 1
Figure 1: Web Application Architecture 2
The Development Life Cycle of a Web Application 2
Figure 2: Effort vs. Time for a Typical Web Application 5
Requirements 5
Implementation: The Prototype as Product 6
Quality in the Web-Space 7
Some Quality Factors of Web Applications 7
Reliability 8
Recoverability 8
Security 9
Usability 10
Performance 10
Testing in the Web-Space- What's different? 11
The Test Environment - Does such a thing currently exist in web-space? 12
Platforms and Browsers 13
Focusing Your Testing Efforts 14
Conclusion 14

Web Application Architecture

What do we mean by "Web Application"? There is an incredible range of sophistication in web applications from a simple company web site with "brochure ware" to sites like Yahoo or Amazon with complex search engines and order fulfillment. One way to look at the web application architecture is to take the model of a traditional business transaction application and to replace the user front end by the web site. A customer acquires goods and/or services from your company, in exchange for money. There are mechanisms in place to facilitate that transaction between client and company. Instead of a sales rep, a clerk, a cashier, or such person, you have a browser pointing at a web site. The company is never closed! Customers can serve themselves!

Think of a vending machine: this machine fills orders based on input from users, verifies transfer of funds, and has a basic user interface. Now add some complexity: make the UI a browser-based solution that must run in multiple browsers on multiple operating systems instead of a touch pad, and have the machine fill orders directly from a warehouse in the mid-Western US, while tracking and re-stocking (real-time) inventory. In general, people will not be putting coins in the machine, but entering their credit card numbers - which requires real time access to credit companies to have each transaction approved. Moreover, we would expect that all credit card information should be extremely secure.

Copyright © 2000 QA Labs Inc.

The average web application architecture is shown in Figure 1 (below). The client end of the system is represented by a browser, which connects to the web site server via the Internet. The centerpiece of all web applications is a relational database which stores dynamic contents. A transaction server controls the interactions between the database and other servers (often called "application servers"). Fulfillment may include interfacing with financial institutions, warehouse systems, etc. The administration function handles data updates and database administration. Of course, there are many possible permutations that form this basic picture.

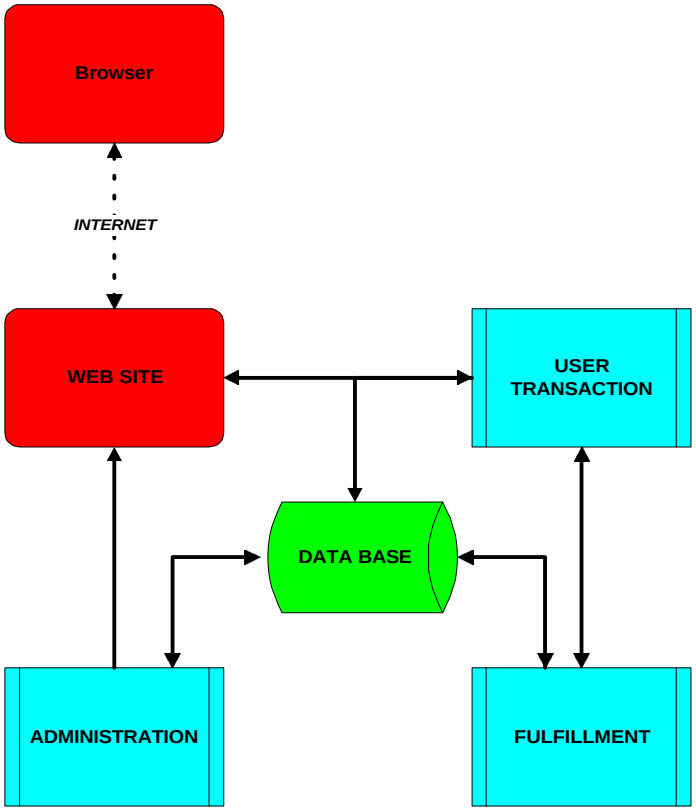


Figure 1: Web Application Architecture

Considering this architecture, it should now become clear that web applications are not simply web sites with some artwork and some HTML or Java. They are very similar to the traditional transaction systems with additional complexity at the front end. The testing effort required for such a system is considerably larger than for applications without web interface.

The Development Life Cycle of a Web Application

Most of us have been exposed to a few software development lifecycle models, such as the Spiral model, the Waterfall model, and so forth. While the typical software project includes such phases as planning, requirement gathering, analysis and design,

Copyright © 2000 QA Labs Inc.

implementation (coding), integration, testing, release and maintenance, how do these phases match up for a web application project? Here is one view the authors have frequently observed:

Typical software project	**Web application project**
Gathering of market/user requirements *"What are we going to build? How does it compare to products currently available?"* This is typically supported by a detailed requirements specification.	Gathering of market/user requirements *"What services are we going to offer our visitors/customers? What is the best user interface and navigation to reach the most important pages with a minimum of clicks? What are the current trends and hot technologies?"* This is typically based on discussions, notes and ideas.
Planning *"How long will it take our available resources to build this product? How will we test this product?"* Typically involves experience-based estimations and planning.	Planning *"We need to get this out NOW! Marketing has picked this date to go live; we'll just have to have it done by then (typically 3 to 4 months)."* Purely driven by available time window and resources.
Analysis and Design *"What technologies should we use? Any design patterns we should follow? What kind of architecture will allow us to reuse code most effectively, particularly for future versions?"* Mostly based on well known technologies and design methods. Generally complete before implementation starts.	Analysis and Design *"How should the site look? What kinds of logos and graphics will we use? How do we develop a 'brand' for our site? Who is our 'typical' customer? How can we make it usable? What technologies will we use?"* Short, iterative cycles of design in parallel with implementation activities.
Implementation *"Let's decide on the sequence of building blocks that will optimize our integration of a series of builds"* Sequential development of design components.	Implementation *"Let's put in the framework and hang some of the key features off of it. Then we can show it as a demo or pilot site to our prospective users."* Iterative prototyping and story-boarding with gradual transition of a prototype to a production site.

Copyright © 2000 QA Labs Inc.

Integration *"How does the product begin to take shape, as the constituent pieces are bolted together? Are we meeting our requirements? Are we creating what we set out to create in the first place?"* Assembly of components to build the specified system.	Integration This phase typically does not exist. It is a point in time when prototyping stops and the site goes live.
Testing *"Have we tested the product in a reproducible and consistent manner? Have we achieved complete test coverage? Have all serious defects been resolved in some manner?"* Systematic testing of functionality against specifications.	Testing *"It's just a website -- the designer will test it as (s)he develops it, right? How do you test a website? Make sure the links all work?"* Testing of implied features based on a general idea of desired functionality.
Release *"Have we met our acceptance criteria? Is the product stable? Has QA authorized the product for release? Have we implemented version control methods to ensure we can always retrieve the source code for this release?"* Building a release candidate and burning it to CD.	Release *"Go live NOW! We can always add the rest of the features later!"* Transfer of the development site to the live server.
Maintenance *"What features can we add for a future release? What bug fixes? How do we deal with defects reported by the end-user?"* Periodic updates based on feature enhancements and user feedback.	Maintenance *"We just publish new stuff when it's ready…we can make changes on the fly, since there's no installation required. Any changes should be transparent to our users…"* Integral part of the extended development life cycle for web apps.
Average timeframe for the above: **One to three years**	Average timeframe for the above: **4 months**

There are many possible terms for the web app development life cycle including the spiral life cycle or some form of iterative life cycle. A more cynical way to describe the most commonly observed approach is to describe it as the unstructured development similar to the early days of software development before software engineering techniques were introduced. The “maintenance phase” often fills the role of adding missed features and fixing problems.

Copyright © 2000 QA Labs Inc.

The fundamental difference from traditional development is the extreme time pressure under which web systems are developed. This necessitates a new approach, which can severely limit the scope of the initial release and pushes many features into iterations that follow the initial release. The resulting life cycle model could be called a whale/dolphin model. The whale is the initial hump of effort to release the first version of the web app and the dolphins are subsequent cycles of functionality increments. These subsequent cycles continue throughout the life of the web application (see Figure 2 below).

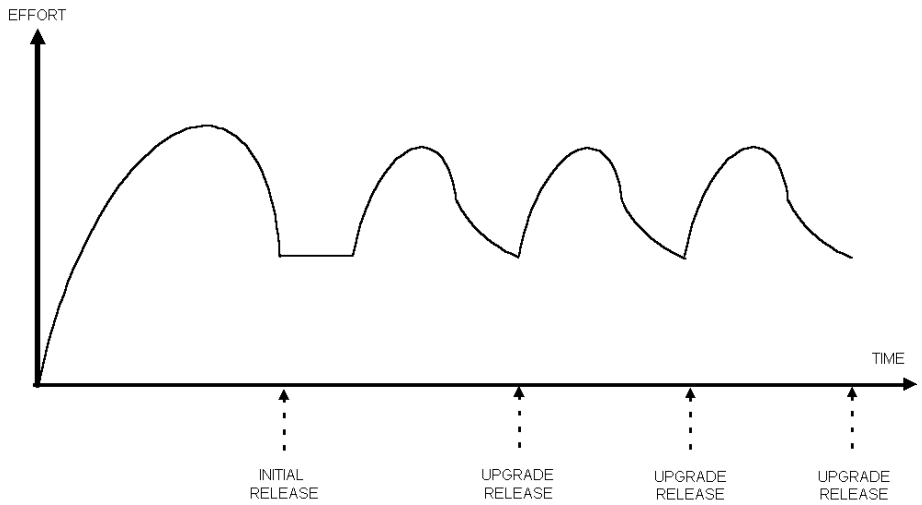


Figure 2: Effort vs. Time for a Typical Web Application

The first step in planning life cycle activities for a web application is to recognize the iterative nature of this product life cycle. Planning for subsequent releases right at the beginning of the project will greatly facilitate tradeoffs that may have to be made in each of the cycles.

Requirements

Requirements are traditionally the foundation for test planning. Web application projects often have requirements that are fuzzy to start with, subject to rapid changes throughout the project lifecycle, and often have more to do with marketing, art, branding, and advertising than providing a solid set of core functionality that works. Typically, there is an initial vision for the basic functions and the look-and-feel for the site. This vision is implemented for the initial release. However, the initial vision is subject to frequent updates and additions; especially once the initial release has gone live. A major reason for the initial release is to meet time-to-market requirements while postponing additional features that often are added later. As a result, we have an initial development lifecycle followed by ripples of smaller cycles for feature additions. The web-time that so constrains these kinds of projects usually does not allow for complete documentation and analysis of the initial requirements and this situation does not improve much for future additions.

Copyright © 2000 QA Labs Inc.

This makes test planning difficult. It may have to be based on intuitive interpretation of intended functionality. Even more aggravating is the very fluid nature of the functionality of the site design and the lack of documenting this change. As a result, many test efforts may go off into a “wild goose chase” testing for functionality that was changed or deleted.

> "We'll write them as we go."
> "We don't have time to write requirements.... We have to ship!"
> "Requirements? For a website?"

Sound familiar? The following is a direct quote from *The Mythical Man-Month* by Frederick Brooks, Jr.: "The most pernicious and subtle bugs are system bugs arising from mismatched assumptions made by the authors of the various components" (p.142). The writing of requirements is one simple method to reduce the amount of assumptions that have to be made by project personnel, including testers. Application software developers, testers, and managers have learned, from painful and expensive mistakes, that this approach can make all the difference in delivering a solid product to the customer.

Serious testing is impossible in an environment of uncontrolled changes to requirements. Combining this statement with the realities of web development, we suggest a simple technique that has proven useful for both the development team and the test group. Requirements and functional specifications for the initial release should be captured in a document. Any subsequent changes (called change requests or "CRs") should be managed and subjected to rigorous scope management. The set of test cases will then be based on the combination of the initial spec and the approved CRs. In the end, it is not uncommon that the set of approved CRs will represent the majority of the feature set for the system. There are many change request management systems on the market that can be used for this purpose.

Implementation: The Prototype as Product

One of the fundamental software engineering principles is that prototypes are thrown away, never shipped. Occasionally, a company will, under time-to-market pressures, ship a product that has morphed from a prototype, often with poor results. However, it is common practice to ship web applications that either are prototypes themselves or are early descendants of prototypes. The implementation phase then looks much like *evolutionary development*, where requirements changes lead to an evolutionary sequence of prototypes. Any of these prototypes may be declared to be the initial production version when time for additional iterations runs out.

This method bears a high risk of not allowing for sufficient testing before the site is launched. If the development organization feels that it must follow the evolutionary model, each iteration should be finished off with a test pass on a separate staging server that is not subjected to any interference from development changes.

Copyright © 2000 QA Labs Inc.

Quality in the Web-Space

Remember, many different companies define quality in many ways, but to the end user, quality always means, "Am I satisfied?" If we define quality as perceived value to the user over cost to the user, then web applications, which have no real cost to the user, should move towards infinitely greater quality. In fact, as the cost goes down and the marketplace becomes saturated with more and more sites that offer similar transaction functionality, and as these sites become increasingly complex, we see the opposite occur. Decreasing quality of software, for both desktop applications and web applications, is something that has been occurring for quite some time now, and the sad truth is that end users have been tolerating (if not encouraging) this trend.

We believe that market forces will cause this trend to reverse. As the Internet becomes swamped with dot.coms competing against each other, the limited attention span of users will reward only those sites that do not disappoint the user. This is the key argument for increasing pressure towards better testing of web applications: with traditional software a user has spent a certain amount of money and hence feels motivated to get the best utility from his or her investment. The alternative would only be to buy a different solution (which implies more expenditure, a new learning curve, incompatibilities etc.). A web user does not have such a difficult choice. If she uses an airline reservation system that does not behave according to expectations, she can click through to another service provider and see if she likes that environment better. The switching costs for most web applications are so low, that users will simply browse away from sites that exhibit poor usability. (Refer to Jakob Nielsen's and Donald A. Norman's article for *InformationWeek Online* entitled "Usability On The Web Isn't A Luxury" at http://www.informationweek.com/773/web.htm, or refer to Dr. Neilsen's website, http://www.useit.com, for more information on usability in general.) We believe that quality and functionality of web sites will ultimately be a major factor in the inevitable shake-out, which will reduce the number of redundant dot.coms.

Some Quality Factors of Web Applications

Security, reliability, and recoverability are all issues that can make or break a site. Up-time requirements for web applications are far more stringent than for off-the-shelf/shrink-wrap software. Web sites are just **not** allowed to fail, become corrupt, or exhibit poor usability, while the average computer user may tolerate buggy software (but not so buggy that they cannot do their work).

Some key quality factors that can be related to web applications include:

1) Reliability;
2) Recoverability;
3) Security;
4) Usability; and
5) Performance.

Copyright © 2000 QA Labs Inc.

Reliability
One definition of "reliable" is *exhibiting a reasonable consistency in results obtained*. How many web sites today can be called "reliable"? Another definition is *that which may be relied upon, worthy of confidence, trustworthy*. Perhaps a web application is trusted by users who use an on-line banking web application (service) to complete all of their banking transactions. One would hope that the results are consistent. However, perhaps the site is not always accessible on a consistent basis or displays periodic performance problems.

One example is that of a financial institution which set-up on-line trading services for its clients. Many customers signed up for and used the service. However, this institution did not grow the back-end application and database servers *and* the fulfillment personnel (brokers, traders, etc.) in line with the growth of the customer base. It wasn't long before customers could not access the site for periods of three to fours hours at a time, and when they did get access to request a trade, it often took several additional hours before their trades were processed. One can easily image the frustration and potential financial losses of the customers when their trades could not be completed on time and as requested. This would be an example where a lack of consistency implies a lack of reliability. (This scenario, from the perspective of the developing company, may be more of a scalability issue; however, from the user's point of view, it is a reliability issue.) A simpler example of reliability might be a shopping site that is constantly unavailable to its users, whether due to excessive traffic or hardware/software problems. A user will, after successive failed attempts to connect to the site, browse to a competitor's site.

It is the opinion of the authors that reliability of web applications is purely a user-based quality factor. That is, the definition of "reliable" as it applies to web applications is a subjective one, and that the user is the creator and modifier of such a definition. Having web application project personnel define "reliability" as it should apply to their web application is misleading. This is one more reason to acquire information regarding targeted audiences when designing and testing web applications.

Recoverability
This is another quality factor that is often ignored or put-off until after the initial release. Many web applications will have a back-up or "redundant" server -- a server to which web traffic is rerouted should the primary server fail. This set-up may be mimicked for the database server component(s) as well. The re-routing mechanism(s) must be tested with methods similar to tests performed on fault tolerant systems. However, recoverability implies much more than a fail-safe switchover. It has to be re-synchronized with all connected systems, such as warehousing systems, payment fulfillment operations, etc., as well as performing data validation to ensure that data has not been lost or become corrupted. This can increase the complexity of test scenarios significantly.

The potential financial losses from having your web application unavailable are large. If a user finds your service unavailable for an excessive period of time (excessive from the user's perspective), the likelihood of that user switching or browsing to a competitor's service is increased. The quicker your site can recover in a manner that is transparent to

Copyright © 2000 QA Labs Inc.

the user, the less chance you give that user to browse away from your site, and thus your services. If the site cannot recover quickly (or in a manner that is transparent to the user), the next step is to manage the user's expectations, by informing the user when you expect your site to be available and functional. Explaining to a user that your site will be available within 24 hours from the shut down will allow you to keep more users coming back to your site once restored, as compared to providing the user with no information.

Security

Probably the most critical criterion for a web application is that of security. The need to regulate access to information, to verify user identities, and to encrypt confidential information is of paramount importance. Credit card information, medical information, financial information, and corporate information must all be protected from persons ranging from the casual visitor to the determined cracker. There are many layers of security, from password-based security to digital certificates, each of which has its pros and cons. For a good on-line reference about security issues, refer to the W3C FAQ on Security (http://www.w3.org/Security/Faq/www-security-faq.html).

Many of the security measures used for web applications are third-party products. Certificates, Secure Sockets Layer (SSL) software, web server software (IIS, Apache, etc.), and so forth, are all products created by companies other than your own. They all have defects, some of which can be exploited by those who wish to intercept secure data, corrupt your servers or databases, hijack your content and/or scripts, or tap into password files. When a security defect is discovered, it can difficult to see a solution in a reasonable time frame from a third-party vendor. Which leaves only one feasible option: modify *your* product to deal with these defects.

Some methods of reducing these kinds of risks include:

1) Researching current security issues in third-party products you plan on using - this may affect your decision to use that product;
2) Reviewing your design to try to remove as many potential security issues as possible, including examining architectural designs;
3) Writing "defensive" code, as demonstrated with the rules for good defensive Java programming listed in Chapter 7 of Securing Java, by Gary McGraw and Edward Felten, (copyright 1999, John Wiley and Sons) available on-line at http://www.securingjava.com/chapter-seven/chapter-seven-1.html.
4) Creating coding standards that reflect some of these "defensive" techniques; and
5) Performing code inspections on the high-risk components of the web application.

A dangerous and common approach is to release software and wait for someone to discover a security-related defect, which can then be addressed by releasing a "patch." For a web application, the patch can usually be installed in a seamless manner (one that is transparent to the user). However, since security has a large dependence on environment, changing the environment by installing such a patch may put your web application at risk.

Copyright © 2000 QA Labs Inc.

Usability
Usability is a critical area for a web application. In the past, only very sophisticated projects consulted GUI designers or experts in human-machine interaction. These specialists are now in greater demand since the success of a web application depends largely on usability. Many of the web GUI designers are in fact artists, desktop publishers, and the like. They may not have had formal training or even exposure into some of the basic GUI and usability principles.

Usability of web applications will gain in importance over the next few years, as more and more web applications emerge, and as web application users become more experienced, or “sophisticated” users. The sophisticated user will have very different usability issues than a novice user. To date, the Internet has been about providing information (largely textual information) and a few services to relatively inexperienced users. As that model moves towards increased services for intermediate or sophisticated users, new approaches will have to be undertaken and implemented.

How do you test this without a large beta test group? Some companies are using the concept of a "pilot" site, which is a scaled-back version of the web application, accessible only to a small to medium-sized focus group. Feedback regarding the interfaces, the look and feel, and the workflow of the site is then collected from these groups and used to modify the application before it goes live (or for future iterative releases). Understanding the types of users that will visit the site will allow initial design efforts to focus on those kinds of users. The focus group can then be used to confirm or correct the initial design.

There are two approaches to this problem. The first is to capture and quantify the meaning of learnability, understandability, and operability in a testable form. In other words, it is an attempt to formulate these testable requirements by describing how they are supposed to be accomplished. This may then form the basis of a usability test plan. A second approach is to gather a group of representatives of the target user community and to let them work through the site while observing their problems and the bugs they run into. However, this approach may make it difficult to clearly determine what constitutes a bug. There are many techniques that can be used to help distinguish usability defects from feature requests, such as using "thinking aloud" and "question asking" combined with basic performance information gathering. (For basic information on these techniques and reference links to other resources, refer to James Hon's *The Usability Methods Toolbox* website at http://www.best.com/~jthom/usability/.)

Performance
Performance testing involves testing a program for timely responses. The time needed to complete an action is usually benchmarked, or compared, against either the time to perform a similar action in a previous version of the same program or against the time to perform the identical action in a similar program. For example, the time to open a new file in one application would be compared against the time to open a new file in previous versions of that same application, as well as the time to open a new file in the competing application. The benchmark time for a piece of software can also be defined explicitly by

Copyright © 2000 QA Labs Inc.

the client in the requirement specification document. When documenting these requirements, state the performance requirements in terms of real numbers instead of using statements like "must be at least as fast as the previous version." Using real numbers makes reporting performance problems much easier to rank in order of priority and severity. Report performance problems in terms of the percentage of improvement needed to meet the performance requirements.

When conducting performance testing, be sure to pay attention to the data volume (i.e. file size) so you're comparing apples to apples instead of apples to oranges. Calibrate your performance test machine against other machines of the same class to make sure that the times are comparable. Make sure that at least some performance testing is done early, even if the formal performance testing is scheduled to be conducted at a later milestone. Early performance testing will give an indication of how much work will need to be done in order to meet the performance requirements. Conduct some of the early performance testing on a minimum configuration machine. Include connecting via a dial-up connection with a modem (many users will have connection speeds of 28.8k, yet many testers are testing via a T1 or xDSL connection) so any problems with performance can be identified as early as possible. Whenever possible automate performance testing so that it is more accurate and can be run on a regular basis.

One flavor of performance testing is load testing. Load testing for a web application can be thought of as multi-user performance testing, where you want to test for performance slow-downs that occur as additional users use the application. The key difference in conducting performance testing of a web application versus a desktop application is that the web application has many physical points where slow-downs can occur. The bottlenecks may be at the web server, the application server, or at the database server, and pinpointing their root causes can be extremely difficult. Refer to Mark D. Anderson's article in *Software Testing and Quality Engineering* (September/October 1999, Vol 1, Issue 5, pages 30-41) for a good discussion on load (multi-user performance) testing.

Testing in the Web-Space- What's different?

By now it should be apparent that testing web applications is not trivial. Testing a web page with relatively static content and little to no forms will take very little time (Are all the links correct and working? Does all content load correctly? Is loading time fast enough?). Testing complete e-commerce applications requires much more sophisticated testing strategies, and thus more time.

One of the major weaknesses of web application testing is inadequate technical expertise and ability. Testers have to understand subtle browser, operating system, web server, and database differences. The more they know about **scripting** (ASP, XML, HTML, etc.), **databases** (Oracle, SQL, etc.), **web servers** (IIS, Apache, etc.), and the **data transfer mechanism** behind the UI (TCP/IP, HTTP, FTP, etc.), the more effective they will be. Testers simply cannot just test the functionality by exercising the UI (in this case, the browser); they will miss all the other aspects of testing required for web applications

Copyright © 2000 QA Labs Inc.

(such as performance, security, database integrity, etc.). Remember, crackers do not use browsers to crack sites; they use scripts.

As well, the lack of mature test tools makes automation difficult. This situation is reminiscent of when Java first hit the high-tech scene. Developers and project managers alike wanted to use this new technology. Testers suddenly had their workloads doubled, tripled, or more, simply because of the number of configurations and the lack of any mature testing and automation tools. One company for whom one of the authors worked, had developed a large Java application, and was forced to write custom debugging and automation tools because, at the time, nothing comparable existed. This situation will not change as long as web technology continues to evolve at the current rate.

The Test Environment - Does such a thing currently exist in web-space?

Poorly defined development and test environments can hamper version control and other configuration management efforts. How do you rollback code changes when you have no previous build? How does new functionality and defect fixes get migrated into each build? Does the term “build” mean anything in the web-space? For most web application projects, it doesn't. Test personnel cannot revert to a "known state" if the source code is not being archived or not being labeled or branched in the version control repository. Not having a previous release to revert to for testing purposes makes isolating and analyzing defects more difficult as the environment continually becomes more complex. Another problem area is the common (and dangerous) practice of migrating defect fixes and new functionality to a live server prior to testing, and testing on live servers. Your test team shouldn't bring down your site; they should bring down a separate test server.

What is the best environment to set-up for a web-testing team? How can companies organize their environment to help testing, instead of making it nearly impossible? We suggest at a minimum three separate servers:

- A development server used for the development team. This may be a place for a prototype or simply to play with features.
- A staging server that is periodically updated with a new release. This server can also be used as the test platform. Crashing the staging server does not affect the development team or the live site.
- A production server that hosts the live site. Nobody, except an appointed web master is allowed to modify this server. Its contents are updated periodically from the staging server.

This is a minimum configuration and there are many variations of this basic scheme, especially when the full (final) web application itself consists of several servers, not all of which are necessarily at the same geographic location.

A test environment should mimic the destined (planned) deployment environment as closely as possible. Testing a web application in a LAN setting will not be the same as testing a web application hosted on a series of external web and application servers. Proper security testing cannot be performed on a web application in any configuration other than the final one. The complexity of the networking, security (firewalls, proxy

Copyright © 2000 QA Labs Inc.

servers, etc.), hardware, and software configurations cannot be reproduced to any degree of accuracy in an internal LAN-type environment. If your company were developing software for hand-held devices, you would test on the intended devices, not on your NT machine. Yet, many companies choose to test in this manner. With a short test schedule, there can be some benefit to testing on an environment other than the final one - if the testing performed is only usability testing or beta group/pilot site testing with the sole purpose of gathering user feedback for future iterations.

Platforms and Browsers

Testing issues are aggravated by the large variety of browser and operating system compatibility testing required. Creating a matrix of operating systems vs. browser versions, we see how large this can be. Since it is very difficult to force users to use the latest browsers (there will always be someone using Netscape 2.x somewhere), you will have to test with older versions.

One cannot assume that browsers behave in similar ways. There are known differences between browser versions and between MS Explorer and Netscape. Nonetheless, it is important to understand the issue of multiple configuration testing, and how that can really grow your test needs in a hurry. Look at this basic matrix. Imagine having to test on each of these platforms with each of the main browsers!

	Netscape	Internet Explorer	HotJava	Other
Windows 95				
Windows 98				
Windows 98SE				
Windows 98ME				
Windows NT 4.0 Workstation w/SP 3				
Windows NT 4.0 Workstation w/SP 5				
Windows NT 4.0 Workstation w/SP 6				
Windows NT 4.0 Server w/SP 6				
Windows 2000 Pro				
Windows 2000 Server				
MacOS 9.0				
MacOS 8.6				
MacOS 8.0				
MacOS 7.6				

This list does not even include WebTV, Linux, UNIX, and UNIX-like machines, nor does it express the multitudes of browser versions. There are significant HTML and/or XML compliance, security, and performance differences between browsers and between browser versions, so it is important to test on a variety of browser versions. A good understanding of the make up of the application's typical users and a breakdown of the

Copyright © 2000 QA Labs Inc.

browsers and browser versions currently being used should help focus your test effort in this area. For example, Microsoft Explorer offers good performance when connecting to the IIS server while the Netscape Navigator browser performance is remarkably more sluggish in that configuration.

The goal is to separate the various configurations into two groups: a core set of "primary" configurations, on which the majority of your testing will be completed; and the remaining or "secondary" configurations, on which security, performance, functionality, system, and acceptance testing will be performed (time permitting of course). Separating the possible configurations into these two lists will allow a test team to best utilize the short amount of time allocated to testing.

Focusing Your Testing Efforts

Testing for usability, browser compatibility and other front-end aspects may easily detract from testing the functionality of the back-end. Fortunately, back-end applications represent more traditional software applications such as data base access, transaction servers, etc. In many cases, these functional areas can be isolated and exercised using test drivers and stubs. Provided the system is designed using a modular architecture, testing the back-end components can be done in parallel to the development and testing of front-end components.

Critical for overall test planning is to recognize that there will probably not be enough time to test all parts of the system exhaustively. In an ideal world, we would develop a test plan and systematically work through the plan. In web time, this is not likely to happen. Here is what we suggest:

- Develop a test plan which is as complete as possible
- Structure the plan by functional component
- Prioritize components using the associated risk of failure as a guide to determine the priority
- Start testing the high priority items and work your way down to lower priorities until you run out of time.

The required risk prioritization is the most critical part. It should be derived from the business case, reviewed with those who define the product vision, and verified with user representatives.

Conclusion

Web applications require a new perspective on software development and testing practices. While many activities are similar to traditional software engineering, we have to adapt to the new realities of developing in web time. Some of these realities, including shorter life cycles, multidisciplinary teams, rapidly changing technology, increase the risks of development. In the past, we learned to increase the rigor of the development process to manage increased risk. However, in an era of shorter and shorter development cycles, the time available for process must also be shortened. The only way out of this conundrum is to focus on those process aspects that give the highest return for the time

Copyright © 2000 QA Labs Inc.

invested. Each team member has to work smarter, not just harder. When it comes to testing, this means that the time spent on test planning is becoming even more valuable since it allows us to test strategically important parts of the application instead of testing tactically convenient ones.

Copyright © 2000 QA Labs Inc.

QW2000 Paper 7M2

Mr. Rex Black
(Rex Black Consulting Services, Inc.)

The Fine Art of Writing a Good Bug Report

BACK TO QW2000 PROGRAM

Key Points

- Bug reporting is a critical, daily task for most test personnel during test execution. Bugs reported well get development and management attention and get fixed; bugs reported poorly are ignored, returned, or improperly prioritized.
- We test professionals have paid a great deal of attention to the art of test software, but we have talked much less about the art of reporting test results, especially failures.
- This presentation will introduce attendees to a ten-step process for writing bug reports that are accurate, concise, thoroughly edited, well-conceived, and high-quality technical documents.

Presentation Abstract

In a speech at Quality Week '99, Roger Sherman, a Microsoft test manager, identified the leading cause of bug report closure as "unreproducible." This is a regrettable circumstance, since such bug reports waste precious time during tight development schedules, add absolutely nothing to product quality, and lead to frustration and bad feelings between development engineers and test engineers. Sometimes, these bug reports arise from transient or random events, inconsistency of tools and configurations between test and development, or a vague definition of "correct" behavior under the tested conditions, but many bug reports closed as unreproducible are unclear, misleading, or just plain wrong.

Fortunately, I have learned some tricks for writing great bug reports that get management attention, communicate clearly to developers, and get fixed. Not only do these techniques provide solid technical payoffs in terms of a greater proportion of bugs fixed, they also communicate to development and to management that testers are serious about helping developers fix bugs. Writing bugs reports using these methods on projects I have managed, only around one out of eight of bug reports are closed without a fix.

About the Author

Rex Black has almost two decades in the computer industry, primarily in testing and quality assurance. He is the President and Principal Consultant of Rex Black Consulting Services, Inc., an international software and hardware testing and quality assurance consultancy. RBCS' clients include Dell, Sun, First USA/Bank One, Hitachi, Netpliance, Motorola, Omnipoint, Pacific Bell, Clarion, and

others. His work with these clients has taken him to Taiwan, Hong Kong, Japan, the U.K., Canada, Germany, Holland, France, Spain, Switzerland, and Italy, along with locations throughout the United States.

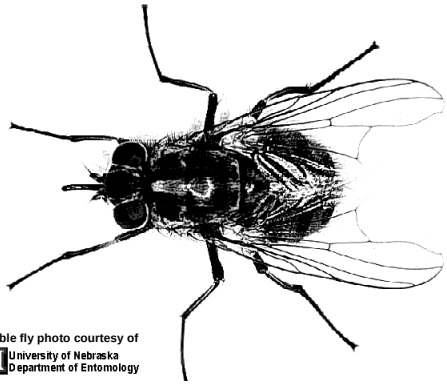
Rex is the author of Managing the Testing Process , published by Microsoft Press in their Best Practices series,. He gives speeches, presents papers, and teaches tutorials at events such as Quality Week, Practical Software Quality Techniques, Software Testing and Analysis Review, and others on topics related to software testing.

He is also a Trainer for the International Institute for Software Testing, teaching their certification course (CSTP) on software test management.

Mr. Black holds a B.Sc. in Computer Science and Engineering from UCLA. He belongs to the Association for Computer Machinery and the American Society for Quality.

BACK TO QW2000 PROGRAM

The Fine Art of Writing a Good Bug Report



Stable fly photo courtesy of
University of Nebraska
Department of Entomology

Rex Black
RBCS, Inc.
31520 Beck Road
Bulverde, TX 78163

Phone: +1 (830) 438-4830
Fax: +1 (830) 438-4831
HTTP: //www.rexblackconsulting.com
E-mail: Rex_Black@rexblackconsulting.com

What Is a Bug Report and Why Do We Write Them?

- Bug report is a technical document
  - Describes failure mode in system under test (SUT)
  - The only tangible “product” of testing
- Not a management problem escalation tool
  - “Build not delivered on time” is not a bug report summary
  - “Build 781 fails to install” is a bug report summary
- Written to increase product quality
  - Documents a specific quality problem quality of SUT
  - Communicates to developers

 Copyright (c) 2000, Rex Black, All Rights Reserved

Are Bad Bug Reports a Problem?

- Anecdotally, developers return many bug reports as unreproducible, leading to:
  - Wasted time writing the report
  - Frustration for tester and developer alike
  - No increase in product quality
- Bug reports can be unreproducible due to:
  - Intermittence
  - Inconsistent test/development environments
  - Disputes over “correct” behavior
- But many unreproducible bug reports are poorly conceived and poorly written

 Copyright (c) 2000, Rex Black, All Rights Reserved

Ten Tips for a Good Bug Report

1 Structure: test carefully
2 Reproduce: test it again
3 Isolate: test it differently
4 Generalize: test it elsewhere
5 Compare: review results of similar tests
6 Summarize: relate test to customers
7 Condense: trim unnecessary information
8 Disambiguate: use clear words
9 Neutralize: express problem impartially
10 Review: be sure
△ Remember, writing is creative: two good bug reports on one problem can differ in style and content (but not substance)

 Copyright (c) 2000, Rex Black, All Rights Reserved

Case Study: SpeedyWriter

- Let's use a case study to illustrate these techniques
- SpeedyWriter
- A Java-based word processor
- Supports the usual word processing functions
- Differences highlighted in *italics* in case study

 Copyright (c) 2000, Rex Black, All Rights Reserved

Structure

- Structured testing foundational to good bug reports
  - Use deliberate, careful approach to testing
  - Follow written test cases or run automated ones per written or standardized process
  - Take careful notes
- Bug reporting begins when expected and observed results differ
- Sloppy testing results in sloppy bug reports

 Copyright (c) 2000, Rex Black, All Rights Reserved

Structured Testing Paradigms

Ben Franklin standing in a thunderstorm hoping to get hit by lightning

Good software testing should be more like a carefully designed laboratory experiment than a random walk in an electrical storm--or worse. Testers are engineers, after all.

M. Maria Sklodowska-Curie performing a carefully designed experiment

 Copyright (c) 2000, Rex Black, All Rights Reserved

Reproduce

- Always check reproducibility of the failure as part of writing bug report
  - ✓Three times is a good rule of thumb
- Document a crisp sequence of actions that will reproduce the failure
- Report intermittent, hard-to-repeat failures
  - – Note failure incidence rate (i.e., 1 in 3 tries)
- Clean steps to reproduce addresses issue of unreproducibility head-on

 Copyright (c) 2000, Rex Black, All Rights Reserved

Good	Bad
<u>Steps to Reproduce</u> *1. I started the SpeedyWriter editor, then I created a new file.* *2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time, using different effects each time, bold, italic, strikethrough, and underline.* *3. I highlighted the text, then pulled down the font menu, and selected Arial.* *4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.* *5. I was able to reproduce this problem three out of three tries.*	*Nasty bug trashed contents of new file that I created by formatting some text in Arial font, wasting my time.*

 Copyright (c) 2000, Rex Black, All Rights Reserved

Isolate

- Change variables that may alter symptom
  - Make changes one by one
  - Requires thought and understanding of SUT
  - May not be immediately obvious
- Can be extensive
  - Match amount of effort to severity of problem
  - Avoid getting into debugging activities
- Good isolation shows due diligence and gives developers head start on debugging

 Copyright (c) 2000, Rex Black, All Rights Reserved

Better

Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time, using different effects each time, bold, italic, strikethrough, and underline.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.

Isolation
On the vague suspicion that this was just a formatting problem, I saved the file, closed SpeedyWriter and reopened the file. The garbage remained.
If you save the file before Arializing the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

Good

Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time, using different effects each time, bold, italic, strikethrough, and underline.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.

Isolation
Doesn't happen on Solaris.

 Copyright (c) 2000, Rex Black, All Rights Reserved

Generalize

- Look for related failures in SUT
  - Does the same failure occur in other modules or locations?
  - Are there more severe occurrences of the same fault?
- Avoid confusing unrelated problems
  - Same symptom can arise from different bugs

➢ Generalizing reduces duplicate bug reports and refines understanding of failure

 Copyright (c) 2000, Rex Black, All Rights Reserved

Better

Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.

Isolation
Also happens with Wingdings and Symbol fonts.
On the vague suspicion that this was just a formatting problem, I saved the file, closed SpeedyWriter and reopened the file. The garbage remained.
If you save the file before changing the font of the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

Good

Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time*, using different effects each time, bold, italic, strikethrough, and underline*.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.

Isolation
On the vague suspicion that this was just a formatting problem, I saved the file, closed SpeedyWriter and reopened the file. The garbage remained.
If you save the file before Arializing the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

 Copyright (c) 2000, Rex Black, All Rights Reserved

Compare

- Examine results for similar tests
  - Same test run against earlier versions
  - Similar conditions, other tests, same version
- Is failure a regression?
  - Change introduces defect not in earlier versions
  - Usually found when previously passed tests fail
- Not always possible
  - Test previous blocked, reinstall impractical
  - Tested feature unavailable in earlier versions

 Copyright (c) 2000, Rex Black, All Rights Reserved

Better

Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.

Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Also happens with Wingdings and Symbol fonts.
On the vague suspicion that this was just a formatting problem, I saved the file, closed SpeedyWriter and reopened the file. The garbage remained.
If you save the file before changing the font of the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

Good

Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.

Isolation
Also happens with Wingdings and Symbol fonts.
On the vague suspicion that this was just a formatting problem, I saved the file, closed SpeedyWriter and reopened the file. The garbage remained.
If you save the file before changing the font of the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

 Copyright (c) 2000, Rex Black, All Rights Reserved

Summarize

- Put a short "tag line" on each report
  - – Capture failure and impact on customer
  - ≈ Analogy: Bumper sticker
- Harder than it seems
  - – Testers must spend time thinking about this

√ Advantages of good summaries
  - – Get management attention
  - – Name bug report for developers
  - – Help set priorities

 Copyright (c) 2000, Rex Black, All Rights Reserved

Better

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.
Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Also happens with Wingdings and Symbol fonts.
On the vague suspicion that this was just a formatting problem, I saved the file, closed SpeedyWriter and reopened the file. The garbage remained.
If you save the file before changing the font of the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

Good

Steps to Reproduce
1. I started the SpeedyWriter editor, then I created a new file.
2. I then typed in four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. I highlighted the text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all the text into meaningless garbage, wasting the user's time.
5. I was able to reproduce this problem three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Also happens with Wingdings and Symbol fonts.
On the vague suspicion that this was just a formatting problem, I saved the file, closed SpeedyWriter and reopened the file. The garbage remained.
If you save the file before changing the font of the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

 Copyright (c) 2000, Rex Black, All Rights Reserved

Condense

- Eliminate extraneous words or steps
  - Reread report carefully
  - Avoid both cryptic commentary and droning on
- Are any details or actions irrelevant?
- Everyone's time is precious, so don't waste any of it on unnecessary verbiage, but don't cut any meat, either

 Copyright (c) 2000, Rex Black, All Rights Reserved

Better

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.
Steps to Reproduce
1. Started SpeedyWriter editor, then created new file.
2. Typed four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. Highlighted text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all text into meaningless garbage, wasting the user's time.
5. Reproduced three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Also happens with Wingdings and Symbol fonts.
On vague suspicion this was a formatting problem, saved file, closed SpeedyWriter and reopened file.
Garbage remained.
Saving file before changing font prevents bug.
Bug does not occur with existing files.
Only happens under Windows 98.

Good

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.
Steps to Reproduce
1. *I* started the SpeedyWriter editor, then *I* created *a* new file.
2. *I* then typed *in* four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. *I* highlighted *the* text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all *the* text into meaningless garbage, wasting the user's time.
5. *I was able to* reproduce *this problem* three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Also happens with Wingdings and Symbol fonts.
On *the* vague suspicion *that* this was *just* a formatting problem, *I* saved *the* file, closed SpeedyWriter and reopened *the* file. *The* garbage remained.
If you save the file before changing the font of the contents, the bug does not occur.
The bug does not occur with existing files.
This only happens under Windows 98.

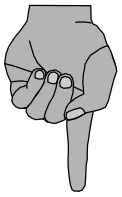
 Copyright (c) 2000, Rex Black, All Rights Reserved

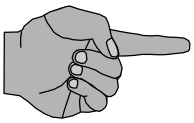
Disambiguate

- Remove, rephrase, or expand vague, misleading, or subjective words and statements
- Make sure report is not subject to misinterpretation

√ Goal: Clear, indisputable statements of fact

∴ Lead developer by the hand to bug





 Copyright (c) 2000, Rex Black, All Rights Reserved

Better

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.
Steps to Reproduce
1. Started SpeedyWriter editor, then created new file.
2. Typed four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. Highlighted *all four lines of* text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all text into meaningless garbage, *including control characters, numbers, and other binary junk,* wasting the user's time.
5. Reproduced three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Reproduced with same steps using Wingdings and Symbol fonts.
On vague suspicion this was a formatting problem, saved file, closed SpeedyWriter and reopened file.
Garbage remained.
Saving file before changing font prevents bug.
Bug does not occur with existing files.
Only happens under Windows 98*, not Solaris, Mac, or other Windows flavors*.

Good

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.
Steps to Reproduce
1. Started SpeedyWriter editor, then created new file.
2. Typed four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. Highlighted text, then pulled down the font menu, and selected Arial.
4. This nasty bug trashed all text into meaningless garbage, wasting the user's time.
5. Reproduced three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Also happens with Wingdings and Symbol fonts.
On vague suspicion this was a formatting problem, saved file, closed SpeedyWriter and reopened file.
Garbage remained.
Saving file before changing font prevents bug.
Bug does not occur with existing files.
Only happens under Windows 98.

 Copyright (c) 2000, Rex Black, All Rights Reserved

Neutralize

- Deliver bad news gently
- Be fair-minded in wording and implications
- Avoid:
  - Attacking developers
  - Criticizing the underlying error
  - Attempting humor or using sarcasm
- Confine bug reports to statements of fact

△You never know who'll read the reports

 Copyright (c) 2000, Rex Black, All Rights Reserved

Better

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.
Steps to Reproduce
1. Started SpeedyWriter editor, then created new file.
2. Typed four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. Highlighted all four lines of text, then pulled down the font menu, and selected Arial.
4. *All text converted to control characters, numbers, and other apparently random binary data.*
5. Reproduced three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Reproduced with same steps using Wingdings and Symbol fonts.
On vague suspicion this was a formatting problem, saved file, closed SpeedyWriter and reopened file.
Garbage remained.
Saving file before changing font prevents bug.
Bug does not occur with existing files.
Only happens under Windows 98, not Solaris, Mac, or other Windows flavors.

Good

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.
Steps to Reproduce
1. Started SpeedyWriter editor, then created new file.
2. Typed four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. Highlighted all four lines of text, then pulled down the font menu, and selected Arial.
4. *This nasty bug trashed all text into meaningless garbage, including control characters, numbers, and other binary junk, wasting the user's time.*
5. Reproduced three out of three tries.
Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Reproduced with same steps using Wingdings and Symbol fonts.
On vague suspicion this was a formatting problem, saved file, closed SpeedyWriter and reopened file.
Garbage remained.
Saving file before changing font prevents bug.
Bug does not occur with existing files.
Only happens under Windows 98, not Solaris, Mac, or other Windows flavors.

 Copyright (c) 2000, Rex Black, All Rights Reserved

Review

- Each tester should submit each bug report to one or more test peers for a review
- Reviewing peers should:
  - Make suggestions to improve report
  - Ask clarifying questions
  - Even challenge "bugginess" if appropriate

△ Test team should only submit best possible bug reports, given time constraints appropriate to priority of bug

 Copyright (c) 2000, Rex Black, All Rights Reserved

Conclusions

- A bug report is a technical document
  - Accurate
  - Concise
  - Well-conceived
  - High-quality
- Instill tester focus on writing good reports
- Quality indicators for good bug reports
  - Clear to management (especially the summary)
  - Actionable by development
  - Move quickly from opened to closed

 Copyright (c) 2000, Rex Black, All Rights Reserved

ROI for Good Bug Reporting

- Improved Test team communication to senior and peer management
  - Can enhance credibility, standing, and resources
- Better tester/developer relationships
  - Helps developers be effective, reduces arguments
- Faster, more efficient bug lifecycles
  - Quicker fixes and fewer reopens

⇒Support increased product quality

 Copyright (c) 2000, Rex Black, All Rights Reserved

The Fine Art of
Writing a Good Bug Report

In a speech at Quality Week '99, Roger Sherman, a Microsoft test manager, identified the leading cause of bug report closure as "unreproducible." This is a regrettable circumstance, since such bug reports waste precious time during tight development schedules, add absolutely nothing to product quality, and lead to frustration and bad feelings between development engineers and test engineers. Sometimes, these bug reports arise from transient or random events, inconsistency of tools and configurations between test and development, or a vague definition of "correct" behavior under the tested conditions, but many bug reports closed as unreproducible are unclear, misleading, or just plain wrong.

Fortunately, I have learned some tricks for writing great bug reports that get management attention, communicate clearly to developers, and get fixed. Not only do these techniques provide solid technical payoffs in terms of a greater proportion of bugs fixed, they also communicate to development and to management that testers are serious about helping developers fix bugs. Writing bugs reports using these methods on projects I have managed, only around one out of eight of bug reports are closed without a fix.

Applying the following ten tips will help you achieve better bug reports:

1. Structure. A tester who uses a deliberate, careful approach to testing, and takes careful notes, tends to have a good idea of what's going on with the system under test. When failures occur, he knows when the first signs of failure manifested themselves.
2. Reproduce. The tester should check reproducibility of a failure before writing a bug report. If the problem doesn't recur, she should still write the bug report, but she must note the sporadic nature of the behavior. A good rule of thumb is three attempts to recreate the failure before writing the report. Documenting a clean set of steps to reproduce the problem addresses the issue of reproducibility head-on.
3. Isolate. After reproducing the failure, the tester should then proceed to isolate the bug. This refers to changing certain variables, such as system configuration, that may alter the symptom of the failure. This information gives developers a head start on debugging.
4. Generalize. After the tester has an isolated and reproducible case, he should try to generalize the problem. Does the same failure occur in other modules or locations? Can he find more severe occurrences of the same fault?
5. Compare. If a tester has previously verified the underlying test condition in the test case that found the bug, the tester should check these prior results to see if the condition passed in earlier runs. If so, then the bug is likely a case of regression, where a once-working feature now fails. Note that test conditions often occur in more than one test case, so this step can involves more work than just checking past runs of the same test case. Also, if you have a reference platform, repeat the test there and note result.

 Copyright © 2000, Rex Black, All Rights Reserved **rex_black@acm.org**

6. Summarize. The first line of the bug report, the failure summary, is the most critical. The tester should spend some time thinking through how the failure observed will affect the customer. This not only allows the tester to write a bug report that hooks the reader and communicates clearly to management, but also helps with setting bug report priority.
7. Condense. With a first draft of the bug report written, the tester should reread it, focusing on eliminating extraneous steps or words. Cryptic commentary is, of course, not the goal, but the report should not wear out its welcome by droning on endlessly about irrelevant details or steps which need not be performed to repeat the failure.
8. Disambiguate. In addition to eliminating wordiness, the tester should go through the report to make sure it is not subject to misinterpretation. Some words or phrases are vague, misleading, or subjective, and should be avoided. Clear, indisputable statements of fact are the goal.
9. Neutralize. Being the bearer of bad news presents the tester with the challenge of delicate presentation. Bug reports should be fair-minded in their wording. Attacking individual developers, criticizing the underlying error, attempting humor, or using sarcasm can create ill will with developers and divert attention from the bigger goal, increasing the quality of the product. The cautious tester confines her bug reports to statements of fact.
10. Review. Once the tester feels the bug report is the best one he can write, he should submit it to one or more test peers for a review. The reviewing peers should make suggestions, ask clarifying questions, and even, if appropriate, challenge the tester's assertion that the behavior is buggy. The test team should only submit the best possible bug report, given the time constraints appropriate to the priority of the bug.

A bug report should be an accurate, concise, thoroughly-edited, well-conceived, high-quality technical document. The test team needs to focus on the task of writing bug reports, and the test leads and manager must make it clear to each member of the test team that writing good bug reports is a primary job responsibility. Quality indicators for a well-tuned bug reporting process include:

- Clarity to management, particularly at the summary level;
- Utility to the development team, primarily in terms of giving the developer all the information needed to effectively debug the problem;
- Brevity of the bug lifecycle from opened to closed, reducing cycles where developers return poor quality reports for more information, leading to tester rework.

Improving the bug reporting process does require an effort, but provides significant payoffs. First, a crisp process improves the test team's communications with senior and peer management, which enhances the team's credibility and professional standing, and can encourage management to invest more resources in testing. Second, the smooth handoff to developers promotes positive relationships. Third, shorter bug lifecycles are more efficient, so the time invested up front writing a good bug report is repaid in time not wasted rewriting a poor bug report. These payoffs help the development process achieve better product quality through effective communication and efficient workflows.

 Copyright © 2000, Rex Black, All Rights Reserved **rex_black@acm.org**

QW2000 Vendor Technical Paper 8V1

Mr. Jim Bampos
(Vanteon)

Testing E-Commerce Applications

BACK TO QW2000 PROGRAM

Presentation Abstract

The Testing e- seminar addresses the needs and strategies for specialized testing of e-applications. Specific approaches to e-application verification are covered, including Load and Performance Testing, Usability Testing, Functional/Transaction Testing, Configuration Testing, and Automated Testing. Test strategies and testing complexities are identified for the e-application from Front-end to Back-end.

About the Author

As Vice President of e-Quality and Test Solutions, James Bampos is responsible for Vanteon's quality assurance (QA) and software testing services for development centers throughout the nation. Mr. Bampos manages the QA Outsourcing division, in addition to managing QA and testing activities in support of the company's eBusiness, wireless, commercial software, and hardware and embedded development solutions. Over the last three years, Mr. Bampos has grown the QA Outsourcing division by 1000%.

Mr. Bampos joined Vanteon in the Boston development center (formerly Turning Point Software) in 1995 and implemented best-of-breed SQA/Test methodologies, standards, and practices that made the organization a quality leader in the outsourcing industry. Mr. Bampos is involved with the Los Altos Workshop for Software Testing and the Software Testing Managers Roundtable - both periodic gatherings of senior practitioners in the QA industry. Mr. Bampos has been published in Software Testing and Quality Engineering magazine and frequently speaks at industry seminars and QA/Test conferences. Mr. Bampos has 14 years of experience in software development, quality assurance and management, and holds a Bachelor of Science degree in Computer Science from Massachusetts' Fitchburg State College.

BACK TO QW2000 PROGRAM

VANTEON™

Testing e-

Presented To

Quality Week
San Francisco, California
May - June 2000

Presented By

Jim Bampos
VP, e-Quality & Test Solutions

Lauri MacKinnon
QA Technical Leader

Overview

- Introduction
- The Need for Testing
- The Test Strategy
- Types of Testing
- e-Business Challenges
- Strategy for e-Quality
- Testing Complexities
- Test Result Interpretations

VANTEON™

Introduction

- Intent
- Who this applies to

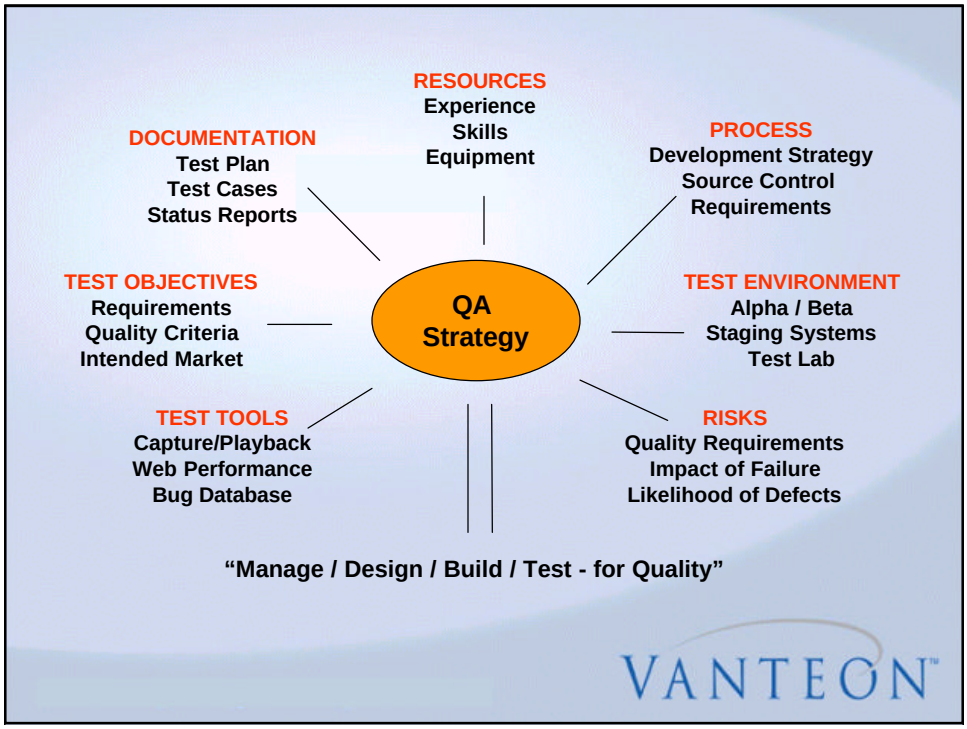
VANTEON™

The Need for Testing

A Situation:

- The Market
- Web Site Profile
- Target Clients
- Company Revenue
- PR Effort
- Problem Definition
- Data on Losses
- The Approach
- The Solution

VANTEON™



Testing e-

- **Stress & Performance Testing**
- **Usability Issues**
- **Configuration Testing**
- **Verification:**
  Is your site ready to go live?

VANTEON

Usability Testing

- Know your users
- Prioritize your goals
- Company goals and user goals
- Choose your principles
  - Learnable
  - Flexible
  - Efficient
  - Understandable
  - Robust
  - Pleasing / Satisfying

VANTEON™

And…

- Browser / Operating System Compatibility Testing
- Performance / Load Testing
- Content Testing
- Security Testing

VANTEON™

Go the Extra Mile…

- Load Testing
- Performance / Benchmark Testing
- Load Balancing
- Capacity / Stress Testing
- Uptime / Availability Testing
- Scalability Testing

VANTEON™

… or Two.

- Error Recovery Testing
- Web Site Optimization
- Bandwidth Verification
- Security Testing
- Market Impact Anticipation
- Competitive & Industry Benchmarks

VANTEON™

Automated Testing

- Feature
- Acceptance
- Regression
- Performance
- Load

VANTEON™

Why is e-Business different?

- e-Business systems are continuously changed based on market conditions [1]
- e-Business systems are designed in a component fashion to facilitate this rapid change [2]
- Fundamentally, e-Business systems are continuously being assembled, validated and deployed [3]

(1,2,3 Courtesy of Segue Software)

VANTEON™

- **Unlike traditional C/S and Legacy Systems**
  - **e-Business systems are not developed only by traditional engineering groups**[4]
  - **e-Business systems are worked on by many different groups (Content Designers, Web Masters, Domain Experts, DBA's, Software Engineers & QA)**[5]
  - **Fundamentally, e-Business systems are designed and developed (and thus owned) by all departments in the enterprise**[6]

(4,5,6 Courtesy of Segue Software)

VANTEON™

Strategy for e-Quality

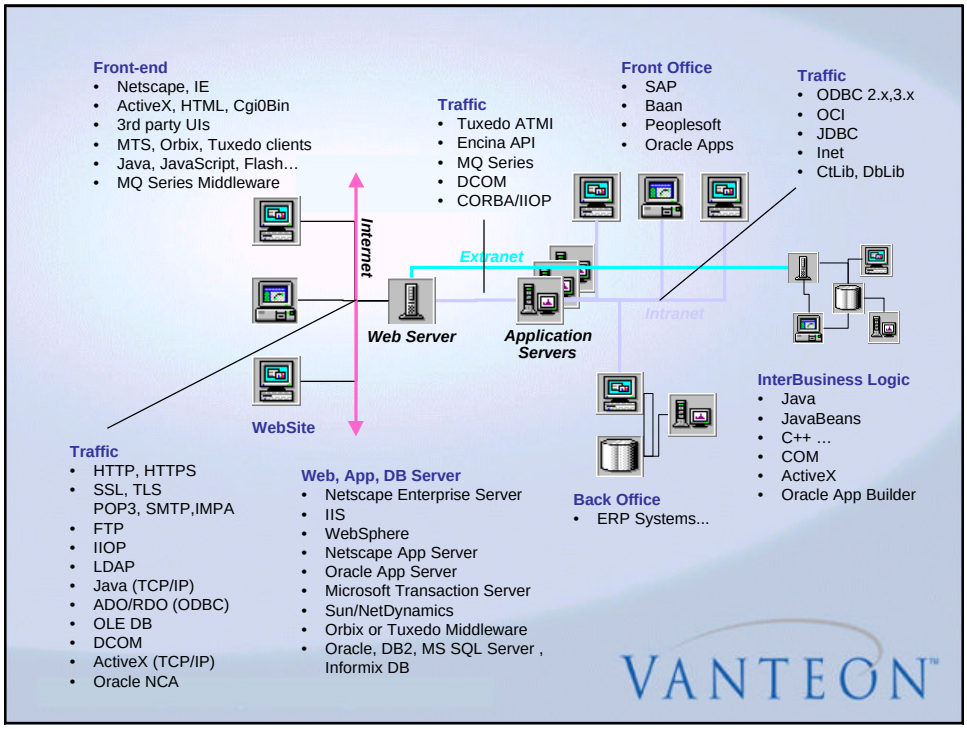
WRQE - a place to start

End to End testing - and everything in between:

	Scenario Tests	Component Tests
Web Site Content Links Online Activity	Transaction Accuracy	Functional Testing
Front Office Transaction Processing Online Transactions	Transaction Volumes	Web Server Load Testing
Back Office Order Processing InterBusiness Service/Suppliers	Timing Bottlenecks	Database Load Testing
Business to Business Internal Corporate Processes	Concurrency Bottlenecks Resource Contentions	Unit and System Testing

(Graph Courtesy of Segue Software)

VANTEON™



Testing Complexities

- **Performance**
  - Multiple protocols (proprietary and standard)
  - Multiple servers and server applications
  - Web Server - App Server Interaction
  - Database Integrity
  - Bandwidth Restrictions
  - Synchronous Transactions
  - Unique/Randomly Generated Data
  - Distributed Java Applets on JVM's; EJB
- **Usability**
  - Permutations of User Configurations (Hardware & Software)
  - Handling Novice and Expert Users
  - Verifying Site Stability
- **Configuration**
  - So Many Permutations
  - Representative Sampling
  - Using Orthogonal Arrays
- **Ready to go Live**
  - The Checklist… Are you ready?

VANTEON

Conclusions

Questions

&

Answers

VANTEON™

QW2000 Paper 8T1

Mr. Atif Memon, Dr. Martha E. Pollack, & Dr. Mary Lou Soffa
(University of Pittsburgh)

A Planning-Based Approach to GUI Testing

BACK TO QW2000 PROGRAM

Presentation Abstract

Graphical user interfaces (GUIs) have become nearly ubiquitous as a means of interacting with software systems. The widespread use of GUIs is leading to the construction of more and more complex GUIs. With the growing complexity comes challenges in testing the correctness of GUIs and the underlying software. Some of the important challenges include test¡case generation, test¡ oracle creation, and regression testing.

Current GUI testing practices involve a significant amount of manual effort on the part of the test designer. Most test designers employ tools that automate certain aspects of the testing cycle. Most common among such tools are capture/playback tools used to capture the user events and GUI screens during an interactive session. The recorded sessions are later played back whenever it is necessary to recreate the same GUI states. These tools generally store information at a low level of abstraction, capturing actual mouse positions, button clicks and storing bit¡maps. Representing the information at such a low level of abstraction makes it difficult to tailor the recorded session for other test cases. A popular alternative to using capture/playback tools is to program a test case generator. Programming requires that the test designer program all possible decision points present in the GUI. However, this approach is time consuming, and is susceptible to missing important GUI decisions. Moreover, the expected output must also be determined by the programmer.

Planning Assisted Tester for grapHical user interface Systems (PATHS) is a research project designed with the primary goal of facilitating the automation of GUI testing. PATHS is based on planning -- a well developed and used technique in Artificial Intelligence (AI). Given a set of operators, an initial state and a goal state, a planning system produces a sequence of operators that will transform the initial state to the goal state. The key idea of using planning as the core of PATHS is that the GUI test designers will often find it easier to specify typical goals that users of the GUI software might have than to specify sequences of GUI actions that users might perform to achieve those goals. Thus we cast GUI testing as an instance of planning. PATHS has the goals of generating test cases automatically, incorporating oracle information into the testing process, and then automatically generating a regression test suite when re¡testing is done.

Currently, the PATHS system is capable of using GUI specifications to automatically generate a test suite, exploit the planning model to create test oracles, execute the test cases and pinpoint errors in the GUI. A test designer provides PATHS with a description of the GUI (e.g., in the form of GUI speci¡

fications). PATHS analyzes this description and creates an abstract model of the GUI and returns a list of operators from the model to the test designer. By using knowledge of the GUI, the test designer then defines the preconditions and effects of the operators in a simple language provided by the planning system. The test designer then triggers test¡case generation and oracle creation by describing scenarios (tasks), represented by a set of initial and goal states. PATHS generates a test suite for the scenarios. The test suite also has the oracle information built into it. During testing, the GUI exerciser uses this oracle information to compose the expected GUI state. The GUI exerciser compares the expected GUI state with the actual state to determine if the GUI's run¡time behavior is correct. After testing, identified problems in the GUI are corrected. Modifi¡ cations then lead to regression testing. The test designer describes the changes in the GUI, and PATHS generates the regression test suite.

The key to efficient test¡case generation, oracle creation and maintenance is PATHS's hierar¡ chical decomposition of the GUI. PATHS models a GUI hierarchically with high¡level operators that decompose into sequences of lower level ones. More specifically, PATHS begins the modeling process by partitioning the GUI events into several classes: (1) Menu¡open events that open menus, i.e., they expand the set of GUI events available to the user, (2) Modal events that open modal windows (i.e., no other window in the program can be interacted with until this dialog is closed), (3) Non¡modal events that open non¡modal windows, and (4) System¡interaction events that interact with the underlying software to perform some action. These event classes are then used by PATHS to create two types of planning operators -- system¡interaction operators and abstract operators. The system¡interaction operators are derived from the system¡interaction events. Abstract operators are more complex and are responsible for driving the hierarchical planning algorithms used by PATHS. The basic idea behind an abstract operator is that it represents a sequence of GUI events that invoke a modal window. The abstract operator encapsulates the events performed on the modal window by treating the interaction within that window as a separate planning problem. Abstract operators are decomposed into lower level operators by an explicit call to the planner. PATHS uses the system¡interaction and abstract operators to generate the test suite.

PATHS automatically generates test oracles to check the GUI's state during and after test execution. The key to oracle creation is to have oracle information integrated into the test case so that the GUI exerciser knows what to check at each step. One of the primary motivations of using planning for GUI testing is that much of the state can be recovered directly from the planning model. During plan construction, the planner keeps track of the intermediate states of the GUI. PATHS extracts these intermediate states and integrates them with the generated test case to act as oracles.

We are currently extending PATHS with efficient regression testing techniques that draw on planning. The hierarchical decomposition of the GUI done by PATHS allows identification and isolation of GUI components that were modified. Changes are then made only to the effected test cases and oracle information. Moreover, hierarchical planning aids in retaining most of the test cases and oracle information defined at the higher levels of abstraction. Changes are made only at the lower levels.

About the Author

Atif M. Memon is a Mellon Fellow at the Department of Computer Science, University of Pittsburgh. He received his BS and MS in Computer Science in 1991 and 1995 respectively. He was awarded a Gold Medal for First Position in BS. He enrolled at the University of Pittsburgh in 1996 and is currently a

Ph.D. candidate. In 1999, he was awarded a Fellowship from the Andrew Mellon Foundation for his Ph.D. research. His research interests include program testing, software engineering, artificial intelligence, plan generation, and code improving compilation techniques. He is a member of the ACM and the IEEE Computer Society.

More information can be found at the Author's Home-page

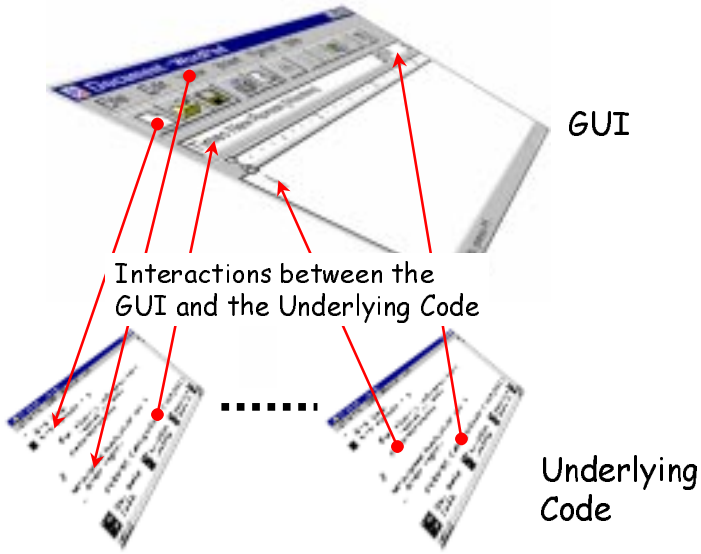
BACK TO QW2000 PROGRAM

A Planning-based Approach to GUI Testing

Atif M. Memon
Martha E. Pollack
Mary Lou Soffa

Dept. of Computer Science
University of Pittsburgh

Testing a GUI



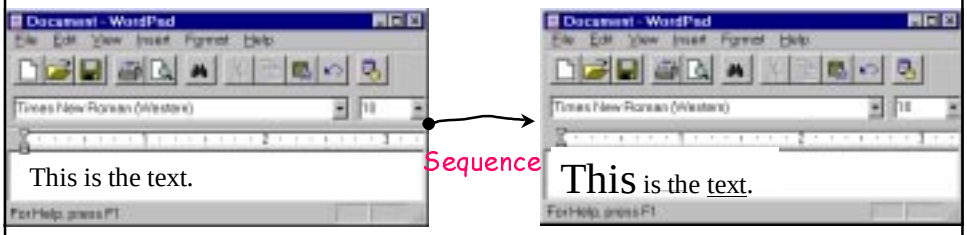
Problems with GUI Testing

- Generating Test Cases
- Verification
- Providing Coverage Criteria
- Performing Regression Testing

What is a GUI Test Case ?

- Individual User Events
  - NOT ENOUGH !
  - Sequences of User Events lead to Different States
- Test Case: Sequence of User Events

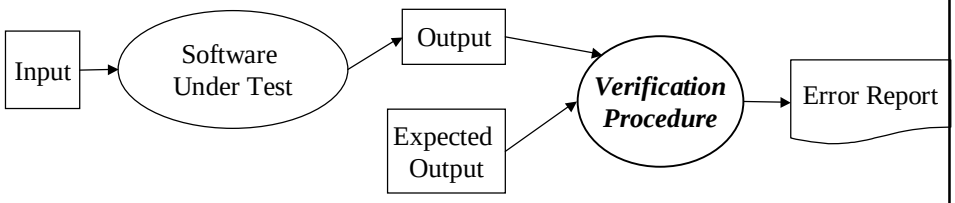
Generating Test Case Sequences

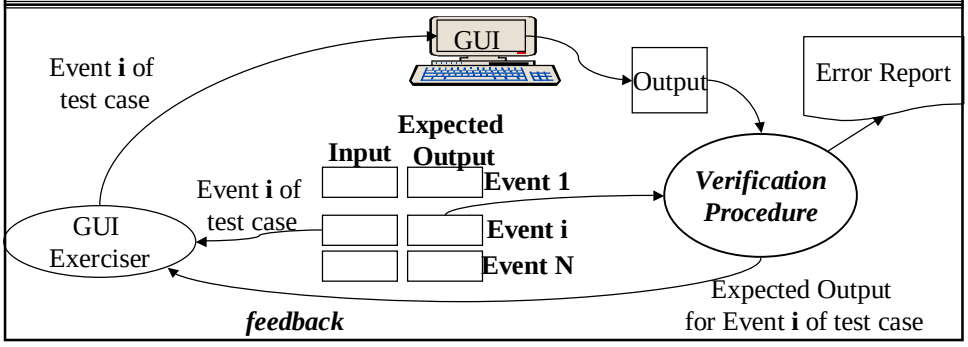


Each Sequence Transforms the GUI State

- Randomly Choose Sequences
- Expert Chooses Sequences
- Generate Sequences from a Description of the GUI

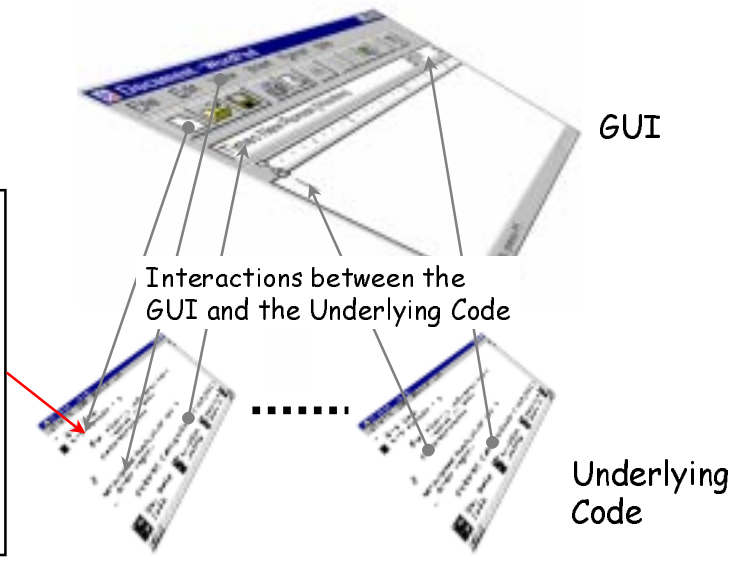
Verification





Problems

Coverage Criteria



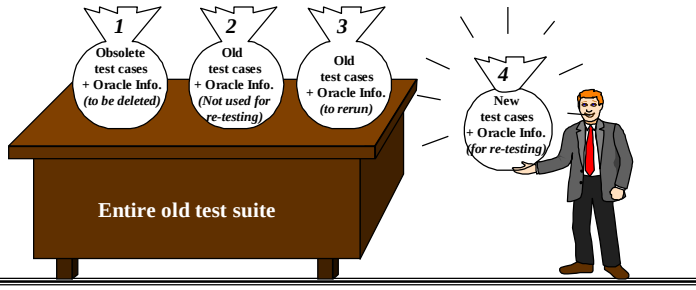
Most Traditional Coverage Criteria Work Here

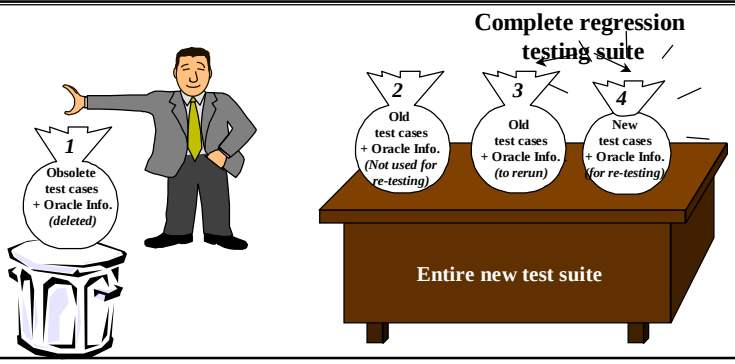
OR

at the Specification Level

Problems

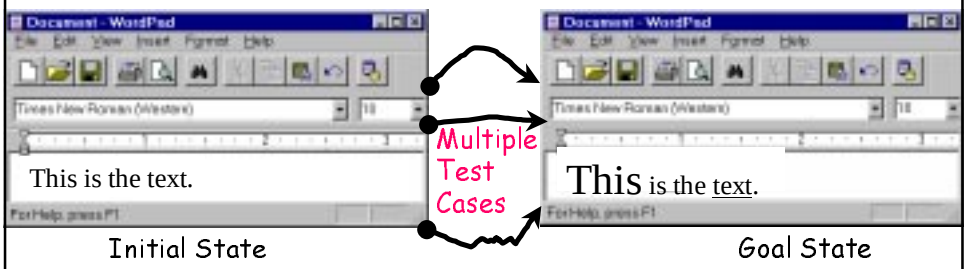
Regression Testing





Novel Idea

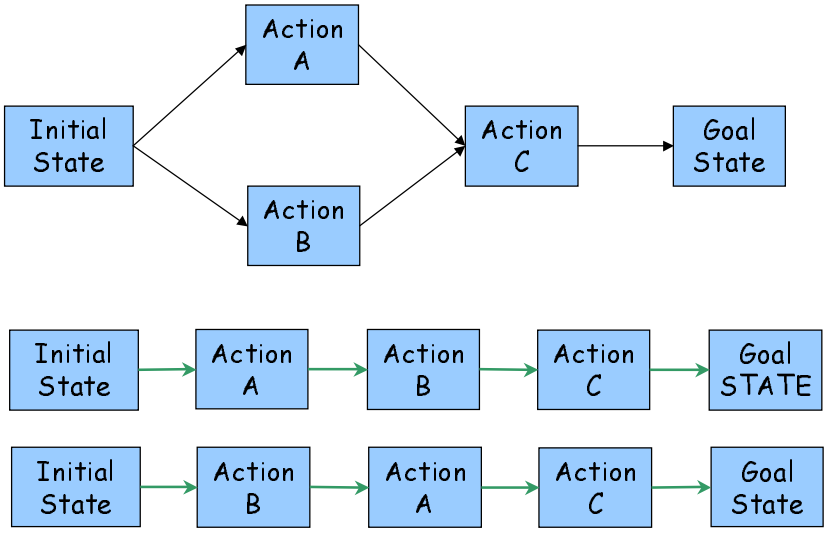
- Automatically Generate Test Cases for EXPECTED TASKS



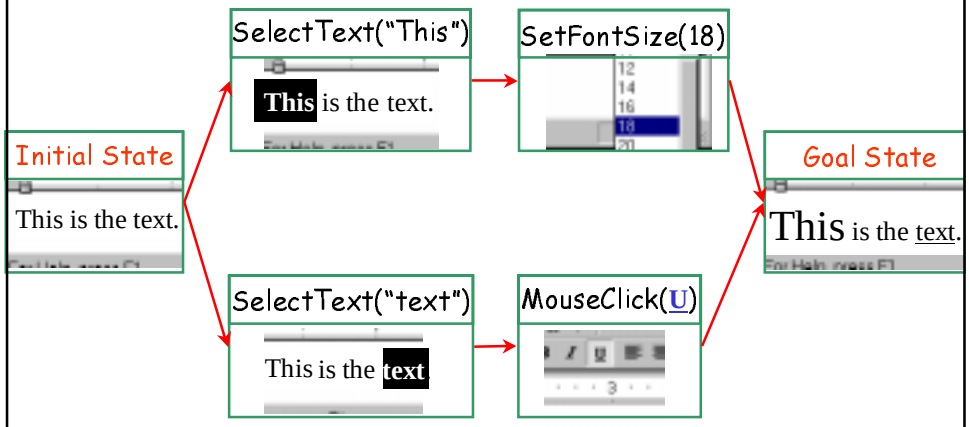
AI Planning and GUI Testing

- Recent Advances in AI Planning Propositional Planners
  - Very Fast
  - Based on
    - Flow-Graphs [Koehler et al. '97]
    - SAT Solving [Kautz & Selman '96]
- AI Planning, Especially Suitable for
  - Generating Test Cases
  - Automatically Generating Verification Information
  - Regression Testing

A Plan



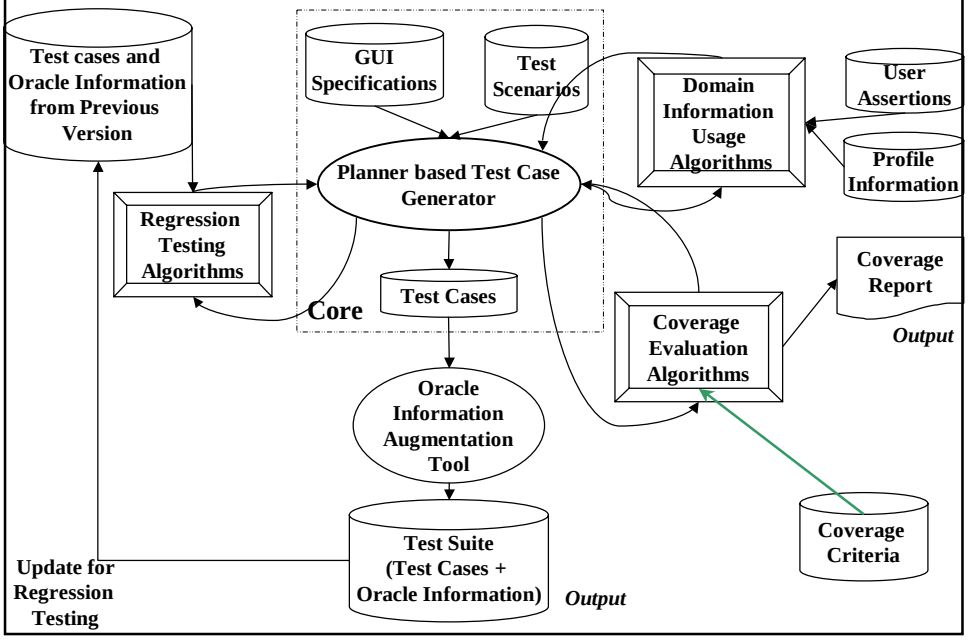
A Plan for a GUI Task



Goals of Designing PATHS

- Features of PATHS
  - Test Case Generation
  - Verification
  - Coverage
  - Regression Testing
- Requirements of PATHS
  - Automated
  - Efficient
  - Robust
  - Portable
  - General

Overview of PATHS

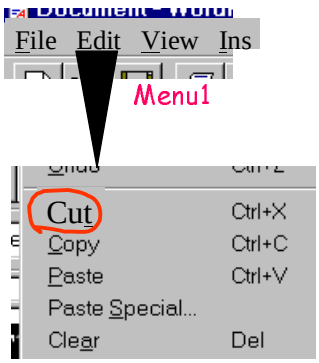


Design Steps

- Develop an Efficient Representation of the GUI
- Choose/Design Planning System
- Use Planning System and Representation to
  - Generate Test Cases
  - Define and Create GUI Test Oracles
  - Design Coverage Criteria for GUIs
  - Develop Techniques for GUI Regression Testing
  - Exploit Domain Information

Straightforward Approach

- Define One Operator for each User Action



Operator :: *CUT*
Preconditions:
isCurrent(Menu2).

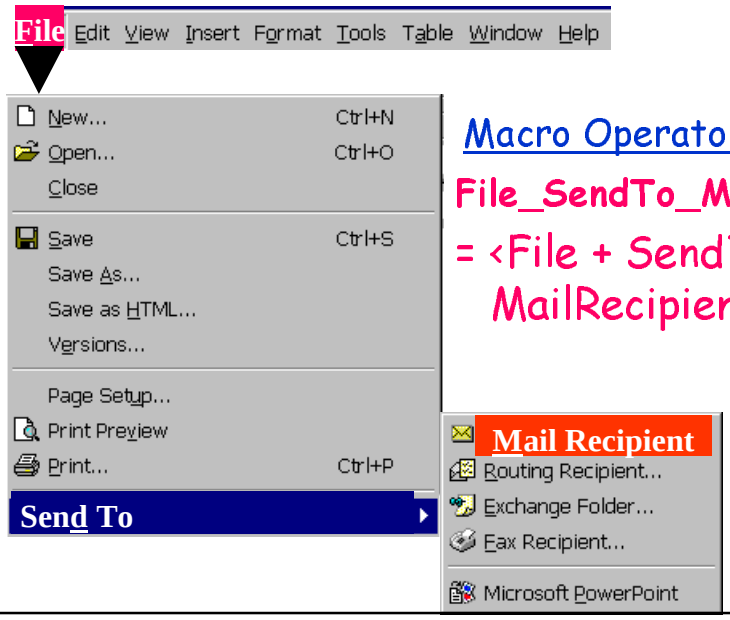
Effects:
FORALL Obj in Objects
Selected(Obj) ⇒
ADD inClipboard(Obj)
DEL onScreen(Obj)
DEL Selected(Obj)
ADD isCurrent(Menu1)
DEL isCurrent(Menu2).

First Order Predicate Logic

Exploit the GUI's Structure

- Reduce the Number of Operators
  - System more Efficient
  - Easier for the Test Designer
- Two Types of Abstractions
  - Combine Buttons ⇒ Create Macro Operators
  - Decompose GUI Hierarchically ⇒ Create Abstract Operators

Create Macro Operators



Macro Operator:

File_SendTo_MailRecipient

= <File + SendTo + MailRecipient >

Create Abstract Operators

Straightforward Approach

Main GUI's Operator Set

...
Set Language
SelectFromList()
Default
OK
Cancel
...

Using Abstraction

Main GUI's Operator Set

...
Set Language
...

Language Window's Operator Set

SelectFromList()
Default
OK
Cancel

Create Abstract Operators

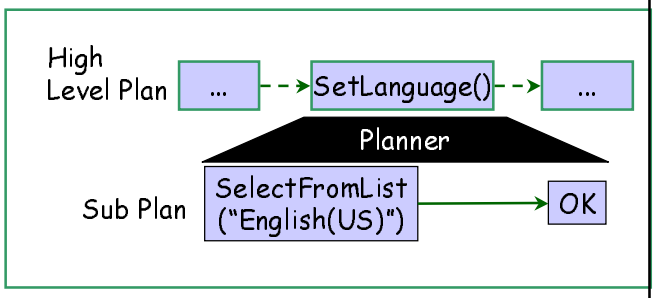
Language Window's Operator Set

SelectFromList()
Default
OK
Cancel

Define Abstraction

SetLanguage()

Abstract Operator



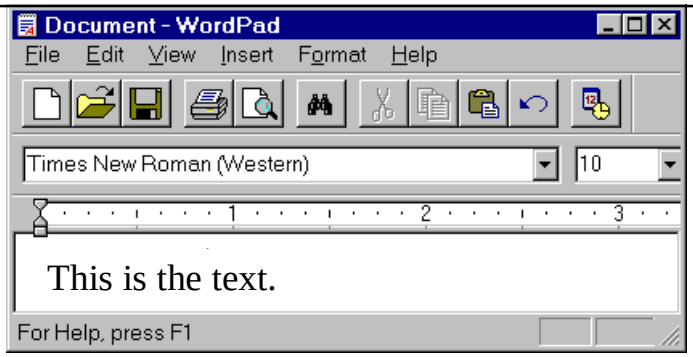
Effects of Exploiting the GUI's Structure

- Reduction in Planning Operators
  - 325 operators ⇒ 32 operators
  - Ratio 10:1 for MS WordPad
- System Automatically Determines the Macro and Abstract Operators

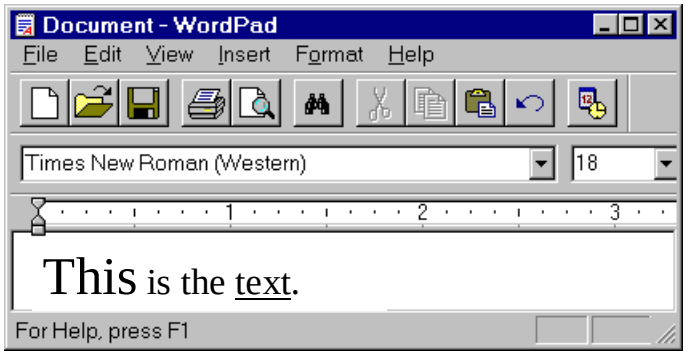
Overview of Test Generation

Phase	Step	Test Designer	Automatic Planning-based System
Setup	1		Derive Planning Operators from GUI
	2	Code Preconditions and Effects of Operators	
Test Case Generation	3	Specify a Task (Initial and Goal States)	
	4		Generate Test Cases

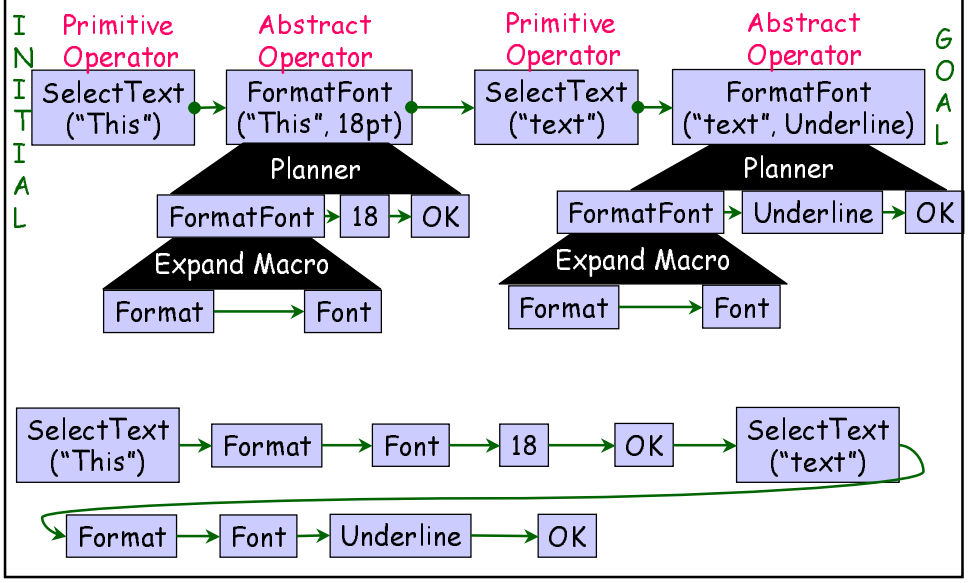
Initial State



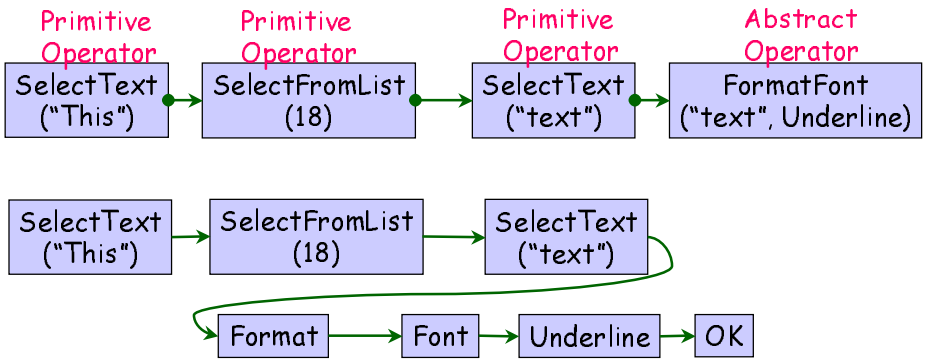
Goal State



Test Case



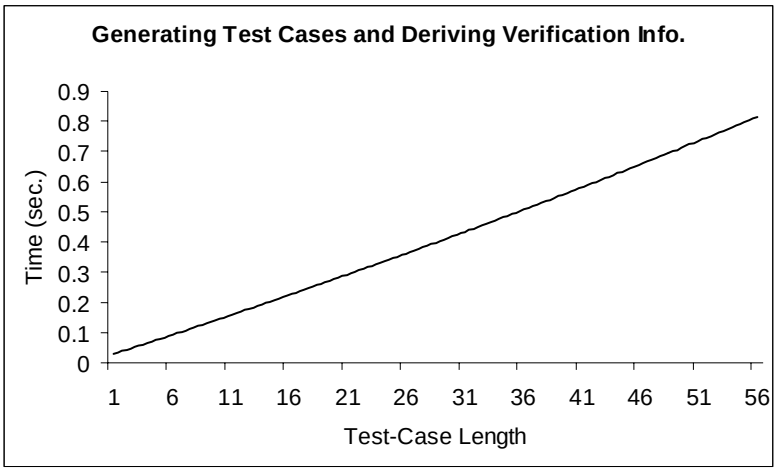
Alternative Test Case



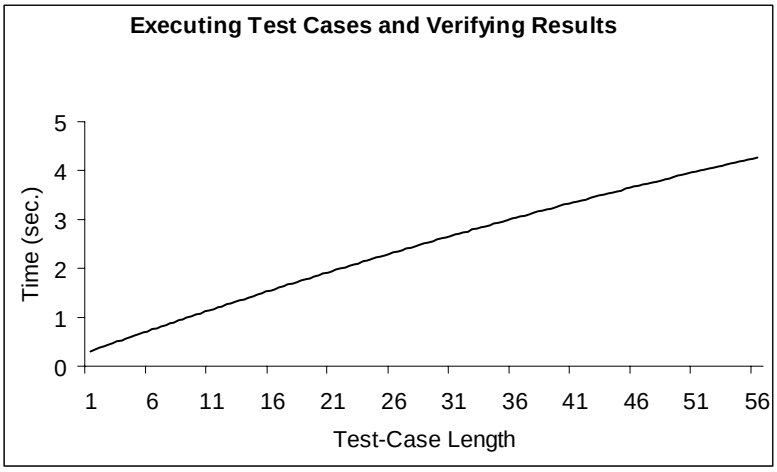
Methods to Generate Alternative Test Cases

- Different Results from Planner
- Abstract Operator Decompositions
- Linearizations of the Partial-order Plan

Experimental Results



Experimental Results (contd…)



Test Oracles & Regression Testing

- Exploit Planning Algorithms to get Expected Behavior of GUI
- Use Hierarchical GUI Model for Efficient Regression Testing

Related Work

- GUI Testing
  - FSM [Esmelioglu and Apfelbaum] and VFSM [Shahady and Siewiorek] Models.
  - Genetic Algorithm Technique [Kasik and George]
  - Visual TDE for GUIs [Foster, Goradia, Ostrand, and Szermer]
- Planning for Testing
  - [Adele Howe, Anneliese Von Mayrhauser, Richard Mraz in ASE '97]

Concluding Remarks

- Automatic Planning is a Feasible Approach for GUI Testing
- Automatic Generation of Preconditions and Effects from GUI Specifications

A Planning-based Approach to GUI Testing*

Atif M. Memon†, Martha E. Pollack, Mary Lou Soffa
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260 USA
+1 412 624-8850
{atif, pollack, soffa}@cs.pitt.edu

Abstract
Graphical user interfaces (GUIs) have become nearly ubiquitous as a means of interacting with software systems. The widespread use of GUIs is leading to the construction of more and more complex GUIs. With the growing complexity comes challenges in testing the correctness of GUIs and the underlying software. Some of the important challenges include test-case generation, test-oracle creation, and regression testing. In this paper, we present the design of Planning Assisted Tester for grapHical user interface Systems (PATHS) – a research project designed with the primary goal of facilitating the automation of GUI testing. PATHS uses a new GUI testing technique based on user event interaction sequences. The key idea is to test the GUI software using interactions most likely to be exercised in actual use. A novel feature of PATHS is its reliance on AI plan generation techniques to generate testing information. Given a set of operators, an initial state and a goal state, a planning system produces a sequence of operators that transforms the initial state to the goal state. Using PATHS, GUI test designers can generate likely user interaction sequences by specifying typical goals that users of the GUI software might have. PATHS first analyzes the GUI and derives hierarchical planning operators from the actions in the GUI. The test designer determines the preconditions and effects of the hierarchical operators, which are then input into a planning system. With the knowledge of the GUI and the way in which the user will interact with the GUI, the test designer creates sets of initial and goal states. Given these initial and final states of the GUI, a hierarchical planner produces plans, or a set of test cases, that enable the goal state to be reached. Our technique has the additional benefit of associating oracle information with the test cases automatically. We implemented our technique by developing the GUI analyzer and extending a planner. We generated test cases for Microsoft's WordPad to demonstrate the viability and practicality of the approach.

Keywords
GUI testing, application of planning, GUI regression testing, automated test-case generation,

1 Introduction

Testing is a critical component of the software development process and is required to ensure the safety, robustness and usability of software. Unfortunately, it is also labor and resource intensive, accounting for 50%-60% of the total cost of software development [9, 29]. Hence, there has been significant research aimed at automating the testing process. Although some success has been achieved, many problems remain. In particular, it is not yet clear how to automate the testing of user interfaces, which constitute an increasingly large portion of software systems, as much as 45-60% of the total software code [21, 25]. The most popular form of user interfaces are direct-manipulation interfaces called Graphical User Interfaces (GUIs) [26]. GUIs have become an important and accepted way of interacting with today's software. As GUIs become more and more popular, they are increasingly being used in critical systems [35] and testing them is necessary to avert catastrophes [26].

Testing the correctness of a GUI is difficult for a number of reasons. First, the space of possible interactions with a GUI is enormous. Each sequence of GUI actions can result in a different state of the combined system (i.e., the GUI and underlying software). In general, a GUI action might have different results in each state, and thus need to be tested in a very large number of states: the amount of testing required can be enormous [34]. Related to this is the fact that measures of coverage that have been defined for testing conventional software systems do not work well for GUIs. For conventional software, coverage is measured using the amount and type of underlying code exercised. In testing GUIs, while one must still be concerned with how much of the code is tested, there needs also to be significantly increased focus on the number of different possible states

* *Partially supported by the Air Force Office of Scientific Research (F49620-98-1-0436) and by the National Science Foundation (IRI-9619579) (EIA0906525).*
† *Partially supported by the Andrew Mellon Pre-doctoral Fellowship, awarded by the Andrew Mellon Foundation.*

in which each piece of code is exercised. Existing metrics do not allow one to say whether a GUI has been "well-enough" tested. As a result, GUI testing often relies on extensive beta testing: for example, Microsoft released almost 400,000 beta copies of Windows95 targeted at finding program failures [15].

GUI testing involves several steps. Initially, a set of test cases must be generated. This is particularly challenging for GUI testing, because of the difficulties mentioned above: the set of possible test cases is huge, and conventional metrics for selecting "good" test case sets do not apply. After test cases are constructed, they must be executed: this is when the actual "testing" occurs, to check whether the GUI is performing correctly. An incorrect GUI state can lead to an unexpected screen, making further execution of the test case useless because events in the test case might not match the corresponding GUI components on the screen. Consequently, the execution of the test case must be terminated as soon as an error is detected. Verification checks, performed by using test oracles, must therefore be inserted after each step, to catch errors as soon as they occur. Yet another challenge is posed by regression testing, i.e., updating the set of test cases and the verification check after changes are made to the GUI during development or maintenance. Regression testing presents special challenges for GUIs, because the input-output mapping often does not remain constant across successive versions of the software [24].

Current GUI testing practices involve a significant amount of manual effort on the part of the test designer. Most test designers employ tools that automate certain aspects of the testing cycle. Most common among such tools are *capture/playback tools* [13, 11] used to capture the user events and GUI screens during an interactive session. The recorded sessions are later played back whenever it is necessary to recreate the same GUI states. These tools generally store information at a low level of abstraction, capturing actual mouse positions, button clicks and storing bit-maps. Representing the information at such a low level of abstraction makes it difficult to tailor the recorded session for other test cases. A popular alternative to using capture/playback tools is to program a test case generator. Programming requires that the test designer program all possible decision points present in the GUI. However, this approach is time consuming, and is susceptible to missing important GUI decisions. Moreover, the expected output must also be determined by the programmer.

In this paper, we present the design of Planning Assisted Tester for grapHical user interface Systems (PATHS) – a research project designed with the primary goal of facilitating the automation of GUI testing. PATHS is based on **planning** – a well developed and used technique in Artificial Intelligence (AI). Given a set of operators, an initial state and a goal state, a planning system produces a sequence of operators that will transform the initial state to the goal state. The key idea of using planning as the core of PATHS is that the GUI test designers will often find it easier to specify typical goals that users of the GUI software might have than to specify sequences of GUI actions that users might perform to achieve those goals. Thus we cast GUI testing as an instance of planning. PATHS has the goals of generating test cases automatically, incorporating oracle information into the testing process, and then automatically generating a regression test suite when re-testing is done.

The main contributions of PATHS are as follows.

- Most of the GUI testing tasks are automated so that the test designer's work is simplified as much as possible.
- The overall testing cycle defined by PATHS is efficient since software testing is usually a tedious and expensive process.
- PATHS is robust in that whenever the GUI enters an unexpected state, the testing algorithms detect the error state immediately, recover from it and report all information necessary to debug the GUI.
- The testing information generated by PATHS is portable. Test information (e.g., test cases, oracle information, coverage report, error report) generated and/or collected on one platform is usable on all other platforms on which the GUI can be executed.
- Finally, PATHS is general enough to be applied to a wide range of GUIs.

In the next section, we present a high-level overview of the design of PATHS. Section 3 presents a discussion of AI planning. In Section 4, we show how PATHS automatically models the GUI hierarchically so that a restricted form of hierarchical planning can be applied to efficiently generate testing information. In particular, we show in Section 5 how the hierarchical model is used to generate test cases. In Section 6, we also indicate how the hierarchical model is used to create test oracles and for regression testing. We present a discussion on related work in Section 7 and conclude in Section 8.

2 Overview of PATHS

PATHS uses a new GUI testing technique based on user event interaction sequences. The key idea is to test the GUI software using interactions most likely to be exercised in actual use. The primary function of PATHS is

to generate likely user interaction sequences and then to test the GUI using these sequences (test cases) as input. A novel feature of PATHS is its reliance on AI plan generation techniques to generate test cases. The central component of PATHS is a planning based test case generator. In addition to planning algorithms, PATHS is supplemented with techniques and algorithms for effective regression testing, coverage evaluation, and incorporation of domain specific knowledge. The test case generator is given a description of the GUI and test scenarios consisting of pairs of initial and goal states as input, and it generates test cases as output. The GUI description is assumed to provide a complete working of the GUI. An oracle information augmentation tool associates additional information with each test case to be used to verify the state of the GUI during test case execution. Currently, the PATHS system is capable of using GUI specifications to automatically generate a test suite, exploit the planning model to create test oracles, execute the test cases and pinpoint errors in the GUI. Various measures for coverage are undergoing development. The coverage evaluation algorithms will measure the quality of the generated test cases. The regression testing algorithms will use results from prior testing sessions to guide regression testing. Additional user supplied domain information will be used for increased efficiency and effectiveness. Actual execution of the test cases will be done by a GUI exerciser. The high-level interactions between the components of PATHS are shown in Figure 1. The ovals represent the processes that control the test case generation and execution. The outputs include the final test suite, coverage report, and the error report. Other entities are either generated by the components or provided as input by the test designer.

3 AI Plan Generation

PATHS makes use of planning for GUI testing. This section gives a brief introduction to planning and the different planning techniques.

Automated plan generation has been widely investigated and used within the field of artificial intelligence. Given an initial state, a goal state, a set of operators, and a set of objects, a planner returns a sequence of actions (instantiated operators) to achieve the goal. Many different algorithms for plan generation have been proposed and developed. Weld presents an introduction to least commitment planning [32] and a survey of the recent advances in planning technology [33].

Formally, a *planning problem* $P(\Lambda, D, I, G)$ is a 4-tuple, where Λ is the set of operators, D is a finite set of objects, I is the initial state, and G is the goal state. The solution to the planning problem is a plan: a tuple $< S, O, L, B >$ where S are steps (instances of operators – typically represented as sets of preconditions and effects), O are ordering constraints on the elements of S, L are causal links representing the causal structure of the plan, and B are binding constraints on the variables in S. Causal links are triples $< S_i, c, S_j >$, where S_i and S_j are elements of S and c is both an effect of S_i and a precondition for S_j. Typically, the ordering constraints only induce a partial ordering, so the set of solutions are all linearizations of S consistent with O.

The output of the planner is a set of actions with certain constraints on the relationships among them. An action is an instance of an operator with its variables bound to values. One well-known action representation uses the STRIPS[1] language [8] which specifies operators in terms of parameterized preconditions and effects. STRIPS was developed more than twenty years ago, and has limited expressive power. For instance, no conditional or universally quantified effects are allowed. Although, in principle, sets of STRIPS operators could be defined to encode conditional effects, such encodings lead to an exponential number of operators making even small planning problems intractable. A more powerful representation is ADL [28, 27], which allows conditional and universally quantified effects in the operators. This facility makes it possible to define operators in a more intuitive manner. A more recent representation is the Planning Domain Definition Language[2] (PDDL), used in the AIPS'98 planning competition. The goals of designing the PDDL language were to encourage empirical evaluation of planner performance, and the development of standard sets of planning problems. The language has roughly the expressiveness of ADL for propositions.

Recently developed planning technology based on propositionalization of the search space has greatly increased the efficiency of plan generation. A well-known planner based on this technology is the Interference Progression Planner (IPP) [19], a system which extends the ideas of the Graphplan system [2] for plan generation. Graphplan introduced the idea of performing plan generation by converting the representation of a planning problem into a propositional encoding. Plans are then found by means of a search through a leveled graph, in which *even levels* $(0, 2, \ldots, i)$ represent all the (grounded) propositions that might be true at stage i of the plan, and *odd levels* $(1, 3, \ldots i+1)$ represent actions that might be performed at time $i+1$. The planners in the Graphplan family, including IPP, have shown increases in planning speeds of several orders of magnitude on a wide range of problems compared to earlier planning systems (but *cf.* [22]).

IPP uses ADL for the representation of actions in which preconditions and effects can be parameterized: subse-

[1] STRIPS is an acronym for STanford Research Institute Problem Solver

[2] Entire documentation available at `http://www.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz`

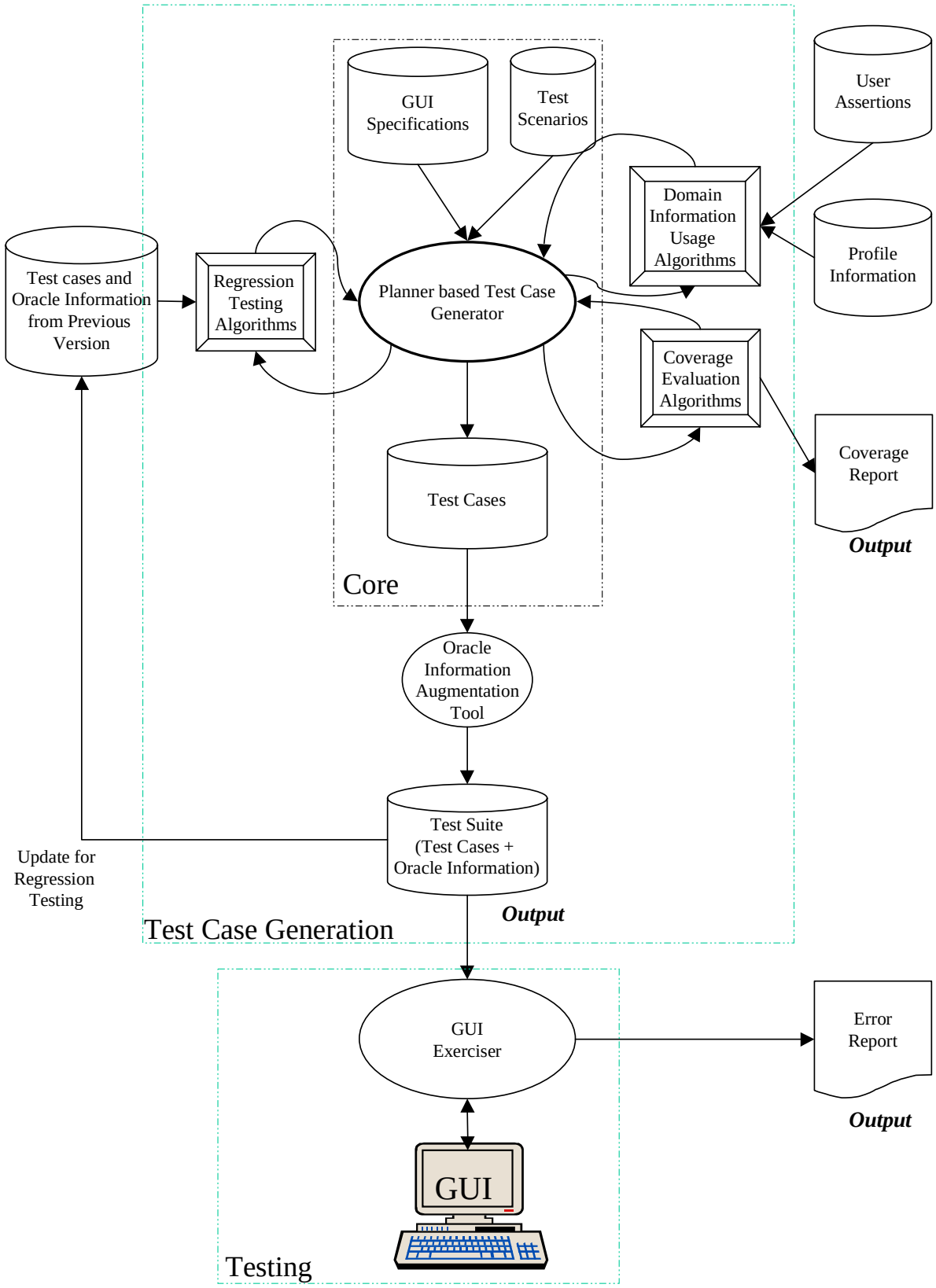


Figure 1: The Components of PATHS.

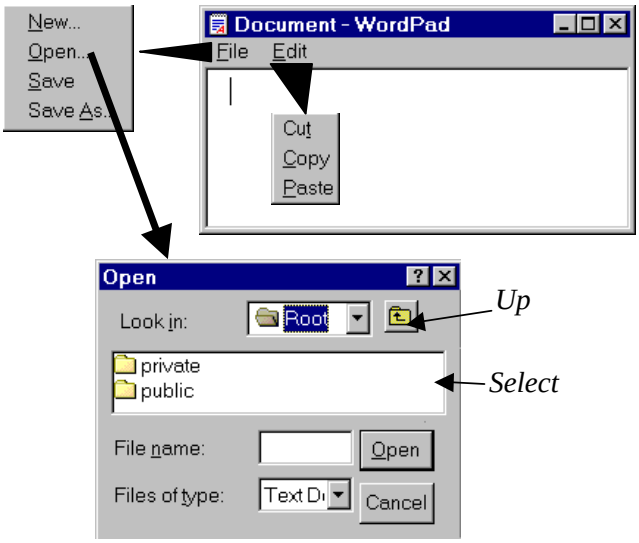


Figure 2: The Example GUI.

quent processing does the conversion to propositional form. In fact, IPP generalizes Graphplan precisely by increasing the expressive power of its representation language, allowing for conditional and universally quantified effects. As is common in planning, IPP produces *partial-order plans*.

Planning at one level of abstraction may be impractical for complex systems which consist of a large number of objects and operators. Techniques have been developed to generate plans at multiple levels of abstraction; this is typically called

QW2000 Paper 8A1

Dr. John Musa
(Consultant)

A Good Idea! But How Do We Get People To Use It?

BACK TO QW2000 PROGRAM

Key Points

- Process improvement
- Organizational change
- Technology transfer

Presentation Abstract

The acceleration of innovation in software development and testing to internet time means that every software professional will be more and more frequently faced with the need to introduce and deploy new technology. This talk will examine how to go about it. It will cover the stages of new technology deployment, the approaches that work and don't work, the pros and cons of various methods for actually transferring technology, and how to handle some of the problems one commonly encounters. It will bring out specific examples from the speaker's extensive experience as a technology transfer manager and as a teacher and consultant.

About the Author

John D. Musa is an independent consultant. He was formerly Technical Manager of Software Reliability Engineering (SRE) at AT&T Bell Laboratories, Murray Hill, NJ. He has varied and extensive experience as a software developer and software development manager.

John Musa is one of the creators of the field of software reliability engineering (SRE). He is widely recognized as the leader in its practical application. He has been involved for some 25 years in deploying the practice, first within AT&T and afterwards worldwide through his consulting work. His new book "Software Reliability Engineering: More Reliable Software, Faster Development and Testing" (McGraw-Hill) was written to help disseminate SRE.

The IEEE elected him Fellow in 1986 for his many seminal contributions. He was recognized in 1992 as the person who had contributed the most to testing technology. His leadership has been noted by every recent edition of Who's Who in America and American Men and Women of Science. He is an experienced international speaker and teacher (over 200 major presentations) with consistently outstanding feedback.

(John Musa's Home Page)
(j.musa@ieee.org

A Good Idea! But How Do We Get People To Use It?

John D. Musa
j.musa@ieee.org

1

Sources of Talk

1. 25+ yr of introducing software reliability engineering technology
2. 10+ yr of managing new technology introduction groups at AT&T Bell laboratories
3. Extensive observation from editorial board and program committee perspectives
4. Software Engineering Institute Metrics Steering Committee

2

DEPLOY - 2

Copyright John D. Musa 2000

Example for Talk

software reliability engineering (SRE): a practice for quantitatively planning and guiding software development to make it faster and cheaper, while ensuring that product reliability and availability meet user needs[1,2,3]. SRE:

1. Quantitatively characterizes the product's expected use and focuses resources on the most used and most critical functions.
2. Sets quantitative objectives for reliability and/or availability as well as schedule and cost and engineers strategies to meet the objectives.
3. Tracks reliability in system test against its objective as one of the release criteria.

DEPLOY - 3

3

Copyright John D. Musa 2000

Stages of Introduction

1. Pilot the idea
2. Test sphere of application
3. Seek endorsement (best practice)
4. Raise awareness and interest
5. Implement organization by organization

DEPLOY - 4

4

Copyright John D. Musa 2000

Pilot the Idea

1. Apply to *real* project
   Example: Guide release of military command and control system
2. Use regular practitioners, not researchers
3. Experiment with details, evaluate successes and failures, modify idea
   Example: Failure of calendar time estimation led to execution time estimation
4. Provide support
5. Evaluate benefits and costs
6. Publicize success of idea

DEPLOY - 5

5

Copyright John D. Musa 2000

Test Sphere of Application

1. Recruit advocates to apply to variety of applications (at least 8 -10)
   Example: Inventory control, naval surveillance, operating system, word processor
2. Develop rudimentary training, support tools
3. Adapt idea to different environments

DEPLOY - 6

6

Copyright John D. Musa 2000

Test Sphere of Application

4. Continue experimenting with details, evaluating successes and failures, modifying idea
   Example: Developing concept and practice of operational profile
5. Reward feedback of problems
6. Publicize success of advocates *globally* to motivate them (also publicizing idea)

DEPLOY - 7

7

Copyright John D. Musa 2000

Raise Awareness and Interest

1. Broadcast the idea (publication and talks targeted to practitioners, especially outside)
2. Translate proven “benefits” to “problems solved”
3. Screen reactions to terminology; modify teminology as needed
   Example: software reliability measurement vs software reliability engineering

DEPLOY - 8

8

Copyright John D. Musa 2000

Raise Awareness and Interest

4. Differentiate approaches by audience:
   A. Practitioners: conflicting pressures
   B. Project managers: reducing risks
   C. Executives: financial success
5. Identify promising organizations - look for:
   A. Extremes
   B. Advocates

DEPLOY - 9

9

Copyright John D. Musa 2000

Implement Organization by Organization

1. Match idea to problems of organization
2. Qualify potential adopters
3 Plan pilot application
4. Transfer technology
5. Follow up

DEPLOY - 10

10

Copyright John D. Musa 2000

Match Idea to Problems of Organization

1. Convene focus group to identify organization key problems
   A. Include all stakeholders, even product users
   B. Refine and prioritize problems
2. Identify problems idea might help

DEPLOY - 11

11

Copyright John D. Musa 2000

Qualify Potential Adopters

1. Introduce idea to organization with highly interactive overview presentation
2. Show how idea can help organization key problems
3. Identify advocates
4. Have advocates prepare draft deployment plan

DEPLOY - 12

12

Copyright John D. Musa 2000

Qualify Potential Adopters

5. Advocates present draft plan to entire organization
   A. Circulate copies in advance
   B. Strongly encourage inputs in meeting
   C. Assign action items for issues that can't be resolved in meeting
   D. Identify candidates for pilot application
6. Identify and deal with resistance

DEPLOY - 13

13

Copyright John D. Musa 2000

Plan Pilot Applications

1. Select pilot applications
2. Plan transfer of technology
3. Monitor for problems; record and provide support as required
4. Plan recording of benefits

DEPLOY - 14

14

Copyright John D. Musa 2000

Transfer Technology

1. Approaches
   A. Consultant do all
   B. Self - teaching
   C. Course
   D. Course with workshops
2. Need to adapt

DEPLOY - 15

15

Copyright John D. Musa 2000

Follow Up Pilot Applications

1. Summarize benefits
2. Summarize problems and solutions
3. Have advocates publicize:
   A. In organization
   B. Outside

DEPLOY - 16

16

Copyright John D. Musa 2000

Problems and Solutions

Problem	Solution
Critique from researchers	Address proactively in working with practitioners
No time to implement	Staged introduction focused on quick initial benefit *Example: operational profile*
"We're already doing it"	Request details and compare with idea's details

DEPLOY - 17

17

Copyright John D. Musa 2000

Problems and Solutions

Problem	Solution
Reorganization	Restart, but leverage alumni
Loss of key advocate	Recruit multiple advocates, restart, leverage advocate
Cost reduction and downsizing	Emphasize cost benefit of idea

DEPLOY - 18

18

Copyright John D. Musa 2000

Conclusion

Deploying a new idea is primarily a marketing, sales, and organizational development problem, not a technical one.

DEPLOY - 19

19

Copyright John D. Musa 2000

References

1. Software Reliability Engineering website: overview, briefing for managers, bibliography of articles by software reliability engineering users, course information, useful references, Question of the Month:
   http://members.aol.com/JohnDMusa/
2. Musa, J. D., *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, ISBN 0-07-913271-5, McGraw-Hill, 1998.
3. Musa, J.D., "Developing More Reliable Software Faster and Cheaper," *Proc. 5th. IEEE International Conference on Engineering of Complex Computer Systems*, Las Vegas, NV, Oct. 18 - 21, 1999, pp. 162 - 176.

DEPLOY - 20

20

Copyright John D. Musa 2000

QW2000 Paper 8W1

Mr. Adrian Cowderoy
(MMHQ)

Technical Quality is Just the Start -- The Real Battle is Commercial Quality

BACK TO QW2000 PROGRAM

Presentation Abstract

In 1997 and 1998 a consortium of European companies and universities, supported by the European Commission, undertook a major exercise to explore the meaning of quality in the context of multimedia and website development, with attention to both the content and functionality. The author was the project manager and technical director of this research project, "MultiSpace".

Technical quality. Each of the different digital assets used in websites and multimedia has distinct characteristics that asset developers believe to represent "quality", such as listed in Table 3 below. While the importance of some technical quality features have loose relationship to system quality, others can be critical. For example, colour accuracy is safety-critical for some medical and defence systems.

The MultiSpace project reviewed system quality features, and especially those of the draft ISO/IEC 9126.1 and .2 ("Software product evaluation - Quality characteristics and guidelines for their use"). The project found that the ISO/IEC 9126 characteristics and sub-characteristics can be interpreted for multimedia and website content in a similar way as multimedia and website functionality. Indeed, the two are sometime inseparable (such as in usability).

However there were some significant technical problems. The first was that the usability sub-characteristic of "attractiveness" was inadequate at describing the richness of the experience of using multimedia and the Internet. The second was that the entire topic of usability concerns only reducing pain of using the human-system interface, and there is minimal attention to the joys. An effective description of website/multimedia quality would also be effective at describing the quality of the component media: books, films, training, games, advertising, etc.

Quality in use. Software systems are developed for a use (or set of uses) resulting from clearly defined needs. Although some multimedia and websites are utilitarian, many are also designed to please personal needs. Especially in the context of websites, these personal needs comes from a highly diverse audience and are difficult to model.

We have found it useful to make a distinction between the system constraints needed for achieving the primary purpose, and the system benefits that create the experience.

System benefits. System benefits involve giving people more than they expected, in a satisfying way.

Some vendors of commercial software packages achieve this by adding a wide range of new functions, some of which may be useful. Apart from resulting in bloatware and increased maintenance costs, this focuses only on utilitarian purpose. In contrast, traditional media combine quality with a heavy emphasis on rewards that create increase the quality of the experience. Specifically, they offer a variety of novelty features, supplementary learning, inter-personal participation and emotional satisfaction (or stimulation). Table 4 below lists sub-characteristics that describe these effects.

About the Author

Adrian Cowderoy is Managing Director of the Multimedia House of Quality Limited, a company which he established to promote quality-improvement methods for the production of websites and multimedia.

Mr Cowderoy was the General chair of ESCOM-SCOPE-99 and ESCOM-ENCRESS-98 conferences, and was Program chair for ESCOM 96 and 97 (The European Software Control and Metrics conference promotes leading-edge developments in industry and research, worldwide û see www.escom.co.uk). He is the METRICS-ESCOM Coordinator for IEEE METRICS 2001 and was on the Program committee of Metrics 98 and 99, European Quality Week 99 and COCOMO/SCM 96-99. In 1998 he was acting Conference Chair of the Electronics and Visual Arts conference in Gifu, Japan. He is a registered expert to the European Commission DGXIII.

He has provided consultancy and industrial training courses on quality management, risk management, and cost estimation to the aerospace and medical industries in the UK, Germany and Italy since 1995. He also lectures at Middlesex University (www.mdx.ac.uk) on e-commerce project management and managing Internet start-up's, and at City University, London (www.city.ac.uk), on project management for systems development.

Mr Cowderoy was project manager and technical director of MultiSpace, a 14-month million-dollar initiative sponsored by the European Commission in which 12 European organizations explored the potential to apply quality-improvement methods to multimedia and website development projects. (See www.mmhq.co.uk/multispace and www.cordis.lu/esprit.)

He was a Research fellow at City University from 1990-1998, and a Research Associate at Imperial College from 1986-1989. He was also a quality consultant and software developer at International Computers Limited, UK, from 1980-1985, where he worked on operating and networking systems for mainframes and distributed systems.

His academic qualifications include an MSc in Management Science from Imperial College, University of London in 1986, and is a member of the Association of MBA's. He received a BSc in Physics with Engineering from Queen Mary College, University of London, in 1979.

Mr. Cowderoy has published and presented extensively on multimedia quality and software cost estimation. He was joint editor of Project Control for 2000 and Beyond (Elsevier, 1998), Project Control for Software Quality (Elsevier, 1999), and Project Control: The Human Factor (Elsevier, 2000).

BACK TO QW2000 PROGRAM

www.nexusworld.net

Technical Quality is just the start – the real battle is *Commercial Quality*

Adrian Cowderoy

Copyright © NexusWorld.net Limited 2000

www.nexusworld.net

What is meant by Quality?

- The MultiSpace Project –
  - Major initiative supported by European Commission to explore the application of quality engineering techniques to multimedia and the web
  - 12 companies and universities
- Questions:
  - Is content different to functionality?
  - Do software quality characteristics describe books, teaching, music and entertainment?
  - How can quality be specified, tested and managed?

"The real battle is Commercial Quality"

QW2000, page 2

Copyright © NexusWorld.net Limited, 2000

The Contract View: System Quality

- Externally visible view of quality
  - allows effective management of contracts
  - is adaptable to "content"
    - Ch.3 of http://www.mmhq.co.uk/multispace/d2-2p.pdf
- Example, ISO/IEC 9126.2
  - Functionality (suitability, accuracy, security, etc)
  - Reliability ()
  - Usability (learnability, understandability, operability, attractiveness)
  - Maintainability ()
  - Efficiency ()

"The real battle is Commercial Quality"

QW2000, page 3

Copyright © NexusWorld.net Limited, 2000

Example measure of system quality

"Using text input"

X = A / B

A = number of places where text input can be used

B = total number of places where input can be given

Interpretation:

0 £ X £ 1

"A higher value means that more text input can be used."

Measures

"The real battle is Commercial Quality"

QW2000, page 4

Copyright © NexusWorld.net Limited, 2000

The Developer's View: Technical Quality

- Asset quality (visible to users)
  www.mmhq.co.uk/my-quality/measure-content-quality.shtml
  - Narrative
    - Spelling, grammar, linguistic simplicity, tone of voice, compactness, words per section, logical structure, text layout
  - Images 2D
    - Image quantity, image compression, contrast control, gradation, edge sharpness, use of color, color mode, color fidelity, printing accuracy, color vibrancy, harmony with scheme, composition.
  - Navigational aids, Movies, 3V & VR, etc
- Use for specifying testing critical features
  - e.g. medical and cultural arts need quality

"The real battle is Commercial Quality"

QW2000, page 5

Copyright © NexusWorld.net Limited, 2000

Other types of Technical Quality

- Asset complexity (internal characteristics)
  http://www.mmhq.co.uk/my-complexity/
  - object complexity
    - e.g. image layers, size, colors, etc
  - structural complexity between objects

  → indicates possible problems:
    - identify high-risk components
    - charge-rate for changes, subject to constraints
    - indicators of poor operability
- Process efficiency
  - asset management system
  - documentation

"The real battle is Commercial Quality"

QW2000, page 6

Copyright © NexusWorld.net Limited, 2000

The Commercial View: System Rewards

- Usability: attractiveness
  - usability is the efficiency of human-machine-interface
  - only "sufficient" usability is needed
- The web involves multiple media
  - what makes a good book, film or music track?
  - what makes effective training?
  - what increases sales?

→To compete with the top web-developers
ALL quality perspectives must be

"The real battle is Commercial Quality"

QW2000, page 7

Copyright © NexusWorld.net Limited, 2000

The Commercial View: System Rewards

- System constraints
  - need <u>just enough</u> for users, immediate business purpose, and long-term maintainability
- Technical quality of assets
  - need <u>just enough</u> for critical assets
- System rewards
  - effects that enhance business purpose
  - need <u>as much as possible</u> within project constraints

"The real battle is Commercial Quality"

QW2000, page 8

Copyright © NexusWorld.net Limited, 2000

System Rewards

- 5 characteristics (with 29 sub-characteristics):
  - Participation (..., ...)
  - Discovery (..., ...)
  - Novelty (experience, paradox, dramatic, technical)
  - Involvement (..., ...)
  - Comfort (..., ...)
- Balance
  - different people like different features
  - some have negative effects
  - some features work best together
  - some features are contradictory

"The real battle is Commercial Quality"

QW2000, page 9

Copyright © NexusWorld.net Limited, 2000

Sub-characteristics of Participation

- Bulletin boards
- Email circulars
- Real-time interaction
- Physical meeting support
- Pushed content
- Personalization of interaction

"The real battle is Commercial Quality"

QW2000, page 10

Copyright © NexusWorld.net Limited, 2000

Sub-characteristics of Discovery

Learning about ...

...people

...events

...organizations

...tools and processes

...science and technology

...the world

"The real battle is Commercial Quality"

QW2000, page 11

Copyright © NexusWorld.net Limited, 2000

Sub-characteristics of Novelty

- Personal experience
- Paradox/riddle
- Dramatic experience
- Technical gimmicks

BEWARE:
Balance novelty with Usability and Comfort

"The real battle is Commercial Quality"

QW2000, page 12

Copyright © NexusWorld.net Limited, 2000

Sub-characteristics of Involvement

Encourage selected emotions
and discourage others

- News media often rely on anger, fear, pity
- Advertising relies on confidence, greed/avarice
- Entertainment exploits opposites (e.g. love and hate) and excites physiological effects (such as lust)

"The real battle is Commercial Quality"

QW2000, page 13

Copyright © NexusWorld.net Limited, 2000

Sub-characteristics of Comfort

- Visual beauty
- Poetic act
- Engineering elegance
- Expectation
- Conformance to social norms

N.B. Comfort is a reward,
Usability is a requirement

"The real battle is Commercial Quality"

QW2000, page 14

Copyright © NexusWorld.net Limited, 2000

Using benefits: Resolve conflicts

- Resolve conflicts
  - business and marketing people need cost control
  - users want as many benefits as possible,
    and sufficient usability, etc.
  - technical staff have technical constraints from the tools and legacy content
  - domain specialists message constraints
    (for training, advertising, etc)
- Proposed mechanism
  - rank each (relevant) feature
    on scale 1 (irrelevant) to 3 (critical)

"The real battle is Commercial Quality"

QW2000, page 15

Copyright © NexusWorld.net Limited, 2000

Using benefits: User diversity

- Linguistic ability
- Familiarity with the domain
- Computer literacy
- Web-behavior
  - seeker, resident, tutor, etc ...
- Cultural
  - different learning methods
  - different problem-solving methods
  - different interpersonal behavior
  - different value systems
  - different history (and symbols)

"The real battle is Commercial Quality"

QW2000, page 16

Copyright © NexusWorld.net Limited, 2000

Using benefits: Plan Quality

1) Is it engaging?
   – it takes one click to leave a web-site !
2) Is it inspiring?
   – why return to the website ?
3) Is it cool? (Is the style right?)
   – being cool is hard work.
4) Is it effective?
   – are the promises fulfilled?
5) Is it maintainable?
   – adaptations and extension often cost more than the original website creation.

"The real battle is Commercial Quality"
QW2000, page 17

Copyright © NexusWorld.net Limited, 2000

Is it cool?

- Cool is being laid back
  – cool websites do not require massive content or functionality (although they may have them)
- Cool is being self-assured
  – your website mustn't pretend to be something else
- Cool is being consistent
  … until you change your mind
- Cool is about style, not functional or utility
  – some cool sites are difficult to use

"The real battle is Commercial Quality"
QW2000, page 18

Copyright © NexusWorld.net Limited, 2000

Examples of building coolness

- Novelty
  - personal experience - create experiences for visitors or describe cool experiences, but stay laid-back
  - paradox/riddle can be used to create coolness
    - take care to maintain consistency
  - dramatic experience requires a great skill to do in a cool way - don't photocopying other people
  - technical gimmicks
    - avoid gimmicks that are merely functional or are widely used
- Likewise for discovery, participation, involvement, comfort

"The real battle is Commercial Quality"

QW2000, page 19

Copyright © NexusWorld.net Limited, 2000

Using benefits: Control Asset wastage

- Content from libraries and subcontractors can dominate project costs
- Cost of errors
  - unsuitable content has to be changed or replaced
  - indirect costs of replacing material in the field
- Impact of changed priorities
  - cost of new content, and waste when some old content is abandoned

"The real battle is Commercial Quality"

QW2000, page 20

Copyright © NexusWorld.net Limited, 2000

Process improvement - how?

- Manufacturing processes: reduction of waste
  - equates waste with inefficiency
  - benefits from statistical process control
  - needs process standardization
  - relies on steady changes in technology and market
- Alternative improvement mechanisms
  - risk reduction
  - quality checking at milestones
  - performance review based on 2-4

"The real battle is Commercial Quality"

QW2000, page 21

Copyright © NexusWorld.net Limited, 2000

Web sites

http://www.mmhq.co.uk/

/my-quality/measure-it.shtml ... quality measures

/my-quality/is-it-ok.shtml ... applying measures

/multispace/framework.shtml ... quality processes from the EU project

/my-complexity/ ... complexity measures

/quality-planning/ ... mini-tutorial

"The real battle is Commercial Quality"

QW2000, page 22

Copyright © NexusWorld.net Limited, 2000

Contact details

Adrian Cowderoy

email adrian@nexusworld.net

phone +44(UK) 118 9 427 970

snailmail NexusWorld.net Limited
46 Western Alms Avenue
Reading RG30 2AN
United Kingdom

URL www.nexusworld.net

"The real battle is Commercial Quality"

QW2000, page 23

Copyright © NexusWorld.net Limited, 2000

Technical Quality is just the start.
The real battle is *Commercial Quality*.

Adrian Cowderoy

NexusWorld.net and The Multimedia House of Quality

In 1997 the European Commission formed a project to explore the *meaning of quality* in the context of multimedia and Internet system development, and to establish whether engineering approaches to quality-improvement could be applied to this emerging domain.

The starting point were the various software frameworks used for quality specification, including the perspectives of system quality, quality in use and Technical Quality ("internal product characteristics"). We anticipated that the perspective of Technical Quality would need massive extension to support individual digital assets (pictures, text, movies, etc). What we did not anticipate, was that that there is a fourth perspective on quality that has been almost entirely ignored in the software industry.

This paper is about the fourth perspective, and how it was found.

1) Common perspectives on quality

The MultiSpace project combined experts from twelve organizations in the software quality industry with website developers and experts. (The term "website" is used in this paper, but most of the material also applies to multimedia, Internet, intranet and extranet systems.) The first meeting was traumatic – the software people had tightly refined ideas about quality which the web-development people saw as seemingly irrelevant to most of the real issues they regularly faced in making effective systems.

The first exercise was to examine the existing materials and listen to the needs of a wide variety of practitioners. With this the team developed its many initial views into a single synthesis that considered what quality meant, how it could be managed and how it fitted within the website development cycle. (There would be little point in defining quality, if it could not be managed.) It also considered how to profile the very diverse range of users. Collectively, this represents the MultiSpace Framework [Cowderoy & Daily, 1997].

Amongst the observations in the MultiSpace Framework are that quality is important for managing contracts, for planning the quality-improvement and testing, and for evaluating the different responses that users have to a single system. For this, descriptions are needed of system quality, Technical Quality, and the users. This section presents the first two of these – the third is presented in section 2.5.

Adrian Cowderoy
Copyright © NexusWorld.net Limited, 2000

1.1) System Constraints

Effective management of contracts is essential to client-contractor relationships when the financial amount is to great for relationships of trust, or when the technical complexity is too great for an individual architect, or when a market-driven approach is used for production of individual components. (Market-driven production involves outsourcing each step of the production chain to whichever contractor is cheapest at the required quality and delivery time [Williamson 1983].)

Efficient contracts for website development need to specify what needs to be delivered in terms of content, functionality and quality. The quality specification needs to address all quality characteristics, not just those that are easy to define.

In the software industry, various schemes have been produced for measuring the quality of delivered functionality. Most of these focus on measuring a specific characteristic, such as reliability or usability, and ignore other issues. Unfortunately schemes often contradict each other. For some years, the ISO/IEC 9126 committee has been working on a quality characterization scheme that describes all quality characteristics and sub-characteristics of software. This also has contradictions with other schemes, and weaknesses (such as with safety issues), but it forms a useful starting point for describing quality. (The 10 Design Quality features proposed by Fellenstein and Wood of IBM [2000], also correspond to a subset of 9126 quality sub-characteristics.)

Websites include deliver both content and functionality. The functionality is described by the existing 9126 definitions. In the MultiSpace project, we experimented with adapting those definitions to describe content. This included text, 2D images, 3D images, video, audio and navigation aids. The subcharacteristics descriptions can be found in chapter 3 of the MultiSpace Priorities [van Veenendaal]. These are supplemented by over 100 measures, with support for all the subcharacteristics [Boonstra, 1998]. When MultiSpace started, the accusation had been labeled at the project that quality was far too illusive to be measured. The work of the Philips and KEMA teams in MultiSpace proved that content quality *can* be measured for websites, in the context of fitness-for-purpose.

Table 1. Examples of Operability metrics for Internet and multimedia systems [Boonstra 1998].

External metrics	Measurement	Interpretation	Scale type	Measure type
Using text input	X = (A / B) A = number of places where text input can be given B = total number of places where input can be given	0<=X<=1 A higher value means that more text input can be used.	Ratio	X = Ratio A = Count B = Count

Using voice input	X = (A / B) A = number of places where voice input can be given B = total number of places where input can be given	0<=X<=1 A higher value means that more voice input can be used.	Ratio	X = Ratio A = Count B = Count
Using displacement input	X = (A / B) A = number of places where displacement input can be given B = total number of places where input can be given	0<=X<=1 A higher value means that more displacement input can be used.	Ratio	X = Ratio A = Count B = Count

Copyright © The MultiSpace partners, as project EP23066 under the ESPRIT Programme.

1.2) Technical Quality

Each of the different digital assets used in websites has distinct characteristics that "quality", such as listed in Table 2 below. The extent of the contribution to quality varies between applications, from extremes of zero relevance to extreme high. For example, color accuracy is safety-critical for some medical and defense systems, textual accuracy is system critical for web-retail sites and teaching materials.

Table 2. Summary of characteristics for different types of digital asset.

Asset type	Characteristics
Narrative	Spelling, grammar, linguistic simplicity, tone of voice, compactness, quantity per section, logical structure, text layout.
Navigational aids	Dynamic effects, navigation harmony with main scheme, hyperlink quantity, URL addresses shown, exclusion of advertising banners.
Images 2D	Image quantity, image compression, contrast control, smooth tonal transition ("gradation"), edge sharpness, use of color, color mode, color fidelity, printing accuracy, color vibrancy, image harmony with main scheme, image composition.
Movies	Frame size, movie compression, frame rate, directing, movie composition, editing, after effects.

For detailed descriptions, see http://www.mmhq.co.uk/my-quality/

There are also suggestions that the adaptability and extendibility of a website may be linked to complexity and size of individual assets, and the compound structures that combine assets [Cowderoy 2000 and http://www.mmhq.co.uk/my-complexity/benefits.shtml].

Copyright © NexusWorld.net Limited, 2000

2) System Rewards

2.1) The argument for System Rewards

For centuries, engineers have worked to satisfy the needs of their clients and the public. In past centuries, this was sometimes supplemented by flights of creativity that added features that were aesthetically pleasing to some people, but served no clear functional purpose.

During the twentieth century, the influence of factory processes was extended to other industries. This created a massive improvement in the standard of living, especially as a result of the widespread use of statistical product and quality techniques that support ever-closer satisfaction of functional purposes. With such techniques, we could now construct the architecture of St Peter's dome, the works of Shakespeare and the music of Beethoven with much greater efficiency – surplus ornamentation, words and notes would be removed, achieving improved value for money and higher system efficiency.

The importance of beauty was recognized by the ISO/IEC 9126.2 committee when they defined the usability sub-characteristic of Attractiveness. However in its current form it is inadequate at describing the richness of the *experience* of using websites.

The ISO/IECE 9126.2 use of Attractiveness is also in contradiction to the other usability sub-characteristics, which describe how the *pain* of using the human-system interface can be reduced.

An effective description of website quality would also be effective at describing the quality of the component media: books, films, training, games, advertising, etc. if Michelangelo, Shakespeare and Beethoven were correct in their works, then something is missing from the engineer's definition of quality. More specifically :-

> *The commercial failure of some websites can be linked to bad system quality, but once adequate system quality has been achieved, there is no structure for defining what makes some sites a commercial success, and others a disaster.*

Following some earlier work in MultiSpace, we proposed at the IEEE Symposium on Software Metrics that, with some exceptions, websites have to provide rewards to their users as well as satisfying needs [Cowderoy *et al*, 1998]. In effect, quality consists of two sets of features:

- *System Constraints* involving the satisfaction of needs to perform the functional purpose of the website. A well-designed system provides "just enough" quality for the purpose – anything more would reduce productivity and delivery time.
- *System Rewards*, involving the provision of features in addition to the primary purpose. Such features are individually dependent – each person likes some features, and ignores (or dislikes) others. The objective of the developers is to provide as much reward as possible within the budget, and timescale. Rewards increase attention, prolong visits to websites, and encourage return visits. The

 Copyright © NexusWorld.net Limited, 2000

financial benefits of these can be so significant that it can justify increased budget and delayed delivery.

The system benefits are listed in detail in section 3 and section 4.

2.2) Asset wastage

Ideally, System Rewards should be defined at the start of the project, and either included in the budget or prioritized so that the most important ones are included. However typically System Rewards can only be outlined at the start of the project in a very general fashion. The specific ideas only come part way through, and may come at the first customer demonstration. These late arrivals may require a change of priorities partway through the project. An effect of changing priorities is a level of waste – i.e. assets that are partly developed, then discarded.

There is a misleading similarity between the wasted resources from discarded assets in content development, and the wasted effort resulting from late detection of faults in software development. The main differences are :-

- Wasted assets can occur from *changed priorities*, resulting in increased financial advantage.
- Wasted assets can also occur because the design includes some design features that are certainties, and others that are *opportunities* – i.e. there is a possibility that their impact will be different to original expectations (greater or less), and they may create secondary risks (such as reduced usability and reliability and new opportunities.)

2.3) Quality improvement strategies

In software engineering, it is often said to be "good practice" to use processes that reduce the possibility of faults. However in content development, the major challenge is to manage the risks and opportunities efficiently. The most efficient way of handling this is with risk management techniques, but to list both good and bad events, and control actions that create leverage to improve each opportunity or risk. Often these control actions correspond to conventional quality assurance and quality control actions, however compared to standardized software engineering practices, these extra quality methods are used only when the benefits clearly outweigh the costs for the current project.

Another promising methods for web development is concurrent engineering [Carter & Baker]. The concurrency helps with Rapid Application Development, the massive broadband communications helps improve creativity and reduces many of the quality risks within the project. It also supports risk management, quality profiling and people management,

 Copyright © NexusWorld.net Limited, 2000

2.4) Introducing quality-improvement techniques

We have encountered some hostility to the introduction of engineering methods into environments that depend on creativity and craft skills. This has led to various conclusions:

- Process definition, standardization and inspection techniques currently only apply to certain activities.
- Quality engineer's belief in the importance of statistical methods is seen as a religion (sometimes fanatic) and not a proposition that is feasible in an industry that can see major changes within only 3 months.
- Quality-training needs to focus on a few key issues to which web development staff can directly relate.

Table 3. Example of a teaching aid in this area
from http://www.mmhq.co.uk/cool-way/

To teach newcomers to project management, the MMHQ website encourages people to focus on four areas: the project participants, the product quality, the project risks, and control mechanisms. (In this context, control is used to improve communications, quality and risks. The strong emphasis on project participants is important for a craft-based industry.) The emphasis is on increasing awareness of how each of the four areas effects the others, and can be used to improve the others. Within each of the four areas, people are encouraged to appreciate the breadth of topics that should be considered. To highlight interest, the exercise is presented as a set of card games ("the Cool Way"). It encourages use of (conventional) methods such as risk management and quality profiling, and it provides a framework on which more detailed and advanced methods can be built.

2.5) User perspectives

Content-rich web-sites may receive a much more diverse range of users than conventional software systems.

- For software, the designers and usability evaluators have to address different levels of computer literacy, physical/visual dexterity and (perhaps) familiarity with the application area.
- For websites, the "familiarity with the application domain" can become much more complex, especially for the System Benefits. There are also important differences from web-behavior (seeker, resident, tutor, etc), linguistic ability, national variations in group behavior and learning strategies, and value sets (such as from ethnic traditions). Failure to adequately address these differences can result in customers abandoning the web-site, and can even result in hostile behavior.

To perform statistical evaluation of the system rewards websites would require a large population, sampled to include all these different categories. This would be expensive, time-consuming and it can be difficult to maintain confidentiality an alternative is to

Copyright © NexusWorld.net Limited, 2000

use a small sample, that are carefully chosen so that each category is represented by one (or more) people in the sample. The people in the sample tend to fit into three groups :-

- *Novices* are new to evaluation, and give unbiased opinions, which need to be solicited using a combination of questionnaire, interview and observation – i.e. the evaluation cost is relatively high.
- *Actors* are experienced evaluators who play a specified role – they may give very detailed responses, responding using questionnaires and freestyle, but there is risk that their responses are stereotyped.
- *Focus-group members* interact with each other (as well as the moderator) may tend to double-guess business strategies and other information that has been withheld from them.

3) Planning the System Rewards

There are five general categories of reward. Existing websites often totally ignore some of these categories, but in commercially-oriented websites, most or all of these need to be considered.

- *Participation* is the core of the one-to-one marketing and community-based e-business which is one of the characteristics of the New Economy [Siegel, 1999].
- *Discovery* increases retention and, if there is sufficient new material, it encourages return visits.
- *Novelty* attracts attention, which increases retention at the site and may improve concentration.
- *Involvement* increases concentration and retention. When there is a strong emotional element it can encourage return visits.
- *Comfort* is essential for people to cope with the stress from using the system and its rewards.

The 5 reward categories subdivide into 29 specific features. Effective quality planning needs all 29 specific features. In assigning importance, beware of the special cases:

- Some features may have negative effects for a particular business or audience. (For example, most sites would be negatively influenced by the inclusion of any feature than incited lust.)
- Some features are complimentary to other features within the same category. These work better when used together. (For example, “poetic act” benefits from co-existence with “visual beauty”.)
- Features in one category may compliment or clash with features in another category, or with System Constraints. (For example, novelty requires the presence of comfort and the system constraint of understandability.)

As well as considering special cases, the importance of each feature needs to resolve conflicting needs from the four types of player in the project:

- *Business and marketing people,* who require the website to justify its costs. To this purpose, they need the inclusion of some features but see no direct financial

Copyright © NexusWorld.net Limited, 2000

benefit from many others, unless these other features change the behavior of users.

- *Users,* who may include different and conflicting audiences and may include people who are hostile to the website owners. As explained in section 2.5, there is usually a diverse range of users to interview.
- *Domain specialists,* who protect the site from misrepresentation of messages. (For example, marketing people protect brand images, and teachers demand strict use of terminology.)
- *Technical staff,* who carry the legacy of the existing website, the constraints from tools, and the constraints of existing software solutions to which the web-site provides a front-end (which is common for advanced e-commerce systems).

It takes a modest effort to identify the priorities for so many players, and then resolve them, (A set of linked spreadsheets helps.) However without that effort there is a risk that key features have been forgotten, or under-emphasized, or exaggerated. The cost of such mistakes can be much greater than the effort of performing quality profiling.

4) Characterization of System Rewards

This section contains lists of the various characteristics and sub-characteristics which collectively describe the System Rewards. Examples can be found at http://www.mmhq.co.uk/my-quality/examples.shtml

4.1) Participation

Participation includes involvement with other people in both real and virtual contexts. On the web, participation tends to be aimed at very specific user communities (i.e. people with some similar objectives). To make participation effective other reward features may be need to emphasized to suit the local needs of the target users.

N.B. Participation often requires major technical challenges, which introduce other problems (such as with reliability, usability and performance).

Feature	Summary	Simple scale
Bulletin boards	Electronic notice-boards to which user's can append their own messages (usually without censorship).	Irrelevant, Minimal, Good practice, or State-of-art.
Email circulars	Letters sent electronically to an distribution list.	Irrelevant, Minimal, Good practice, or State-of-art.
Real-time interaction	In which user's engage in real-time dialogue in an on-line environment. (Includes use of avatars and video conferencing.)	Irrelevant, Minimal, Good practice, or State-of-art.

Copyright © NexusWorld.net Limited, 2000

Physical meeting support	Support of meetings with preparation, broadcast, and dissemination.	Irrelevant, Minimal, Good practice, or State-of-art.
Pushed content	Information pushed onto a recipients computer without the user directly requesting it.	Irrelevant, Minimal, Good practice, or State-of-art.
Personalization of interaction	Automatic selection of content or functionality based on a user's actual or perceived needs. (Used as a marketing tool as well as for handling information overload.)	Irrelevant, Minimal, Good practice, or State-of-art.

4.2) Discovery

This includes the discovery of useful or interesting information not related to the primary purpose of the system. Covers other indirectly related topics. (Currently the most common intended uses of supplementary learning is in website design, where related indirect topics are intentionally included in site content in order to please some of the visitors.)

Feature	Summary	Simple scale
Learning about people	Provision of information about people related to current context.	Irrelevant, Occasional, Explicit, or Dominating.
Learning about events	Reports of upcoming and completed events loosely related to the current context.	Irrelevant, Occasional, Explicit, or Dominating.
Learning about organizations	Provision of information about organizations and social groups indirectly related to current context.	Irrelevant, Occasional, Explicit, or Dominating.
Learning about tools and processes	Provision of information about the tools and processes used to improve their work efficiency and the quality of their results.	Irrelevant, Occasional, Explicit, or Dominating.
Learning about science and technology	Provision of information about how and why things work.	Irrelevant, Occasional, Explicit, or Dominating.
Learning about the world	Provision of information about mankind, nature, science, our planet and outer space.	Irrelevant, Occasional, Explicit, or Dominating.

Copyright © NexusWorld.net Limited, 2000

4.3) Novelty

Novelty involves the provision of something unusual or new. The extent of novelty can be influenced by the user's familiarity with the domain. (N.B. High novelty frequently results in significantly reduced usability - see Usability measures and http://www.emmus.org/)

Feature	Summary	Simple scale
Personal experience	A system satisfying a user's previously unrecognized need or stimulus.	Irrelevant, Average, Unusual, or Exceptional.
Paradox/riddle	Provision of contradictory features or phrases to which a solution is promised.	Irrelevant, Average, Unusual, or Exceptional.
Dramatic experience	Provision of a novel state or situation in the traditions of story-telling, art or music making.	Irrelevant, Average, Unusual, or Exceptional.
Technical gimmicks	Provision of attention-getting device.	Irrelevant, Average, Unusual, or Exceptional.

4.4) Involvement

Emotional involvement includes the excitement or strong feeling, encouraging psychological (and physiological) effect. This can be addictive, encouraging return visits. Emotional involvement is used extensively in advertising, entertainment and news services.

In the context of a website, only some emotional features are desirable, while some others may be highly undesirable. In highly utilitarian websites, it is normal (and easy) to avoid all emotional involvement.

Feature	Summary	Simple scale
Confidence	Faith and trust in the future. Courage to handle events. (The inverse is the condition of "learned helplessness", in which all actions are believed to result in failure, despite new evidence to the contrary.)	Irrelevant, Occasional, Explicit, or Dominating.
Love	Strong affection for a person or group of people, based on admiration, benevolence, common interests or personal ties.	Irrelevant, Occasional, Explicit, or Dominating.
Pity	Empathy with one or more people in a significantly worse situation than the user.	Irrelevant, Occasional, Explicit, or Dominating.

Copyright © NexusWorld.net Limited, 2000

Greed	Desire for acquisition, consumption or power.	Irrelevant, Occasional, Explicit, or Dominating.
Anger	Strong displeasure resulting from actual or perceived injury. Anger is often accompanied by a desire for action. The quality feature needs to be carefully combined with appropriate other features.	Irrelevant, Occasional, Explicit, or Dominating.
Fear	Anticipation or awareness of danger to self, or people with one has an affinity.	Irrelevant, Occasional, Explicit, or Dominating.
Lust	Strong carnal desire, devoid from love. This may have an immediate effect (positive or negative) that can greatly diminish the effect of most other quality features.	Irrelevant, Occasional, Explicit, or Dominating.
Selfishness	Disregard for others. Encouraging selfishness, and can aggravate racism and bigotry.	Irrelevant, Occasional, Explicit, or Dominating.

4.5) Comfort

Comfort includes the provision of reassurance and pleasure. Comfort is used to balance the difficulties typically associated with other rewards (novelty, supplementary learning, participation and emotional involvement). Some of the comfort features listed below are important only in some for certain types of product.

There is a superficial similarity between the System Reward of “comfort” and the system constraint of “usability”. However comfort is a positive effect, and some people require as much comfort as possible. In contrast, usability involves only the reduction of pain in using the system, and it is only necessary to have “just enough” usability within the system.

Feature	Summary	Simple scale
Visual beauty	Visual impression designed to create positive emotion.	Irrelevant, Average, High, or Extreme.
Poetic act	Elegance of act or thought, with altruistic motive. Written poetry is one of many forms of poetic action.	Irrelevant, Average, High, or Extreme.
Engineering elegance	Provision of efficient structure with functional simplicity.	Irrelevant, Average, High, or Extreme.
Expectation	Use of themes that promise a later reward.	Irrelevant, Average, High, or Extreme.
Conformance to social norms	Adoption of political correctness and good morals.	Irrelevant, Average, High, or Extreme.

Copyright © NexusWorld.net Limited, 2000

5) Conclusion

Web-site owners and developers dream of creating killer websites. The probability of achieving this can be greatly increased by identifying the priorities for System Rewards, then methodically improving the most important ones.

This paper is a start. There is still much research that is needed to improve the dichotomy of System Rewards (as presented in section 4), to provide suitable measures and to provide appropriate quality-improvement schemes. However the key lesson is that engineering methods can be applied to support creative and commercial activities.

References

Boonstra, F., "Definition of external metrics", in *MultiSpace Quality Pack 2: Quality specification: questionnaires and metrics*, EP23066-D3.2C (Feb 3, 1998).

Carter, D. E. and Baker, B. S. *Concurrent Engineering: The Product Development Environment for the 1990's*. Reading, MA: Addison-Wesley, 1991, ASIN 0201563495.

Cowderoy, A.J.C., Daily, K., *The MultiSpace Framework*, ESPRIT EP23066, 1997. Available at http://www.mmhq.co.uk/multispace/d2-1p.pdf

Cowderoy, A.J.C., Donaldson, A.J.M., Jenkins. J.O., "A metrics framework for multimedia creation", *Proc. of the 5th International Software Metrics Symposium*, Nov 20-21, 1998, Maryland, USA, IEEE Computer Society, ISBN 0-8186-9201-4.

Cowderoy, A.J.C., "Measures of size and complexity for web-site content", ESCOM SCOPE 2000, April 18-20, 2000. Published in *Project Control, the Human Factor*, Shaker. Available at http://www.mmhq.co.uk/papers/escomscope2000.pdf

Fellenstein, C., and Wood, R., *Exploring E-Commerce, Global E-Business and E-Society*, Prentice Hall, 2000, ISBN: 0130848468.

Siegel, D., Futurize Your Enterprise: Business Strategy in the Age of the E-customer, John Wiley & Sons, 1999, ISBN: 0471357634. Website: http://www.futurizenow.com/

Williamson, O.E., *Markets and Hierarchies: Analysis and Antitrust Implications,* Free Press, 1983, ISBN 0029347807.

van Veenendaal, E.P.W.M., "Quality characteristics for multimedia systems", Ch.3 of *The MultiSpace Priorities,* EP23066-D2.2P, 1998. Available at http://www.mmhq.co.uk/multispace/d2-2p.pdf

QW2000 Paper 8M1

Mr. D.J. Law
(QWest Communications)

Certification Programs for Software Quality and Test Professionals

BACK TO QW2000 PROGRAM

Presentation Abstract

For the serious software quality and test practitioner, a quality or test professional certification enhances one's career, realizes professional development goals, gains recognition for performance excellence, and distinguishes yourself from your peers in today's competitive marketplace. In this presentation, you will learn why it is important to become certified, the benefits of certification, what you will need to qualify, how the certification process works, how to pass the examinations, and how to keep your certification.

There will be a detail analysis of six certification programs for the software quality and test professional, from two different quality certifying bodies and one test product vendor, and the prescribed common body of knowledge for all six certification programs will be examined.

In addition, there will be a discussion of exam taking techniques, sample test questions, resources for further study, and how to market your certification.

About the Author

Mr. Law is an independent consultant at Qwest Communications where he specializes as a web-based test engineer. He has 16 years of software testing experience in managing test teams, designing test cases, developing test scripts, and establishing automated test infrastructures for test organizations.

His prior clients have included IBM, Lockheed Martin, Fannie Mae, Oracle, EDS, Boeing, Mobil, and The Executive Office of the President. Mr. Law is a member of the American Society for Quality and is an ASQ Certified Software Quality Engineer.

BACK TO QW2000 PROGRAM

Professional Certification Programs for the Software Quality and Test Practitioner

Presented by D. J. Law, CSQE, CSTE

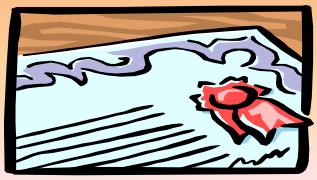
Presentation Outline

- What is certification?
- Why become certified?
- How to get certified?
- Which certification?
- How to study?
- How to pass the exam?
- How to keep your certification?
- Where to go for resources?
- How to get started?

Copyright © 2000 D. J. Law

What is Certification?

- Certify (verb): 1. To confirm formally as true, accurate, or genuine. 2. To guarantee as meeting a standard. (Dictionary)
- Certification is the formal recognition that an individual has demonstrated a proficiency within and comprehension of a specified body of knowledge at a point in time. (ASQ)
- Not a license or registration



Copyright © 2000 D. J. Law

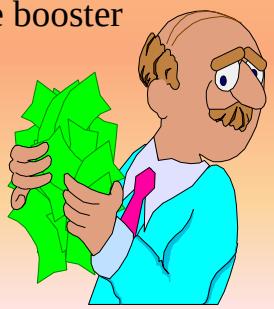
Why Become Certified?

- Formal proof of credibility in a profession
- Demonstration of competency in a specified subject matter
- Official recognition of proficiency within a prescribed body of knowledge
- A common standard to evaluate knowledge and skill
- A credential to measure an individual's job-related skill set
- An up-to-date performance-based assessment of a person's abilities and mastery in a particular area of expertise

Copyright © 2000 D. J. Law

Benefits of Certification

- Career advancement
- Salary goal fulfillment
- Faster promotion
- Hiring advantage
- Better job opportunities
- Professional development
- Greater marketability
- Professional credibility
- Affirmation of your commitment to quality
- Enhanced confidence and self-esteem
- Personal satisfaction
- Financial rewards
- Peer recognition
- Sense of professional accomplishment
- Resume booster



Copyright © 2000 D. J. Law

Benefits for Employer and Client

- Tangible measure of employee's capabilities
- Ensures clients that employee has proven experience
- A reliable benchmark for hiring and promoting
- Provide higher levels of service and customer satisfaction
- Assurance of quality when outsourcing services
- Win-win situation for employee, employer, and client
- Recognition and reward for productive employees
- Objective evidence of employee's abilities
- Increased employee productivity
- Adds value to key job functions
- Greater job commitment
- Validates training investment
- Competitive market advantage

Copyright © 2000 D. J. Law

Disadvantages and Criticisms

- Limited useful lifespan
- Must maintain certification
- Recertification due to new release or version
- Employee may leave
- Must have time to study
- Expensive application fee

- Book knowledge
- No real-world skills required

Benefits Outweigh Drawbacks

Copyright © 2000 D. J. Law

Certification Program Types

- Organization-based (vendor-independent, CBOK)
  - ASQ (**CSQE**, CQMgr**)**
  - QAI (**CSTE**, **CQA**)
  - BCS (CST)

- Institution-based (vendor-neutral, no renewal)
  - IIST (**CSTP**)
- Vendor-based (specific product and version, no CBOK)
  - Mercury Interactive (**CPS**, **CPI**)
  - Segue (CSSC)

Copyright © 2000 D. J. Law

Some Certification Facts

- History
  - ASQ: 1968, 81,000+
  - ICCP: 1973, 50,000+
  - Novell: 1989, 400,000+
  - Microsoft: 1992, 400,000+
- Over 320 certifications from over 60 certifying entities
- Mostly ASQ and QAI for the quality and test profession
- Future possibilities
  - ASQ: Certified Software Test Engineer?
  - Segue: Certified Segue Functional Consultant?
  - BCS: Practitioner and Diploma levels of CST?

Copyright © 2000 D. J. Law

How to Become Certified?

- Certification lifecycle:
  - Eligibility determination
  - Application submittal
  - Eligibility notification
  - Examination preparation
  - Examination
  - Results notification
  - Certification maintenance

Copyright © 2000 D. J. Law

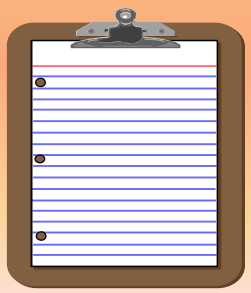
Eligibility Criteria

- Must satisfy qualification requirements
  - Baccalaureate degree (diploma, transcript)
  - Proof of professionalism (letters, references, societies)
  - Subscribe to code of ethics (signature)
  - X years of professional paid experience (resume)
- Waivers
  - Substitute advanced degrees for x years of experience

Copyright © 2000 D. J. Law

Application Process

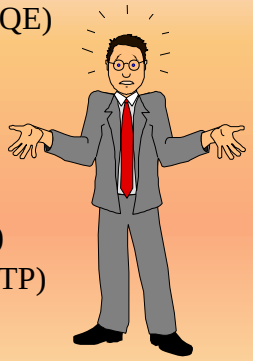
- Complete application
- Submit documentary materials
- Enclose application fee
- Letter of notification



Copyright © 2000 D. J. Law

Which Certification?

- For the Software Quality Professional:
  - Certified Software Quality Engineer (CSQE)
  - Certified Quality Analyst (CQA)
  - Certified Quality Manager (CQMgr)
- For the Software Test Professional:
  - Certified Software Test Engineer (CSTE)
  - Certified Software Test Professional (CSTP)
  - Certified Product Specialist (CPS)
  - Certified Product Instructor (CPI)
  - Certified Segue Scalability Consultant (CSSC)
  - Certificate in Software Testing (CST)



Copyright © 2000 D. J. Law

Certified Software Quality Engineer

- Sponsor --- American Society for Quality
- Eligibility --- 8 years work experience, 1 to 5 years waiver permissible
- Application due --- 2 months before exam date
- Fee --- $90 member, $195 non-member
- Format --- 160 multiple choice questions, not categorized, random order
- Answer sheet --- optical scanned (mark with No. 2 pencil)
- Type --- open book, but no sample questions/answers in materials
- Permissible --- basic calculator, reference books (proctor must review)
- Length --- 4 hours
- Breaks --- permitted as necessary at your convenience
- Dates --- twice a year (1st Sat. in June and December)
- Times --- usually 8:00 AM to 12:00 PM noon
- Location --- hosted by local ASQ chapters (offices/schools)

Copyright © 2000 D. J. Law

CSQE

- Passing score --- scale score of 550 (range 200 to 750)
- Expiration --- 3 years from exam date
- Renewal --- every 3 years, $30 members, $50 non-members
- Refunds --- yes, minus $20 processing charge
- Retake --- no limit on number of times but charges a resit fee
- Common body of knowledge (8 areas)
  - General knowledge, conduct, and ethics (24, 15%)
  - Software quality management (16, 10%)
  - Software processes (24, 15%)
  - Software product management (16, 10%)
  - Software metrics, measurement, and analytical methods (24, 15%)
  - Software inspection, testing, verification, and validation (24, 15%)
  - Software audits (16, 10%)
  - Software configuration management (16, 10%)

Copyright © 2000 D. J. Law

Certified Quality Analyst

- Sponsor --- Quality Assurance Institute
- Eligibility --- (1) no degree, 6 years experience, (2) 2-year degree, 2 years experience, (3) 4-year degree, 0 years experience
- Application due --- 30 days before exam date
- Fee --- $250
- Format --- (1) 50 mc/tf/m, (2) 10 essays, (3) 50 mc/tf/m, (4) 10 essays
- Answer sheet --- non-standard, hand-graded, traditional, low-tech
- Type --- closed book
- Length --- 4 hours total, (1) 45 min, (2) 75 min, (3) 45 min, (4) 75 min
- Breaks --- 10 minute break between each of 4 parts
- Dates --- throughout the year at different locations nationwide
- Times --- usually 8:30 AM to 1:00 PM
- Location --- hosted by local QAI federation chapters (offices/hotels)

Copyright © 2000 D. J. Law

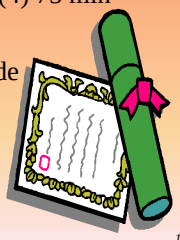
CQA

- Passing score --- 75% on each of 4 separate parts
- Expiration --- 1 year from exam date
- Renewal --- $20 maintenance fee, due by April 30th every year
- Reschedule --- 10 days prior, $25 fee; withdrawal: refund - $100 fee
- Retake --- Can resit up to 2 times for failed portion of exam for $50 fee
- Common body of knowledge (16 domains)
  - Communications
  - Human resource principles
  - Training and development
  - Quality management
  - Quality assurance
  - Quality control techniques
  - Quantitative methods
  - Reviews
  - Auditing and control
  - Testing
  - Vendor control
  - Standards
  - Disaster recovery
  - Management techniques
  - Change management
  - Principles of information systems

Copyright © 2000 D. J. Law

Certified Software Test Engineer

- Sponsor --- Quality Assurance Institute
- Eligibility --- (1) no degree, 6 years experience, (2) 2-year degree, 2 years experience, (3) 4-year degree, 0 years experience
- Application due --- 30 days before chosen exam date
- Fee --- $250
- Format --- (1) 50 mc/tf/m, (2) 10 essays, (3) 50 mc/tf/m, (4) 10 essays
- Answer sheet --- non-standard, hand-graded, traditional, low-tech
- Type --- closed book
- Length --- 4 hours total, (1) 45 min, (2) 75 min, (3) 45 min, (4) 75 min
- Breaks --- 10 minute break between each of 4 parts
- Dates --- throughout the year at different locations nationwide
- Times --- usually 8:30 AM to 1:00 PM
- Location --- hosted by local QAI chapters (offices/hotels)
- Passing score --- 75% on each of 4 separate parts



Copyright © 2000 D. J. Law

CSTE

- Expiration --- 1 year from exam date
- Renewal --- $20 maintenance fee, due by April 30th every year
- Reschedule --- 10 days prior, $25 fee; withdrawal: refund - $100 fee
- Retake --- Can resit up to 2 times for failed portion of exam for $50 fee
- Common body of knowledge (16 domains)
  - Communications
  - Professional development
  - Quality principles and concepts
  - Methods for software development and maintenance
  - Test principles and concepts
  - Verification and validation methods
  - Test management, standards, and environment
  - Reviews
  - Test tactics (approaches, tools, and environment)
  - Planning process
  - Test design
  - Performing tests
  - Defect tracking and management
  - Quantitative measurement
  - Test reporting
  - Improving the test process

Copyright © 2000 D. J. Law

Certified Software Test Professional

- Sponsor --- International Institute for Software Testing
- Requirements --- job experience and formal training
- Experience --- 1+ years of testing experience; letter from supervisor
- Training --- 10 days of classroom instruction
- Course length --- 1 or 2 days
- Tests --- given at the end of each 1-day or 2-day course
- Passing score --- 80% on each test
- Time limit --- 2 years to complete all courses
- Substitutions --- up to 2 days of non-IIST courses; also certain 1-day tutorials in IIST-sponsored conference (PSQT)
- Dates --- twice a month but different courses and different cities
- Locations --- Minneapolis, Las Vegas, Orlando, Boston, San Francisco

Copyright © 2000 D. J. Law

CSTP

- Tuition --- 1-day course, $450; 2-day course, $845; 3-day combination, $1,295; 4-day combination, $1,595
- Discounts --- 10% off when 5+ students from same company register
- Refunds -- at least 10 working days, $10 fee; less than 5 days, none
- Courses offered
  - Practical techniques for software quality assurance (2)
  - Testing web and e-business applications (2)
  - Software test planning and design (2)
  - Software requirements exploration and definition (2)
  - Software test automation and scripting techniques (2)
  - Software inspections and reviews (2)
  - Managing the software testing process (2)
  - Testing object-oriented systems (2)
- 9 additional 1- and 2-day courses available for on-site delivery only

Copyright © 2000 D. J. Law

Certified Product Specialist

- Sponsor --- Mercury Interactive Corporation
- Product --- TestSuite (WinRunner and TestDirector), LoadRunner (NT or Web); both are version-specific
- Eligibility --- no specific academic or experience requirements
- Recommended training --- beginning and advanced instructor-led courses
- Fee --- $2,500 for 1 week course; $500 exam grading fee
- Discounts --- 50% off exam fee for Quality Channel Partners
- Type --- self-paced, self-study, take-home project
- Format --- 3 to 6 chapters; each chapter short answers and programming specifications
- Length --- no time limit, but recommend 1 to 2 weeks full-time
- Dates --- apply anytime you are ready
- Location --- work on your own at office or home
- Expiration --- none, but version dependent

Copyright © 2000 D. J. Law

CPS

- Passing score --- 75%
- Renewal --- $100 annual fee
- Grading criteria:
  - Adherence to programming specifications
  - Adherence to programming guidelines
  - Portability
  - Maintainability
  - Documentation
  - Modularity
  - Best practices
- After submitting application, candidate receives certification package on CD by mail containing training materials, AUT, and project
- Application under test (AUT) is the sample flight reservation system
- When completed, return project by mail for evaluation and grading

Copyright © 2000 D. J. Law

Certified Product Instructor

- Sponsor --- Mercury Interactive Corporation
- Product --- TestSuite (WinRunner and TestDirector), LoadRunner (NT or Web); both are version-specific
- Eligibility --- (1) must be CPS certified, (2) must have had training within last 3 months, (3) product implementation experience
- Fee --- $3,500
- Discounts --- 50% off for Quality Channel Partners
- Type --- 3 day workshop and lab
- Format --- 33% trouble-shooting exercises, 47% presentation/teaching exercises, 20% final exam
- Dates --- apply anytime you are ready, event offered at least once per month but at different locations
- Location --- Mercury Interactive regional offices and corporate headquarters in Sunnyvale, California

Copyright © 2000 D. J. Law

CPI

- Expiration --- none, but version dependent
- Passing score --- 75%, graded on delivery of training presentation and final exam
- Restrictions --- if failed, must wait 6 months before retake
- Renewal --- $750 annual fee
- Day 1 - Problem Labs
  - Common installation and configuration problems
  - Common student errors
- Day 2 - Presentation Delivery
  - Deliver 1 lesson from training materials
  - Graded on knowledge of lessons, enthusiasm, presentation skills
- Day 3 - Final Exam
  - 25 questions, 2 points each
  - 3 categories: (1) Troubleshooting, (2) Labs, (3) Student frequently asked questions

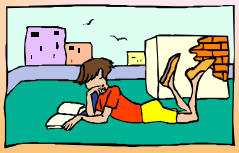
Copyright © 2000 D. J. Law

How to Study?

- Learn to select what is most important to learn
- Review by predicting questions on test
- Review by reorganizing subject matter into logical divisions (CBOK)
- Focus on concepts, principles, fundamentals
- Avoid trivia, incidentals, details, opinion, conjecture
- Know your test and quality vocabulary
- Take a review or refresher course
- Don't cram the week before

Copyright © 2000 D. J. Law

Some Successful Smart Study Strategies



- Use mnemonic devices
- Organize your study time
  - Example: QAI exams, 1 week, 1 domain
  - Example: ASQ exams, 1 month, 1 section of CBOK
- Identify your weak areas of domain or CBOK
- Take sample exams at every opportunity
- Determine your best place and time to study
- Get a good night's rest the night before
- Consider spending the night in a nearby hotel

Copyright © 2000 D. J. Law

How to Pass the Exam?

- General
  - Answer all questions
    - A missing answer is an incorrect answer
    - Prevents "answer skipping offset" on answer sheet
  - Preview entire exam first
  - ASQ exams allows feedback on questions
  - Time yourself, bring watch
  - Use red tape flags

- Multiple choice
  - Usually only one ***BEST*** answer
  - ASQ Exams: Which of the following are roles played by participants in an inspection meeting? I. Proctor. II. Moderator. III. Author. IV. Manager (a) I and II, (b) I, III, and IV, (c) I only, (d) I, II, III, and IV
  - Watch for ***NOT*** and ***EXCEPT***, double negatives and reverse logic

Copyright © 2000 D. J. Law

Test-Taking Tips, Traps, Tricks, Tactics, Techniques

- True/False
  - Observe qualifying words like **sometimes, never, always, usually, most, some, no, none, all, every**
- Essays
  - Usually based on scenario that requires problem-solving
  - Brainstorm first, jot down all ideas, concepts, points
  - Arrange in order of presentation
  - Use narrative style or bulleted style (recommended)
  - Use underline or all caps to emphasize key concepts
  - Content counts (objective), but neatness helps (subjective)
  - Understand instructional verbs like **compare, contrast, criticize, define, describe, discuss, evaluate, explain, illustrate, justify, narrate, outline, relate, state, summarize**

Copyright © 2000 D. J. Law

How to Keep Your Certification?

- Purpose
  - To maintain the integrity of your certification
  - To maintain the same level of proficiency originally demonstrated
- CSTE and CQA
  - Renewal period 1 year
  - Renewal fee $20 due by April 30th
  - Require 40 CPE credits
  - Each CPE credit = 1 hour of activity time
  - Example: 1 college course = 30 CPE credits
  - Work experience does ***NOT*** count

Copyright © 2000 D. J. Law

Maintaining Your Certification

- CSQE
  - Renewal period 3 years
  - Renewal fee $30 member, $50 non-member
  - Each activity is assigned a point value (RU)
  - Example: employment is 0.3 RU per month
  - Maximum allowed per activity (10.8 RU)
  - Require 18 RUs
- CSTP - none
- CPS and CPI - none (version specific)

Copyright © 2000 D. J. Law

Qualifying Activities

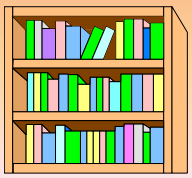
- **For ASQ (CSQE)**
  - Professional development
  - Paid employment
  - Instructor ***
  - Student
  - Professional society meetings
  - Committees
  - Other certifications
  - Proctoring exams ***
  - Viewing videotapes
  - Publishing books and articles
  - Presentation/speech
- **For QAI (CSTE, CQA)**
  - Educational activities
  - Publications
  - Presentations
  - Professional participation
  - Self-study courses
  - Unique and innovative job tasks

Copyright © 2000 D. J. Law

Resources

- **ASQ (CSQE)**
  American Society for Quality
  P. O. Box 3005
  Milwaukee, Wisconsin 53201-3005
  Phone: 414-272-8575
  Toll-free: 800-248-1946
  Fax: 414-272-1734
  Email: cs@asq.org
  Website: www.asq.org

- **QAI (CSTE, CQA)**
  Quality Assurance Institute
  7575 Dr. Phillips Boulevard
  Suite 350
  Orlando, Florida 32819-7273
  Phone: 407-363-1111
  Fax: 407-363-1112
  Email: certify@qaiusa.com
  Website: www.qaiusa.com



Copyright © 2000 D. J. Law

More Resources

- **IIST (CSTP)**
  International Institute for Software Testing
  8476 Bechtel Avenue
  Inver Grove Heights, MN 55076
  Phone: 651-306-1387
  Fax: 651-552-0791
  Email: sbrown@psqtconference.com
  Website: www.softdim.com/iist/

- **Mercury (CPS, CPI)**
  Mercury Interactive Corporation
  1325 Borregas Avenue
  Sunnyvale, California 94089
  Phone: 408-822-5200
  Toll-free: 800-TEST-911
  Fax: 408-822-5300
  Email: info@merc-int.com
  Website: www.merc-int.com

Copyright © 2000 D. J. Law

Really Reliable Reputable Resources

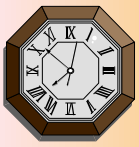
- Quality Council of Indiana
  - www.qualitycouncil.com
  - Primer and solutions text for CSQE
  - 400 practice exam questions on diskette
- Quality Assurance Institute
  - www.qaiusa.com
  - Study guides for CQA and CSTE
- Your online bookstore
  - 40+ titles on software testing
  - 15+ titles on software quality
- Your local ASQ section
  - CSQE review course

Copyright © 2000 D. J. Law

How to Get Started?

- Decide to get certify
  - Motivation, need, desire, commitment
- Select certification program
- Determine eligibility
- Complete the application
- Study and review
- Practice taking sample exams
- Take exam
- Reap the benefits and rewards

Copyright © 2000 D. J. Law



When is the Best Time?

- When do you want to …
- The time is now … it's never too late
- Remember:
  - If you do what you've been doing, you'll get what you've been getting (Anonymous)
  - I know of no more encouraging fact than the unquestionable ability of man to elevate his life by a conscious endeavor (Henry David Thoreau)

Copyright © 2000 D. J. Law

Questions and Answers

Copyright © 2000 D. J. Law

Certification Programs for the Software Quality and Test Professional

Presented by D. J. Law, CSQE, CSTE

13th Annual International Quality Week 2000
Paper 8M1
June 1 2000, Thursday 3:30 PM,
Management Track

Certification Programs for the Software Quality and Test Professional

Professional certification provides the certificant (certification holder) with the necessary credentials and knowledge to succeed in today's competitive marketplace. There are a number of certification programs available for the software quality and test professional: Certified Software Quality Engineer, Certified Quality Analyst, Certified Software Test Engineer, Certified Software Test Professional, Certified TestSuite Product Specialist, and Certified LoadRunner Product Instructor, just to name a few.

Through a series of nine Frequently Asked Questions (FAQs), you will learn a great deal about the current array of certification programs.

(1) What is certification? You will learn the meaning of certification.
(2) Why become certified? The benefits of being certified are discussed.
(3) How to get certified? The certification process is explained.
(4) Which certification? Details of six certifications are examined.
(5) How to study? Study strategies and techniques are presented.
(6) How to pass the exam? Test-taking skills and methodologies are explained.
(7) How to keep your certification? The certification maintenance process is examined.
(8) Where to go for resources? Various sources are listed for your convenience.
(9) How to get started? Steps to take to get started.

What is Certification?

Let's examine what it means to be certified. The dictionary defines the verb *certify* as "to meet a standard or confirm formally as true, accurate, or genuine". The American Society for Quality (a certifying organization) provides a working definition of the word *certification*: "Certification is the formal recognition that an individual has demonstrated a proficiency within and comprehension of a specified body of knowledge at a point in time".

Certification, then, is a formal proof of credibility in a profession, a demonstration of competency in a specified subject matter, and an official recognition of proficiency within a prescribed body of knowledge. Because certification is based on a common standard to evaluate knowledge and skill, it serves an update-to-date performance-based assessment of an individual's abilities and mastery in a particular area of expertise. Note however, it is neither licensure nor registration.

Why Become Certified?

The Benefits of Certification - For the Employee

There are tremendous advantages to being certified from an employee's perspective: career advancement potential, easier fulfillment of salary goals, faster promotions, hiring advantages, better job opportunities, professional development, greater marketability,

professional credibility, personal satisfaction, financial rewards, peer recognition, sense of professional accomplishment, and a resume booster, just to name a few. Certification literature from the American Society for Quality (ASQ) states that by becoming certified, you may realize your salary goals, advance within your organization, enhance your career and self-esteem, gain recognition for performance excellence, and affirms your commitment to quality. An ASQ survey published in the November 1998 issue of *Quality Progress* cites an average salary difference of $11,388 for those with a Certified Software Quality Engineer designation as compared to those not certified.

Benefits of Certification - For the Employer

There are an equal number of benefits from the employer's perspective: a tangible measure of an employee's capabilities, a reliable benchmark for hiring and promotion, ensures clients that employee has proven expertise, provides higher levels of service and customer satisfaction, assurance of quality when outsourcing services, recognition and reward for productive employees, objective evidence of an employee's abilities, increased employee productivity, adds value to key job functions, greater employee job commitment, validates training investment, and is a competitive market advantage when submitting bids or proposals, just to name a few. Certification indeed appears to be a win-win situation for the employee, employer, and client.

Disadvantages and Criticisms

Certification is not without drawbacks and disadvantages. Critics say that certification exams only test book knowledge (theory, not practice) and that no real-world skills are required. A certification is only valid for a specified period of time; each certificate has a limited useful lifespan. The applicant must maintain his or her certification, must have the time to study for the exam, and in some instances may pay a high application fee of several hundred dollars. For certifications based on a product from a vendor, regular re-certification is necessary due to a new release or version of the software. Employers who certify their employees may be afraid that the new certificant might leave the company after obtaining his or her certification. In the final analysis, it appears that the benefits outweigh the drawbacks by a wide margin.

How to Get Certified?

Certification Program Types

Certification programs can be grouped according to the type of the issuing certifying entity: organization-based, institution-based, or vendor-based. Organization-based are those professional societies and associations that serve a particular industry segment or profession. The certifications are vendor-independent, that is, they do not focus on any one particular product, manufacturer, or vendor. The exams and tests of organization-based certifications concentrate on a prescribed common body of knowledge (CBOK) that is defined by the organization for that specific profession or discipline. Examples of these professional organizations and their certifications are: the American Society for

Quality (ASQ), which issues the Certified Software Quality Engineer (CSQE) designation, the Quality Assurance Institute (QAI), which grants the Certified Software Test Engineer (CSTE) and Certified Quality Analyst (CQA) designations, and the British Computer Society (BCS), which offers the Certificate in Software Testing (CST) certification.

Institution-based programs are those offered by educational institutions, either for profit or not. These programs are vendor-neutral and no renewal or maintenance is required once certification is obtained. Certification is obtained by both class attendance and the passing of a series of tests given after each class. An example is the Certified Software Test Professional (CSTP) certification program offered by the International Institute for Software Testing (IIST).

Vendor-based certification programs are those offered by manufacturers or vendors of a particular product. The certification always applies to a specific version or release of a product. The exams are not based on an industry-wide common body of knowledge, but the ability to use the product to the fullest extent in a real-word situation. Examples are the Certified Product Specialist (CPS) and Certified Product Instructor (CPI) certifications offered by Mercury Interactive Corporation, and the Certified Segue Scalability Consultant (CSSC) designation issued by Segue Corporation, both makers of automated software test tools.

How to Become Certified?

The certification process usually consists of several steps: eligibility determination, application submittal, eligibility notification, examination preparation, the taking of the exam, and results notification.

Most certification programs have eligibility requirements that must be satisfied in order to even apply for admission to the certification program. Some of these qualifications are a baccalaureate degree, a certain number of years of paid experience, proof of professionalism in the form of letters or membership in a professional society, and your commitment to subscribe to a professional code of ethics in the form of a signature. Some certifying bodies may allow the candidate to substitute advanced degrees for lack of the prerequisite years of professional experience.

An applicant may submit the certification application either online at the certifying entity's website, or by fax, or via regular mail. Some organizations may require resumes, letters of character references and work ethics, or other supplemental documentation. After the application is processed, the applicant is notified of his or her eligibility to sit for the certification examination. In the mean time, the candidate then prepares for the examination by study the common body of knowledge, taking certification review courses, and reading the recommended books. Notification of test results is usually sent to the candidate 4 to 6 weeks after the examination.

Which Certification?

Note that a definite distinction is made between software quality assurance and software testing.

There are two professional certifications currently available for the software quality practitioner: (1) the Certified Software Quality Engineer (CSQE) from the American Society for Quality (ASQ), and (2) the Certified Quality Analyst (CQA) from the Quality Assurance Institute (QAI). Note that three other quality-related certifications are available from ASQ: (1) the Certified Quality Manager (CQMgr), (2) the Certified Quality Engineer (CQE), and (3) the Certified Quality Auditor (CQA, not to be confused with the CQA from QAI). However, these three certifications are not software-related, only quality-related.

For the software test professional, there are presently five certifications available: (1) the Certified Software Test Engineer (CSTE) from the Quality Assurance Institute (QAI), (2) the Certified Software Test Professional (CSTP) from the International Institute for Software Testing (IIST), (3) the Certified Product Specialist (CPS) in either the TestSuite (WinRunner and TestDirector combined) or the LoadRunner products, both from Mercury Interactive Corporation, (4) the Certified Product Instructor (CPI) in either the TestSuite (WinRunner and TestDirector combined) or the LoadRunner products, both from Mercury Interactive Corporation, (5) the Certified Segue Scalability Consultant (CSSC) from Segue Corporation, and (6) the Certificate in Software Testing (CST) from the British Computer Society.

The Certified Software Quality Engineer (CSQE)

Sponsor: the American Society for Quality

Eligibility: a minimum of 8 years of work experience in one or more areas of the CSQE CBOK. A minimum of 3 of 8 years must be in a decision-making position. 1 year may be waived if the applicant has a diploma from a technical or trade school. 2 years waived for an associate degree. 4 years waived for a bachelor's degree. 5 years waived for a Master's or doctorate degree.

Application Due: approximately 2 months before a scheduled exam date.

Fee: $90 for members of ASQ, $195 for non-members.

Exam Format: 160 multiple choice questions, not categorized according to areas in the CBOK, random order.

Answer Sheet: standard format (4 choices a through d), optical scanned, mark with No. 2 pencil.

Exam Type: open book, can bring reference materials, but no sample questions/answers are allowed (if book contains questions/answers, they must be completely covered).

Permissible Materials: basic 4-function calculator (non-programmable, non-graphing), reference books (proctor must review books).

Exam Length: 4 hours

Breaks: no scheduled breaks, permitted as necessary at your convenience (snacks, bathroom, etc.)

Exam Dates: twice a year, usually the first Saturday in June and the first Saturday in December, in the year 2000, the dates are Saturday June 3 and Saturday December 2, check ASQ website for latest dates.

Exam Times: usually in the morning, from 8:00 AM to 12:00 Noon.

Exam Location: at various offices/hotels/schools hosted by local ASQ chapters and proctored by local ASQ members.

Passing Score: scale score of 550 minimum (range from 200 to 750).

Certificate Expiration: 3 years from examination date, which is also used as the certification date.

Certificate Renewal: every 3 years, $30 for ASQ members, $50 for non-members.

Application Refunds: yes, but less a $20 processing fee.

Exam Retake: no limit on number of times, but must pay a resit fee, and must retake exam within 2 years of last attempt.

Results: if pass, no score is given, only that you are certified. If fail, score and analysis is provided.

The Common Body of Knowledge (8 areas):

(1) General knowledge, conduct, and ethics (24 questions, 15% of exam)
(2) Software quality management (16 questions, 10% of exam)
(3) Software processes (24 questions, 15% of exam)
(4) Software product management (16 questions, 10% of exam)
(5) Software metrics, measurement, and analytical methods (24 questions, 15% of exam)
(6) Software inspection, testing, verification, and validation (24 questions, 15% of exam)
(7) Software audits (16 questions, 10% of exam)
(8) Software configuration management (16 questions, 10% of exam)

A CSQE textbook, *The CSQE Primer*, is available from the Quality Council of Indiana, 602 West Paris Avenue, West Terre Haute, Indiana 47885, phone 812-533-4215, fax 812-533-4216, email qci@qualitycouncil.com, website http://www.qualitycouncil.com. Price

$65. A solution text and a 400 question diskette are also available for $35 and $60 respectively.

A CSQE correspondence course and reference book is available from TBSC Inc., P. O. Box 23436, Rochester, New York 14692, phone 716-385-7570, fax 716-385-5662, email info@tbscinc.com, website http://www.planetquality.com.

ASQ's most recently certified software quality engineers recommends the following books for study:

(1) *Software Engineering: A Practitioner's Approach*, by Roger S. Pressman, 1997, 4th edition, McGraw-Hill, ISBN 0-07-052182-4, current list price $85.95.
(2) *Managing the Software Process*, by Watts S. Humphrey, 1989, Addison-Wesley, ISBN 0-201-18095-2, current list price $59.95.
(3) *Metrics and Models in Software Quality Engineering*, by Stephen H. Kan, 1995, Addision-Wesley, ISBN 0-201-63339-6, current list price $43.95.

For a 12-page application and certification information brochure:
American Society for Quality, P. O. Box 3005, Milwaukee, Wisconsin 53201-3005, phone 414-272-8575, toll-free 800-248-1946, fax 414-272-1734, email cs@asq.org, website http://www.asq.org.

The Certified Quality Analyst (CQA)

Eligibility: 6 years of work experience if no Bachelor's degree, 2 years of work experience if applicant has a 2-year degree, 0 years of work experience if Bachelor's degree.

Application Due: 30 days before selected exam date.

Fee: $250.

Exam Format: 4 parts, part 1 contains 50 multiple choice, true/false, and matching questions, part 2 contains 8 to 10 essay/short answer questions, part 3 contains 50 multiple choice, true/false, and matching questions, part 4 contains 8 to 10 essay/short answer questions.

Answer Sheet: traditional, low-tech, hand-graded, not optical-scanned, non-standard format (number of answer choices per question varies).

Exam Type: Closed book.

Exam Length: 4 hours total, part 1 is 45 minutes, part 2 is 75 minutes, part 3 is 45 minutes, part 4 is 75 minutes.

Breaks: 10 minute break between each of 4 parts.

Exam Dates: throughout the entire year in various cities nationwide, and at QAI-sponsored conferences, check QAI website for latest dates.

Exam Times: usually in the morning, from 8:30 AM to 1:00 PM.

Exam Location: at various offices/hotels/schools hosted by local QAI regional offices and federation chapters.

Passing Score: 75% on each of 4 parts, each part is graded individually.

Certificate Expiration: 1 year from examination date, which is also used as the certification date.

Certificate Renewal: every year, $20 maintenance fee, due by April 30th every year.

Exam Rescheduling: 10 days prior to exam - $25 fee, withdrawal – refund minus $100 fee.

Exam Retake: Can resit up to 2 times for failed portion of exam for $50 fee.

Results: separate score is provided for each of 4 parts, notification in 6 to 8 weeks.

The Common Body of Knowledge (16 domains):

(1) Communications
(2) Human resource principles
(3) Training and development
(4) Quality management
(5) Quality assurance
(6) Quality control techniques
(7) Quantitative methods
(8) Reviews
(9) Auditing and control
(10) Testing
(11) Vendor control
(12) Standards
(13) Disaster recovery
(14) Management techniques
(15) Change management
(16) Principles of information systems

A CQA Study Guide is automatically sent to the applicant upon receipt of the application form. This reference contains detailed information on the 16 domains that make up the QAI CQA common body of knowledge.

In addition to the above CQA Study Guide, you may wish to study the following books on software quality:

(1) *Quality Assurance for Information Systems*, by William E. Perry, 1991, John Wiley & Sons, ISBN 0-471-58804-0.
(2) *Handbook of Software Quality Assurance*, edited by G. Gordon Schulmeyer and James I. McManus, 3rd edition, 1999, Prentice Hall, ISBN 0-13-010470-1.
(3) *Software Quality: Concepts and Plans*, by Robert H. Dunn, 1990, Prentice Hall, ISBN 0-13-820283-4.
(4) *Software Quality: Analysis and Guidelines for Success*, by Capers Jones, 1997, International Thomson Computer Press, ISBN 1-85032-867-6.

For application form and certification information: Quality Assurance Institute, 7575 Dr. Phillips Boulevard, Suite 350, Orlando, Florida 32819, phone 407-363-1111, fax 407-363-1112, email certify@qaiusa.com, website http://www.qaiusa.com.

The Certified Software Test Engineer (CSTE)

Eligibility: 6 years of work experience if no Bachelor's degree, 2 years of work experience if applicant has a 2-year degree, 0 years of work experience if Bachelor's degree.

Application Due: 30 days before selected exam date.

Fee: $250.

Exam Format: 4 parts, part 1 contains 50 multiple choice, true/false, and matching questions, part 2 contains 8 to 10 essay/short answer questions, part 3 contains 50 multiple choice, true/false, and matching questions, part 4 contains 8 to 10 essay/short answer questions.

Answer Sheet: traditional, low-tech, hand-graded, not optical-scanned, non-standard format (number of answer choices per question varies).

Exam Type: Closed book.

Exam Length: 4 hours total, part 1 is 45 minutes, part 2 is 75 minutes, part 3 is 45 minutes, part 4 is 75 minutes.

Breaks: 10 minute break between each of 4 parts.

Exam Dates: throughout the entire year in various cities nationwide, and at QAI-sponsored conferences, check QAI website for latest dates.

Exam Times: usually in the morning, from 8:30 AM to 1:00 PM.

Exam Location: at various offices/hotels/schools hosted by local QAI regional offices and federation chapters.

Passing Score: 75% on each of 4 parts, each part is graded individually.

Certificate Expiration: 1 year from examination date, which is also used as the certification date.

Certificate Renewal: every year, $20 maintenance fee, due by April 30th every year.

Exam Rescheduling: 10 days prior to exam - $25 fee, withdrawal – refund minus $100 fee.

Exam Retake: Can resit up to 2 times for failed portion of exam for $50 fee.

Results: separate score is provided for each of 4 parts, notification in 6 to 8 weeks.

The Common Body of Knowledge (16 domains):

(1) Communications
(2) Professional development
(3) Quality principles and concepts
(4) Methods for software development and maintenance
(5) Test principles and concepts
(6) Verification and validation methods
(7) Test management, standards, and environment
(8) Reviews
(9) Test tactics (approaches, tools, and environment)
(10) Planning process
(11) Test design
(12) Performing tests
(13) Defect tracking and management
(14) Quantitative measurement
(15) Test reporting
(16) Improving the test process

A CSTE Study Guide is automatically sent to the applicant upon receipt of the application form. This reference contains detailed information on the 16 domains that make up the QAI CSTE common body of knowledge.

In addition to the above CSTE Study Guide, you may wish to study the following books on software testing:

(1) *The Art of Software Testing,* by Glenford J. Myers, 1979, John Wiley & Sons, ISBN 0-471-04328-1.

(2) *Testing Computer Software*, by Cem Kaner, Jack Falk, and Hung Quoc Nguyen, 2nd edition, 1999, John Wiley & Sons, ISBN 0-471-35846-0.
(3) *Managing the Testing Process*, by Rex Black, 1999, Microsoft Press, ISBN 0-7356-0584-X.
(4) *Software Testing in the Real World*, by Edward Kit, 1995, ACM Press (Addison-Wesley), ISBN 0-201-87756-2.
(5) *Software Test Automation*, by Mark Fewster and Dorothy Graham, 1999, ACM Press (Addison-Wesley), ISBN 0-201-33140-3.
(6) *Automated Software Testing*, by Elfriede Dustin, Jeff Rashka, and John Paul, 1999, Addison-Wesley, ISBN 0-201-43287-0.

For application form and certification information: Quality Assurance Institute, 7575 Dr. Phillips Boulevard, Suite 350, Orlando, Florida 32819, phone 407-363-1111, fax 407-363-1112, email certify@qaiusa.com, website http://www.qaiusa.com.

Certified Software Test Professional (CSTP)

Sponsor: International Institute for Software Testing

Eligibility/Requirements: job experience and formal training

Experience: minimum of 1 year software testing experience, letter from supervisor or manager is necessary.

Training: Attendance of 10 days of classroom instruction is required.

Course Length: 1 or 2 day instructor-led classes.

Exams: given at the end of each 1-day or 2-day course.

Passing score: 80% on each test.

Time limit: 2 years to complete all courses.

Substitutions: up to 2 days of non-IIST courses. Also certain 1-day tutorials in IIST-sponsored conference (PSQT) may be substituted.

Course dates: usually twice a month, different courses in different cities.

Course locations: usually in Minneapolis, Las Vegas, Orlando, Boston, and San Francisco.

Tuition: 1-day course - $450. 2-day course - $845. 3-day combination - $1,295. 4-day combination - $1,595.

Discounts: 10% when 5+ students from same company register.

Refunds: at least 10 working days prior to class - $10 fee. Less than 5 days – no refunds.

2-day courses offered:

(1) Practical techniques for software quality assurance
(2) Testing web and e-business applications
(3) Software test planning and design
(4) Software requirements exploration and definition
(5) Software test automation and scripting techniques
(6) Software inspections and reviews
(7) Managing the software testing process
(8) Testing object-oriented systems

9 additional 1-day and 2-day courses are available for on-site delivery only.

Books are included as part of the courseware.

In addition to the provided classroom materials, you may wish to study the following books on software testing:

(1) *The Art of Software Testing,* by Glenford J. Myers, 1979, John Wiley & Sons, ISBN 0-471-04328-1.
(2) *Testing Computer Software,* by Cem Kaner, Jack Falk, and Hung Quoc Nguyen, 2nd edition, 1999, John Wiley & Sons, ISBN 0-471-35846-0.
(3) *Managing the Testing Process,* by Rex Black, 1999, Microsoft Press, ISBN 0-7356-0584-X.
(4) *Software Testing in the Real World,* by Edward Kit, 1995, ACM Press (Addison-Wesley), ISBN 0-201-87756-2.
(5) *Software Test Automation,* by Mark Fewster and Dorothy Graham, 1999, ACM Press (Addison-Wesley), ISBN 0-201-33140-3.
(6) *Automated Software Testing,* by Elfriede Dustin, Jeff Rashka, and John Paul, 1999, Addison-Wesley, ISBN 0-201-43287-0.

For additional information and course registration form:
International Institute for Software Testing, 8476 Bechtel Avenue, Inver Grove Heights, Minnesota 55076, phone 651-306-1387, fax 651-552-0791, email sbrown@psqtconference.com, website http://www.softdim.com/iist/.

Certified Product Specialist (CPS)

Sponsor: Mercury Interactive Corporation

Product: TestSuite (WinRunner and TestDirector), or LoadRunnerNT. Both are version specific. Latest version is 6.0 for both products.

Eligibility: no specific academic or work experience requirements.

Recommended Training: beginning and advanced instructor-led courses offered by Mercury Interactive or its authorized training partners.

Fee: $2,500 for a 5-day training course (recommended), $500 exam grading fee.

Discounts: 50% off for channel partners.

Exam type: self-paced, self-study, take-home project.

Exam format: 3 to 6 chapters, each chapter has preparation questions, programming specifications and guidelines, and final questions. Questions are short answer.

Exam length: recommend approximately 160 hours to complete exam. Applicant has up to 3 months maximum.

Exam dates: apply anytime you are ready.

Exam location: work on your own at office or home.

Expiration: none. Certification is version dependent. Certificant is expected to take add-on exam when new version is deployed.

Passing score: 75%

Renewal: $100 annual fee.

Grading criteria:

(1) Adherence to programming guidelines
(2) Adherence to programming specifications
(3) Portability
(4) Maintainability
(5) Documentation
(6) Modularity
(7) Best practices

After submitting application form, candidate receives certification package on CD by mail containing training materials, the application under test (AUT), and the programming project. When the project is completed, applicant returns project my mail for evaluation and grading. Applicant is notified of results within 30 days.

For application form and certification information:
Mercury Interactive Corporation, 1325 Borregas Avenue, Sunnyvale, California 94089, phone 408-822-5200, toll-free 800-TEST-911, fax 408-822-5300, email info@merc-int.com, website: www.merc-int.com.

Certified Product Instructor (CPI)

Sponsor: Mercury Interactive Corporation

Product: TestSuite (WinRunner and TestDirector), or LoadRunnerNT. Both are version specific. Latest version is 6.0 for both products.

Eligibility: must be CPS certified. Must have had training within the last 3 months. Must have product implementation experience.

Fee: $3,500.

Discounts: 50% off for channel partners.

Exam type: 3-day workshop and lab.

Exam format: 33% trouble-shooting exercises, 47% presentation/teaching exercises, 20% final exam.

Exam dates: apply anytime you are ready, CPI certification event is offered at least once per month, but maybe at different locations.

Exam location: Mercury Interactive regional offices and corporate headquarters in Sunnyvale, California.

Expiration: none. Certification is version dependent. Certificant is expected to take add-on exam when new version is deployed.

Passing score: 75%, graded on delivery of training presentation and final exam.

Renewal: $750 annual fee.

Restrictions: if failed, must wait 6 months before retake.

Day 1: Problem Labs – common installation and configuration problems, common student errors.

Day 2: Presentation Delivery – deliver one lesson from training materials, graded on knowledge of lessons, enthusiasm, and presentation skills.

Day 3: Final Exam – 25 questions, 2 points each. 3 categories: (1) troubleshooting, (2) labs, and (3) student frequently asked questions (FAQs).

For application form and certification information:

Mercury Interactive Corporation, 1325 Borregas Avenue, Sunnyvale, California 94089, phone 408-822-5200, toll-free 800-TEST-911, fax 408-822-5300, email info@merc-int.com, website: www.merc-int.com.

How to Study?

Preparing for the examination is the most difficult and time-consuming part of the certification process. Here are some time-proven study techniques and strategies:

(1) Learn to select what is most important to learn.
(2) Review by predicting questions on the test.
(3) Review by organizing subject matter into logical divisions (CBOK areas or domains).
(4) Focus on concepts, principles, and fundamentals.
(5) Avoid trivia, incidentals, details, opinions, and conjecture.
(6) Know your test and quality vocabulary.
(7) Take a review or refresher course.
(8) Don't cram the week before the exam.
(9) Use mnemonic devices to help you memorize specific facts.
(10) Organize your study time. Example: QAI exams – 1 domain per 1 week. ASQ exam – 1 area of CBOK per 1 month.
(11) Identify your weak areas of domain or CBOK early.
(12) Take sample exams at every opportunity.
(13) Determine your best place and time to study.
(14) Get a good night's rest the night before the examination.
(15) Consider spending the night in a nearby hotel to minimize stress.

How to Pass the Exam?

An applicant may know everything on an exam but still not do well because he or she might lack test-taking skills. Here are some hints and tips to ensure that you perform your utmost during your test:

(1) Answer all questions. A missing answer is an incorrect answer.
(2) Always preview the entire exam first.
(3) Some exams, such as those from ASQ, allows for feedback on questions.
(4) Time yourself, bring a watch.
(5) Use tape flags to mark questions that you wish to come back to.
(6) For multiple choice questions, there is usually 1 best answer.
(7) Watch for **Not**, **Except, Never, Always, No, None, All, Every**.
(8) For short answer questions, use bulleted style when answering. Don't use complete sentences.
(9) Make sure you understand instructional verbs like **compare, contrast, define, describe, discuss, evaluate, explain, illustrate, justify, narrate, outline, state, summarize.**

How to Keep Your Certification?

All organization-based certifications require the certificant to participate in some type of certification maintenance program in order to remain certified. This is usually accomplished by participating in some kind of professional development activities. Institution-based certificates usually do not require any "maintenance". For vendor-based certifications, usually the certificant "upgrades" to the next product version or release in order to stay current. This is accomplished by taking an "upgrade" or add-on exam.

Details of the CSTE and CQA certification maintenance from QAI are as follows:

(1) Renewal period is 1 year.
(2) Renewal fee is $20, due by April 30th of each year.
(3) Requires 40 CPE (continuing professional education) credits
(4) Each CPE credit is equivalent to 1 hour of activity time
(5) Unlike the ASQ certification maintenance, work experience does **NOT** count toward CPE credits.
(6) Example: 1 college course = 30 CPE credits.

Qualifying activities for the CSTE and CQA include:

(1) Educational activities
(2) Publications
(3) Presentations
(4) Professional participation
(5) Self-study courses
(6) Unique and innovative job tasks

Details of the CSQE certification maintenance from ASQ are as follows:

(1) Renewal period is 3 years.
(2) Renewal fee is $30 for ASQ members, $50 for non-members.
(3) Each activity is assigned a point value called RU (recertification unit).
(4) Example: employment is 0.3 RU per month.
(5) There is a maximum RU allowed per activity.
(6) Program requires 18 RUs every 3 years.

Qualifying activities for the CSQE maintenance program include:

(1) Professional development
(2) Paid employment
(3) Instructor
(4) Student
(5) Professional society meetings
(6) Committees
(7) Other certifications

(8) Proctoring exams
(9) Viewing videotapes
(10) Publishing books and articles
(11) Presentation/Speech

The CSTP, CPS, and CPI certifications do not require professional activities for certification maintenance.

Where to Go for Resources?

ASQ (CSQE certification):

American Society for Quality
P. O. Box 3005
Milwaukee, Wisconsin 53201-3005
Phone: 414-272-8575
Toll-free: 800-248-1946
Fax 414-272-1734
Email: cs@asq.org
Website: http://www.asq.org

QAI (CQA and CSTE certifications):

Quality Assurance Institute
7575 Dr. Phillips Boulevard
Suite 350
Orlando, Florida 32819-7273
Phone: 407-363-1111
Fax: 407-363-1112
Email: certify@qaiusa.com
Website: http://www.qaiusa.com

IIST (CSTP certification):

International Institute for Software Testing
8476 Bechtel Avenue
Inver Grove Heights, Minnesota 55076
Phone: 651-306-1387
Fax: 651-552-0791
Email: sbrown@psqtconference.com
Website: http://www.softdim.com/iist/

Mercury Interactive (CPS and CPI certifications):

Mercury Interactive Corporation
1325 Borregas Avenue

Sunnyvale, California 94089
Phone: 408-822-5200
Toll-free: 800-TEST-911
Fax: 408-822-5300
Email: info@merc-int.com
Website: www.merc-int.com

How to Get Started?

You can become certify in seven easy steps. Here's how:

(1) Decide to get certify (motivation, need, and desire).
(2) Select your certification program (CSQE, CSTE, CQA, etc.).
(3) Determine the program's eligibility requirements.
(4) Complete the application online, fax, or mail it in.
(5) Study and review and common body of knowledge.
(6) Practice taking sample exam.
(7) Take the certification exam.

Finally, you can sit back and reap the benefits and rewards that certification may bring. Good luck.

QW2000 QuickStart 8Q

Mr. Thomas Drake
(CRTI)

Testing Network Based Software Systems -- The Future Frontier

BACK TO QW2000 PROGRAM

Presentation Abstract

The increasing cost and complexity of software development for enterprise electronic commerce and other network based applications and solutions is leading software organizations in the industry to search for new and innovative ways for improving the quality of the software they develop and deliver.

However, the overall process is only as strong as its weakest link. This critical link, I would argue, is software quality engineering as an activity and as a process and testing is the key instrument for making that happen. But what should testing measure in these new and emerging environments?

As conventional programs get increasingly complex the corresponding cost, development time, and defect rates are climbing. It is reasonable to expect, based on current trends, that this problem will only grow worse with the advent of even more complex, interactive, and interdependent programming and development environments for network based information technology solutions.

However, it also appears that the more intelligent and dynamic we try to make computers and our programs the more they slow down and break. Conventional software development and testing practices and processes are increasingly strained and coming up short in the face of this growing and alarming complexity.

So what does a tester do in the face of all this complexity? A tester must take a destructive attitude toward the code and the associated system, knowing that this activity is, in the end, constructive. Testing is a negative activity conducted with the explicit intent and purpose of creating a stronger software product and is operatively focused on the "weak links" in the software. So if a larger software quality engineering process is established to prevent and find errors, we can then begin to change our collective mind-set about how to ensure the quality of the software developed.

The other problem is that we really never have enough time to test, anyway. We need to change our conceptual understanding about the development and delivery environment and use the testing time we do not have time for and apply it to the earlier phases of the software development life cycle. We need to think about testing the first day we think about the system, rather then viewing testing as something that takes place after development, and focus instead on the testing of everything. This includes the concept of operations, the requirements and specifications, the design, the code, and of course, the tests! But what are we actually testing and how do we determine those weak links?

This presentation would introduce a biologically inspired model-based conceptual framework for network-centric testing. It involves an architecture that can deal with computers and software viewed as a system of interactive and dynamic behavioral objects rather than strictly for data processing and number crunching that are themselves part of a larger system.

This conceptual framework for testing would allow for testing a range of behaviors and outcomes and the possible interactions for these application objects without the necessity for fully understanding them in advance! This could permit testing the fundamental structure of the program and the application environment and the executable functional mechanisms underneath as a testing framework that is anchored in living systems theory. It permits an "inside out" approach such that testing is based on the "genetic" makeup of the expected and anticipated dynamic "state" attributes and characteristics of the system using its own behavioral specifications as the test instruments for locating and stimulating the "weak" links.

About the Author

Mr. Drake is a software systems quality specialist and management and information technology consultant for Integrated Computer Concepts, Inc. (ICCI) in the United States. He currently leads and manages a U.S. government agency-level Software Engineering Knowledge Based CenterÆs quality engineering initiative.

As part of an industry and government outreach/partnership program, he holds frequent seminars and tutorials covering code analysis, software metrics, OO analysis for C++ and Java, coding practice, testing, best current practices in software development, the business case for software engineering, software quality engineering practices and principles, quality and test architecture development and deployment, project management, organizational dynamics and change management, and the people side of information technology.

He is the principal author of a chapter on "Metrics Used for Object-Oriented Software Quality" for a CRC Press Object Technology Handbook published in December of 1998. In addition, Mr. Drake is the author of a theme article entitled: "Measuring Software Quality: A Case Study" published in the November 1996 issue of IEEE Computer. He also had the lead, front page article published in late 1999 for Software Tech News by the US Department of Defense Data & Analysis Center for Software (DACS) entitled: *Testing Software Based Systems: The Final Frontier*.

Mr. Drake is listed with the International WhoÆs Who for Information Technology for 1999, is a member of IEEE and an affiliate member of the IEEE Computer Society. He is also a Certified Software Test Engineer (CSTE) from the Quality Assurance Institute (QAI).

BACK TO QW2000 PROGRAM

Testing Network Based Software Systems

The Future Frontier

Thomas A. Drake

Quality Architect/Software Anthropologist

Enterprise Management and Information Technology Consulting

Certified Software Test Engineer (CSTE)

Integrated Computer Concepts, Inc. (ICCI)

Quality Week 2000

thomas.drake@integratedcc.com

© Copyright 2000 by Thomas Drake. All Rights Reserved

Introduction and Overview

Setting the Stage

A Little Bit of Background

The Challenge for the Enterprise

- ☑ **Boston Consulting Group, Inc. Study - Winter 2000**
- ☑ **100 executives in industries from manufacturing, telecommunications, to financial services**
  - ✓ **Only one of three enterprise management software implementations was successful in terms of cutting costs, meeting business goals and showing a tangible financial impact**
  - ✓ **60% of respondents said the new systems had helped**
  - ✓ **Just over 1/2 said it met their business goals**
  - ✓ **1/3 said software vendors encouraged "excessive" spending**
  - ✓ **12% of the vendors were "fired"**

Sobering Numbers (1)

- ☑ **Standish Group Study in late 1995**
  - ✓ **USA spent $81 billion for cancelled software projects**
  - ✓ **$59 billion for projects completed late, over budget, or lacking key or essential functionality**
  - ✓ **Only 16.2% of projects were completed on time & within budget with only 9% in larger companies**
  - ✓ **In larger companies completed projects had an average of 42% of the desired functionality**
- ☑ **Causal Analysis?**
  - ✓ **Lack of user/customer input (requirements not understood/captured)**
  - ✓ **Incomplete requirements and specifications**
  - ✓ **Changing requirements and specifications/requirements creep (+ & -)**

Sobering Numbers (3)

- ☑ **Standish Group study updated 4 years later…**
  - ✓ **Over $250 billion spent annually on IT application development**
  - ✓ **31% of all projects are cancelled before completion**
  - ✓ **88% of all projects are over schedule, over budget, or both**
  - ✓ **For every 100 projects started, there are 94 restarts**
  - ✓ **Average cost overrun is 189% of original estimates**
  - ✓ **Average time overrun is 222% of original estimates**
- ☑ **Even with strong technical skills many project managers and project team members find themselves in over their heads**
  - ✓ **On projects out of control**
  - ✓ **Without the necessary business organization and political skills**
- ☑ **Faith is not a management method!**

Historical Evolution

E = Enterprise
T = Technology

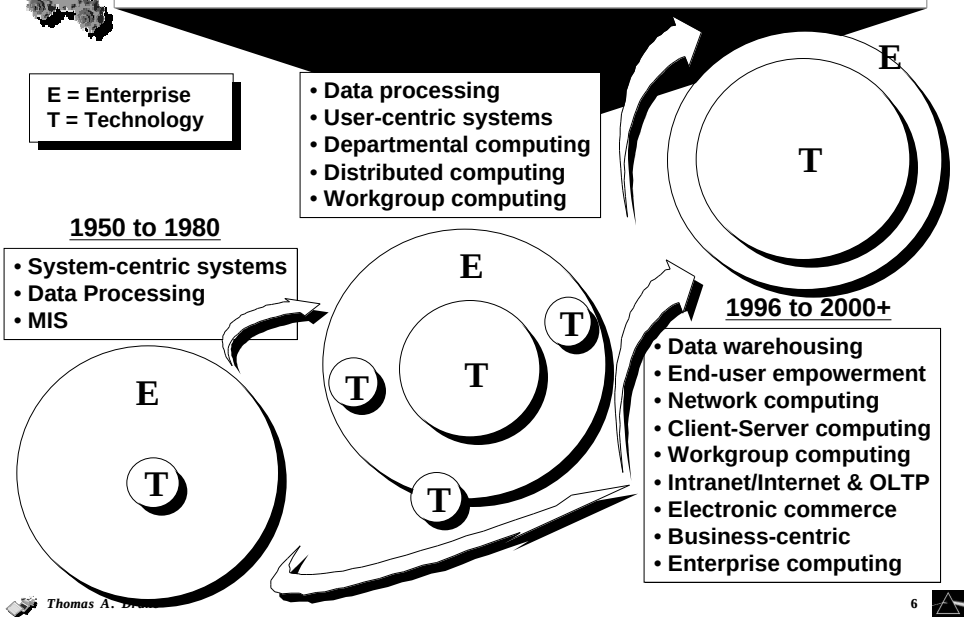
1950 to 1980

- System-centric systems
- Data Processing
- MIS

- Data processing
- User-centric systems
- Departmental computing
- Distributed computing
- Workgroup computing

1996 to 2000+

- Data warehousing
- End-user empowerment
- Network computing
- Client-Server computing
- Workgroup computing
- Intranet/Internet & OLTP
- Electronic commerce
- Business-centric
- Enterprise computing



Running Rampant in Cyberspace

Missing the Obvious

Forgetting the Fundamentals

Shortened Product Release Schedules

Testing Never Stops when Content is Continuously Changing

Web Time is ALL the Time!

Internet/Cyberspace Quality (1)

- ☑ **Internet and World Wide Web are the next logical step**
  - ✓ **The Web is huge - 24/7 & 365**
  - ✓ **Part of our culture**
  - ✓ **Look at any ad on television and you will see a Web address**
- ☑ **Endless possibilities for the Web and they go far beyond just static advertising**
  - ✓ **A cybersurfer can find information on almost anything, play games, download software, interact with "live" humans**
  - ✓ **Web is the "Great Equalizer"**
  - ✓ **A one or two person company can look like a multibillion dollar corporation to its customers**

Internet/Cyberspace Quality (2)

- ☑ **Competing for shelf space versus competing for Web space**
  - ✓ **Producing a shift toward the Web and more and more testing of internet-based software**
- ☑ **More complex programs and applications like Java and Shockwave are emerging**
- ☑ **Testing on multiple platforms and operating systems**
  - ✓ **Different Internet service providers and methods of connecting to the Internet**
  - ✓ **Can't afford to put out a shoddy product on the Web**
- ☑ **Quality is STILL quality and even more so on the Web**
  - ✓ **Stakes are much higher in this kind of "operating" environment and bugs/problems/defects are much more visible**

Where We Are Going

- ☑ **Integrated Component Design and Code**
- ☑ **Software Reuse**
- ☑ **Network Common Services**
- ☑ **XML & Web Technologies**
- ☑ **Intelligent Agents**
- ☑ **Data Visualization**
- ☑ **Security**
- ☑ **Private/Public Network**
- ☑ **Web-based Workflow**

Our IT Love Affair

TECHNO CULTURE

But Seriously...

BUT SERIOUSLY...

The IT Adventure - Software is at the Heart of IT!

- ☑ **Internet/Virtual Reality**
- ☑ **Surfing the Planet/Real-Time Data Flows**
- ☑ **E-mailing/Electronic Communication and Messaging**
- ☑ **Critical Reliance on Cyberspace**
- ☑ **Satellites/Computer Networks/Digital broadcasting/Cellular/Interactive TV**
- ☑ **Entertainment and Games!**

And Only Getting MORE Connected

- ☑ The future is not what it used to be!
- ☑ Pace of change (Default Standard)
- ☑ Time, communications, space (compressing)
- ☑ Implication - Speed of light access and impact
- ☑ All linked together in one dense, interconnected web of information and data!

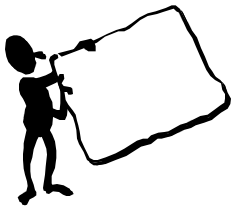
Our Reliance on Software is Huge!

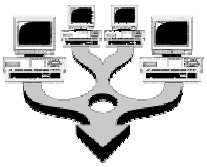
- ☑ The network is the program
- ☑ Increased recognition that software is a critical national resource and vital to the economy

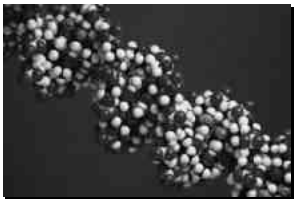
Living in a state of constant ambiguity...

Network Testing Overview

- ☑ **Design is fundamental!**
- ☑ **Business Rule Algorithms**
- ☑ **Behavioral modeling**
- ☑ **Attributes - The key for testing**

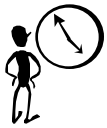


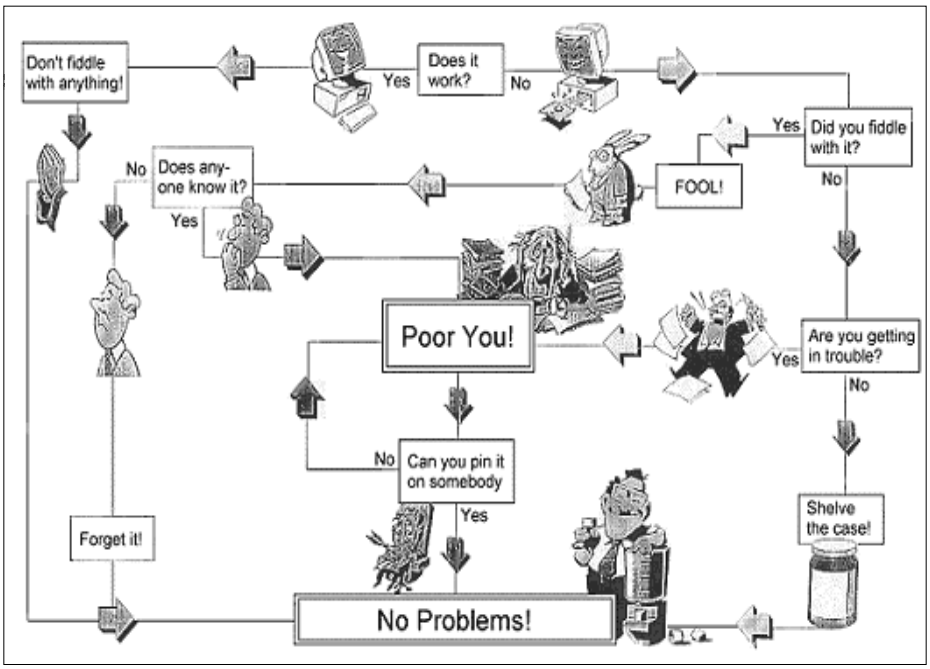




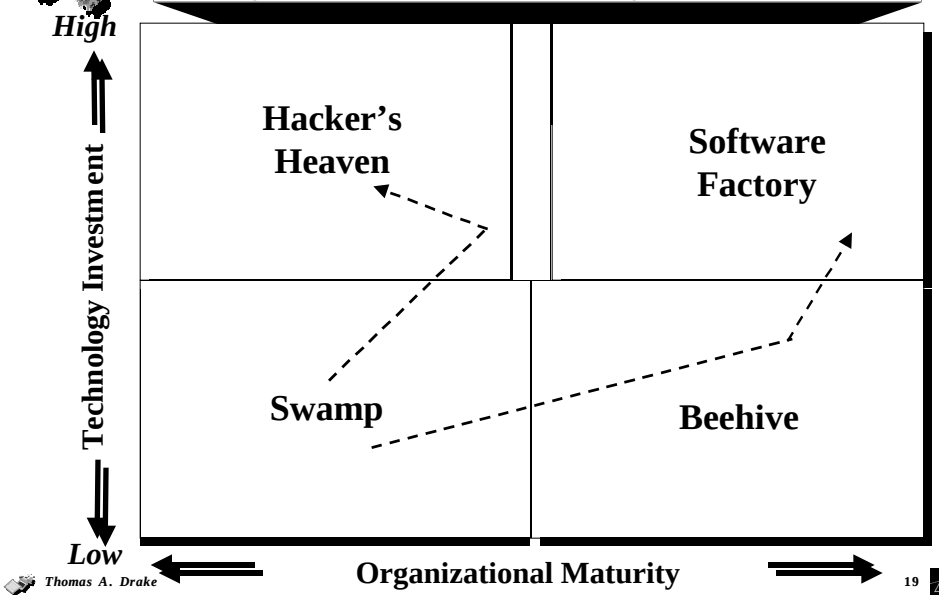
Legacy Issues in Development

- ☑ **Issue of market cycle time and shelf life**
- ☑ **Problem of code entropy**
- ☑ **Impact of incremental patching and upgrading**
- ☑ **Doing it right the first time - interface design is everything**
- ☑ **Simple elegant solutions - stand the test of time and the marketplace**





The Software Development Challenge: "Easy to Hack Out - Easy to Hack In..."



Testing and QA

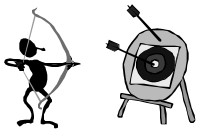
- ☑ **Get the bugs out!**
- ☑ **Indispensable for high quality development/delivery**
- ☑ **Usability labs/focus testing**
- ☑ **In house testing - unit, component, white and black box with gray, integration, system, *compatibility*, chipsets, functional, automated, regression, ad hoc, *dirty* testing (break the software), stress and load testing (resource constraints/bugs and memory leaks)**
- ☑ **Deriving test "cases" (read complex coverage scenarios) from the scripting engine - usually created by the testers**

Functional vs. Compatibility Testing

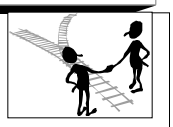
- ☑ **Functionality testing - Process of systematically testing all aspects of a software program**
  - ✓ **Verify program/software system performs the way it was intended by its designers**
  - ✓ **The purpose of functionality testing is to help develop the most stable program possible**
- ☑ **Compatibility or configuration testing - Process of running a stable software program on a variety of hardware components**
  - ✓ **Produce a profile of the program's expected "compatibility" in the real-world of end users/customers**
  - ✓ **Observe how a functionally stable program performs when integrated with the products of various hardware/network vendors and manufacturers (including your own!)**

The Critical Role of QA

- ☑ **The primary objective of QA is in really making sure that the right “mix” of testing talent is available and that the right amount of quality is present**
- ☑ **They are also the binary go, “no go” gates for that all important release quality**
- ☑ **They also ensure that the vital information and feedback loops are running at or near full bandwidth on the project and are constantly pulsing the project in order to make sure it meets its goals.**



The Importance of Engineering Process and Practice Discipline (1)



- ☑ **Programming Standards/Coding Standards -**
- ☑ **Peer Reviews**
  - ✓ **Permit the removal of defects from the software product early and more efficiently.**
  - ✓ **Better understanding of the game product elements and the defects that might actually be prevented**
  - ✓ **The discovery of problems and potential problems before they are found in the testing stage**
  - ✓ **All the participants in a peer review process learn to avoid both common and subtle errors in their own work**
  - ✓ **Peer reviews are finding important problems that have traditionally defied other debugging and testing methods**

The Importance of Engineering Process and Practice Discipline (2)

- ☑ **Process working groups -**
  - ✓ **Concept**
  - ✓ **Requirements definition**
  - ✓ **Design**
  - ✓ **Interface and implementation**
  - ✓ **Maintenance/support**

- ☑ **All of these processes have feedback mechanisms designed for continual improvement**
- ☑ **Software testing plans are initially created in the design phase of the process, completed in the implementation phase**
  - ✓ **High quality software test plans describe both the methodology and the elements that make up the actual tests for the software based on the requirements**

The Importance of Engineering Process and Practice Discipline (3)

☑ **High Quality Network Testing Techniques - The Heart of It All!**

- ✓ **Transaction Based Testing**
- ✓ **Event Triggers**
- ✓ **End to End**
- ✓ **Functional "Thread" Testing**
- ✓ **"Flow"**
- ✓ **Interface Testing**
- ✓ **Domain requirements**

The Real Business of Business!

☑ **One thing in common - Transformation!**

☑ **Leveraging of information, period!**

☑ **Information is the center of the enterprise**

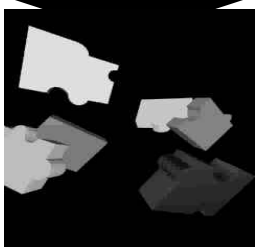
☑ **Building business around information & people**

☑ **The "Crown Jewels" of who we are (intellectual capital)**

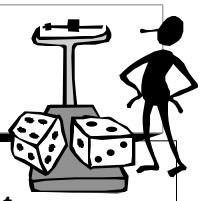
☑ **Questions -**

- ✓ **What is business transformation?**
- ✓ **Why is the customer critical to success?**
- ✓ **How does this accommodate change?**
- ✓ **And what about "good enough" business systems??**

The Real Quality Challenge...



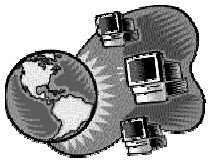
Measuring Quality



- ☑ **Software is fundamentally chaotic by nature**
  - ✓ **Complexity can quickly exceed what we can predict**
  - ✓ **Difficult to determine let alone predict what it will do next!**
- ☑ **There remains a lot of variation in software - So reduce the variation! The problem is systemic in nature.**
  - ✓ **Given a known environment with known inputs software CAN operate in a precise and well understood manner by reducing the inherent instabilities!**
- ☑ **Oh Really! So what must we "add" to software?**
  - ✓ **Identify the functional "flow" abstraction layers of a program! Reqs/Specs**
  - ✓ **Determine the "state condition" information in order to reduce the uncontrolled distribution of information and data**

Out of the Box

- ☑ **A Raising of the Bar - How high can you jump? Far more than just "getting" the next release out. "Unintended consequences..."**
- ☑ **Value high-quality, usable, reliable IT systems and software from the beginning!**
- ☑ **So....**
  - ✓ **Abstraction - "flow" tasking**
  - ✓ **Analysis - problem space**
  - ✓ **Design - system space**
  - ✓ **User interfacing and prototyping - an "outside in" approach**



Best Practices - The Use of Domain/Design Patterns (1)

- ☑ **Set of objects with certain known roles and responsibilities**
  - ✓ **Relationship to each other**
  - ✓ **Common usage**
  - ✓ **Prerequisites**
  - ✓ **Cataloged/documented**
  - ✓ **Refinement/updates/extensibility**
- ☑ **Emerging due to Internet time/intense schedules**
- ☑ **Program structures are fundamental**
  - ✓ **Where are the execution cycles**
  - ✓ **"The most efficient instruction set is the one that's never executed!"**
  - ✓ **Provides for the abstraction that provides summary/overview**

Best Practices - The Use of Domain/Design Patterns (2)

- ☑ **Patterns should also identify the intended use audience**
  - ✓ **Provide for the external and internal assumptions**
- ☑ **Document/Document/Document! (1)**
  - ✓ **Name, problem, context, constraints, trade-offs, static relationships and dynamic rules/behavior, variants/specializations**
  - ✓ **Examples - sample implementations**
  - ✓ **Known uses - describes known occurrences of the pattern and its relationship and application within existing systems**
- ☑ **Generalize! - How can the program be developed such that it minimizes the code interdependencies among the various subsystems?**

(1) Source: Patterns and Software: Essential Concepts and Terminology - Brad Appleton (www.enteract.com/~bradapp/docs/patterns-intro.html)

The Future Frontier?

- ☑ **Net time is test time!**
- ☑ **Customer are approaching zero tolerance for poor performing web sites due to bugs, performance and use "quirks"**
- ☑ **Front-end applications and back-end service providers**
- ☑ **Find and fix network problems no matter how trivial they are**
- ☑ **What is at stake? What will it really take?**

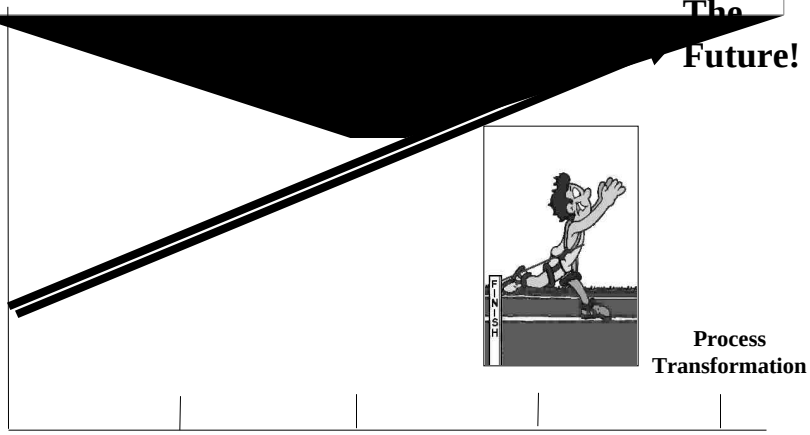
Network Testing Strategy (1)

- ☑ **Risk - manage and mitigate**
- ☑ **Client security**
- ☑ **Customer security**
- ☑ **CM and defect tracking**
- ☑ **Link verification and continuity**
  - ✓ **End to end - top to bottom - depth and breadth**
- ☑ **Web code testing**
- ☑ **Usability testing**

Network Testing Strategy (2)

- ☑ **Performance/Load Testing**
- ☑ **State Testing (California is different!)**
  - ✓ **Event triggering**
- ☑ **Platform configuration**
- ☑ **Business rules/domain logic testing**
- ☑ **System and end-user testing**
- ☑ **Continuous continuity testing**

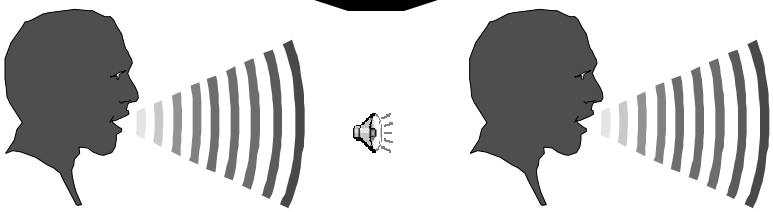
IT Paradigm Shift
Where are You on the Line?



Some Final Thoughts About Network Based Software Systems

- ☑ **Dealing with multiple levels of complexity**
  - ✓ **Testing enemy #1**
- ☑ **Nth Order Computing (modeled after cell life)**
  - ✓ **Computer is 1st order**
  - ✓ **Objects/components are 2nd order**
  - ✓ **Interaction of objects is 3rd order**
  - ✓ **Result/outcome/state transition is 4th order**
  - ✓ **Nth Order is shifting function onto structure and code onto data**
- ☑ **Bandwidth, Turnaround Time, and Complexity**

Questions & Answers



QW2000 Vendor Technical Paper 8V2

Mr. Mark Myers
(Teradyne SST)

A Model-Based Technique for Test Program Creation

BACK TO QW2000 PROGRAM

Key Points

- Test programs for most modern software-based systems are still generated manually.
- Automating software test generation significantly increases test quality and reduces life cycle costs.
- Using a model-based approach, TestMaster not only automates test generation, but makes it easy to respond quickly to changes in requirements, system configuration, or test execution environment.

Presentation Abstract

Software verification has become a bottleneck in the delivery of new communications products. Creating test content is the biggest part of the problem. Organizations that adopt an automated approach to test content creation will have a significant competitive advantage in the race to cover the glove with reliable communications systems. TestMaster offers a new approach to creating test content that combines a graphical specification, or model, of system-under-test behavior with an automatic test script generator. TestMaster customers report 80-90% improvements in test development time and productivity across a wide range of software applications and test execution environments.

About the Author

Mark A. Myers is product manager for in-circuit board test systems and for the BoardWatch Test Process Management System at Teradyne Inc. He attended the University of Michigan, where he earned his B.S.E.E., M.S.E., and M.B.A degrees. He joined Teradyne in 1977 as a sales engineer, and managed the company's Detroit sales office before assuming his current position.

BACK TO QW2000 PROGRAM

TestMaster:
Model Based Test Generation for Software Systems

Teradyne Software & Systems Test
www.teradyne.com/sst

©1998 Teradyne Software and System Test
Teradyne Confidential

The Compounding Effect of Defects

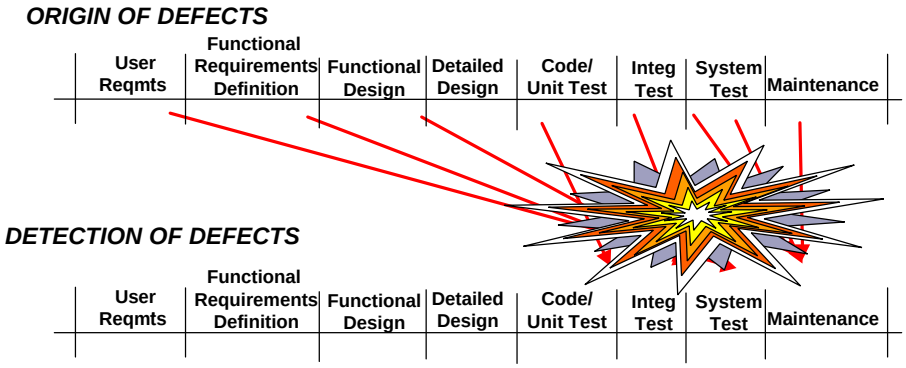
Requirements	Functional	Design	Code
Correctly defined requirements	Correctly defined functional description	Correctly designed specifications	Correctly coded software
Incorrectly defined requirements: missing, misunderstood, ignored outdated, unneeded, unspoken (assumed)	Erroneous functional specs based on incorrectly defined requirements	Design errors based on incorrectly defined requirements	Coding errors based on incorrectly defined requirements
	Incorrectly defined functional specs	Design errors based on incorrectly defined functional specs	Coding errors based on incorrectly defined functional specs
		Design errors	Coding errors based on design errors
			Coding errors

Even good work on a faulty base will result in a defective product

Source: A. Davis. Software Requirements; J.Taft, Nationsbank

©1998 Teradyne Software and System Test
Teradyne Confidential

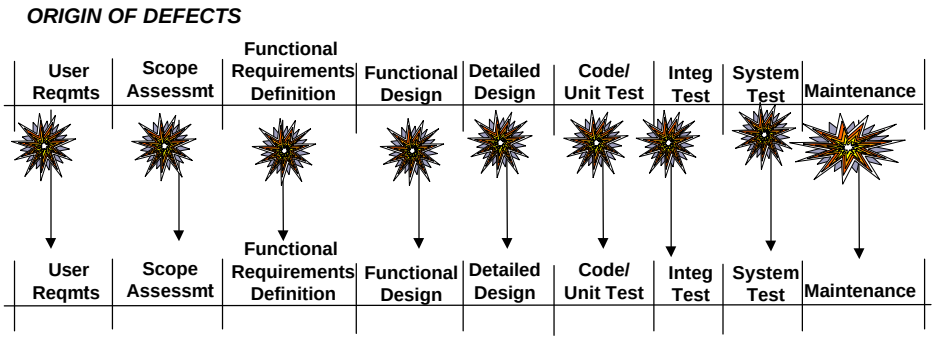
Defects Introduced In Early Stages Are Not Found Until Very Late In the Process



a.k.a. The Big Bang Approach to Quality

©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

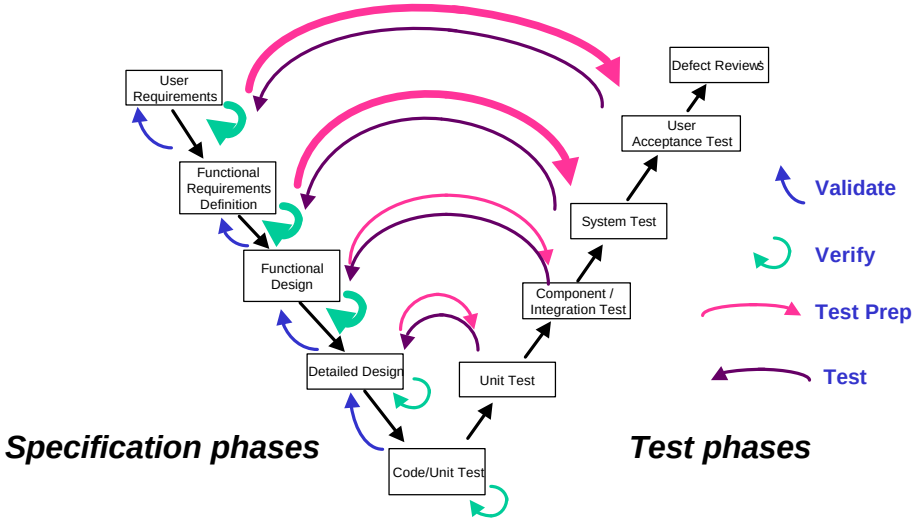
Model Based Testing Enables Incremental Testing to Detects Defects Within the Phase of Origin



Phase Containment - finding and fixing errors within the phase of origin

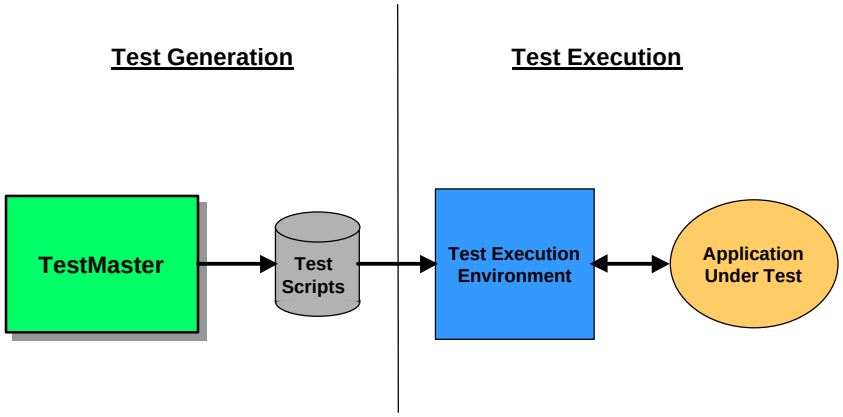
©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

TestMaster Improves V Model by Improving Up Front Validation and Leveraging Validation Effort for Test Development



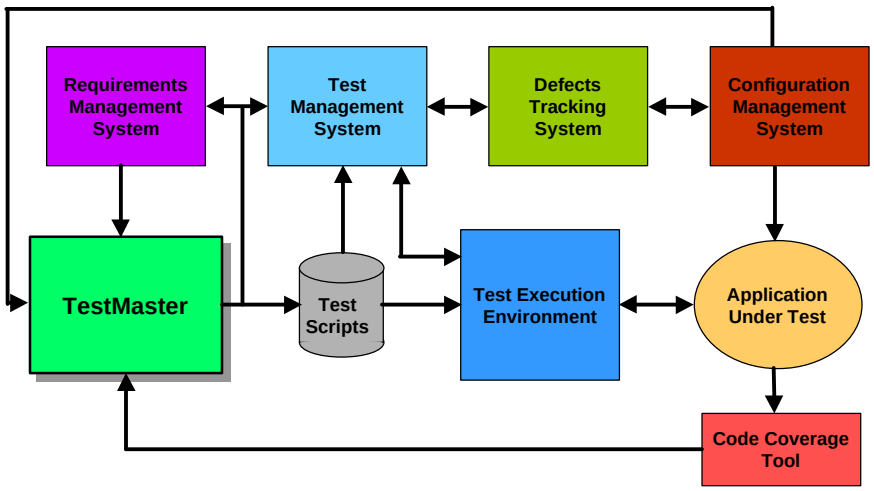
©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

TestMaster Automatically Generates Tests Programs for Test Execution Environments



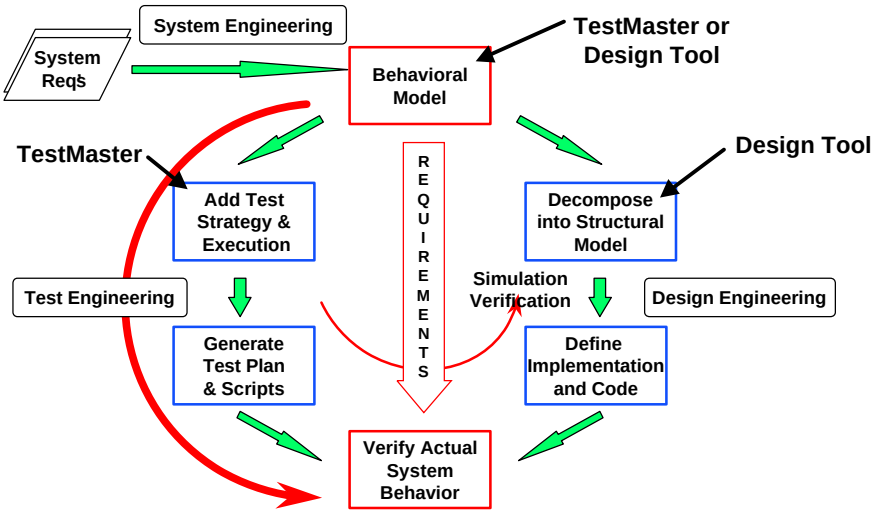
©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

TestMaster Complements Other Test Automation Tools



©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

TestMaster Complements Model-Based Design Tools



©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

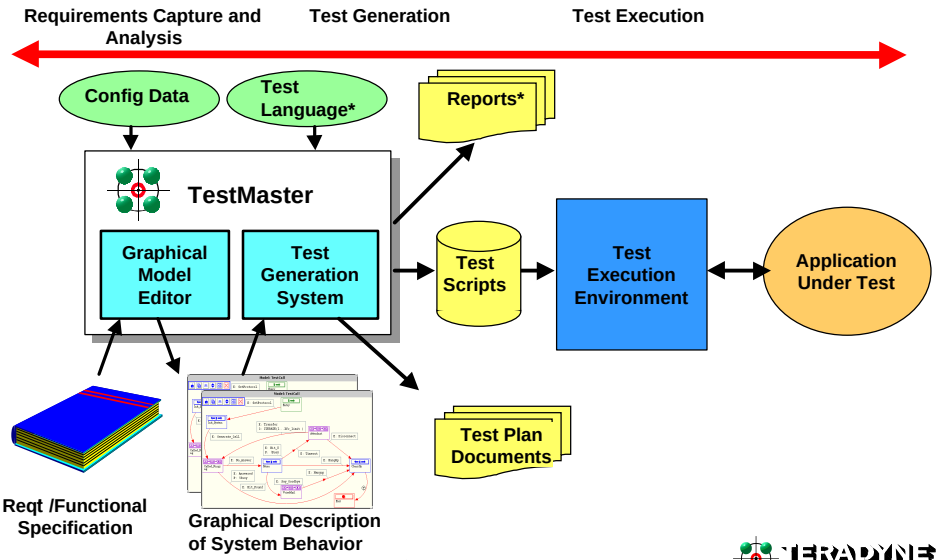
TestMaster's Unique Capabilities Deliver these Major Benefits

Feature	Benefit
Known & Controllable Fault Coverage	High Quality Product
Concurrent Design & Test Automatic Test Generation	Fast Time-to-Market
High Engineering Re-Use	Low Test Development Cost

©1998 Teradyne Software and System Test

TERADYNE Software & Systems Test

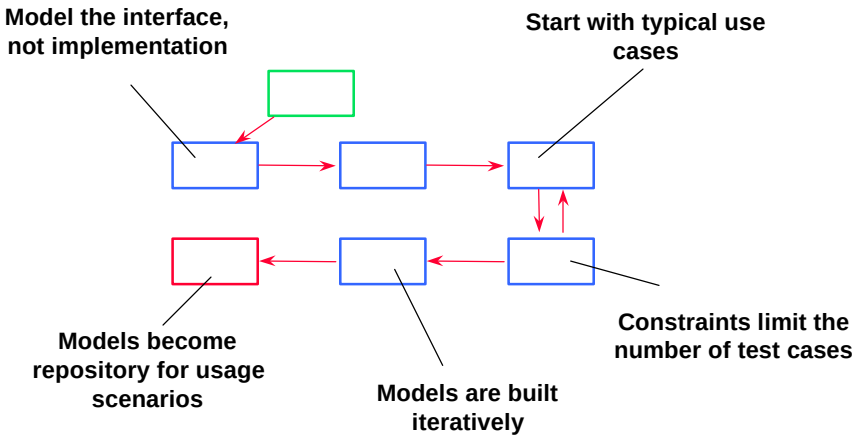
TestMaster Integrates Requirements Definition Through Test Execution



©1998 Teradyne Software and System Test
Teradyne Confidential

TERADYNE Software & Systems Test

TestMaster Models Behavior at One or More of the System Interfaces



©1998 Teradyne Software and System Test

TERADYNE Software & Systems Test

TestMaster is Ideally Suited for Integration & System Test

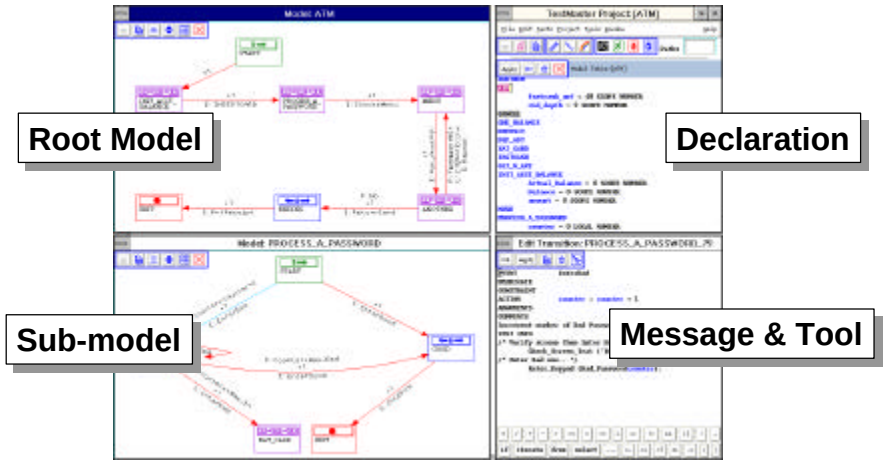
System Executable	Behavior Modeled	Test Phase
External Interfaces: User Interface; Communication Interface; Hardware Interface	GUI / Call Processing	System
Subsystems & Features: Print/ Report; Analysis; Routing; Control	Protocols / Classes / Messages&Data	Integration
Services/Framework	API / Language	Unit

©1998 Teradyne Software and System Test

TERADYNE Software & Systems Test

The TestMaster User Interface Consists of These Primary Windows

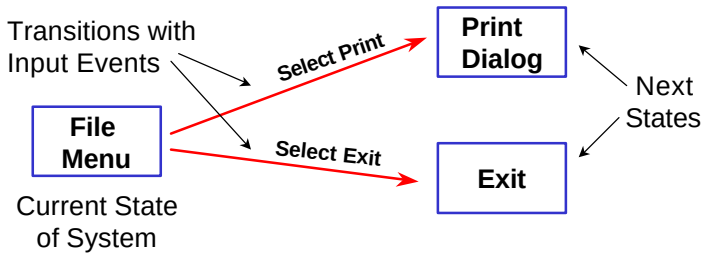
Example: Automatic Teller Machine



©1998 Teradyne Software and System Test

TERADYNE Software & Systems Test

Models are Composed of States and Transitions

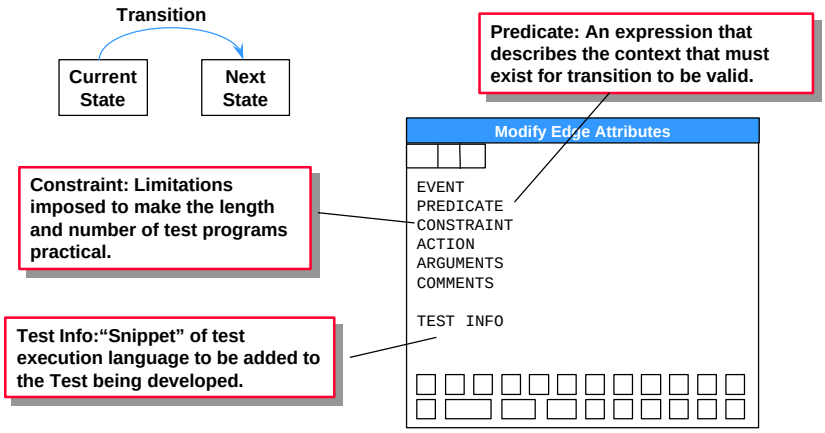


- **All Transitions contain an INPUT EVENT and a NEXT STATE**
- **Test Generation is accomplished by traversing from STATE to STATE via transitions**
- **A Transition can also: define probability, outputs, manipulate variables, describe test actions**

©1998 Teradyne Software and System Test

TERADYNE Software & Systems Test

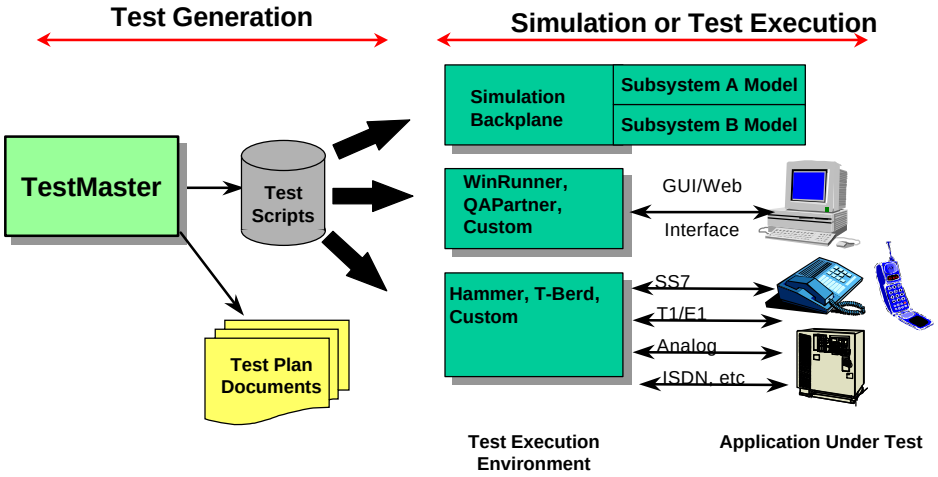
TestMaster Models Are Extended Finite State Machines Where Transitions Contain Most of the Data



©1998 Teradyne Software and System Test

TERADYNE Software & Systems Test

TestMaster Generates Test for Any Execution or Simulation Environment

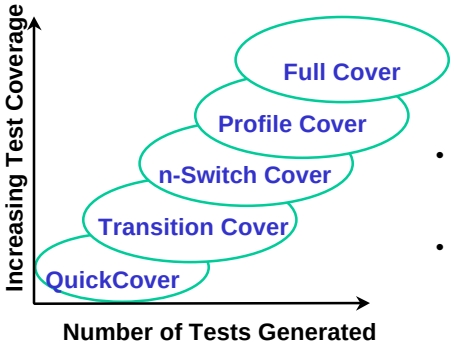


©1998 Teradyne Software and System Test

Teradyne Confidential

TERADYNE Software & Systems Test

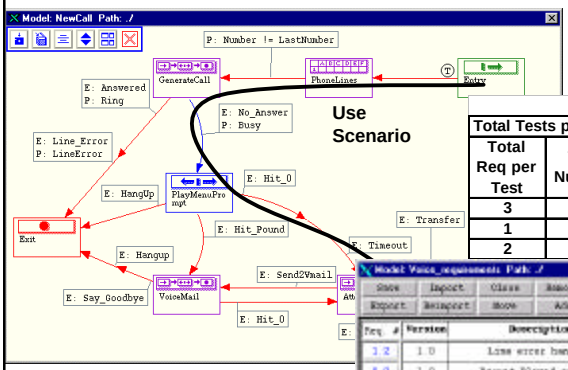
A Single Model can Create Multiple Test Suites



- **Once Captured a Model is a Source of Tests for All Aspects of the Testing Process**
  - Specific Requirements or Modules
  - Negative Behavior
  - Full Regression Tests
- **Test Suites are Easily Rebuilt For Saved Configurations**
- **Tests are Generated Independent of Test Language**

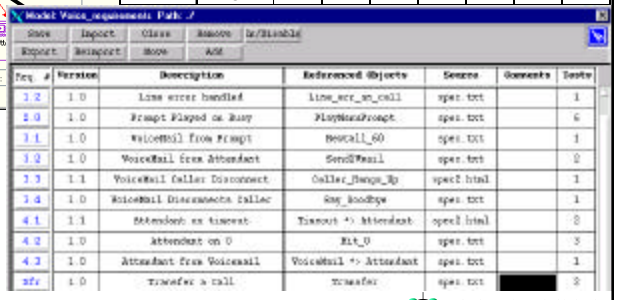
©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

Each Path Defines a Test / Use Scenario and is Traced Back to the Requirements



Use Scenario

		Requirement ID						
Total Tests per Req		2	3	2	3	3	0	1
Total Req per Test	Test Number	1.0	1.0.1	1.0.2	1.1	1.1.1	1.2	1.2.1
3	1		X			X		X
1	2	X						
2	3		X		X			

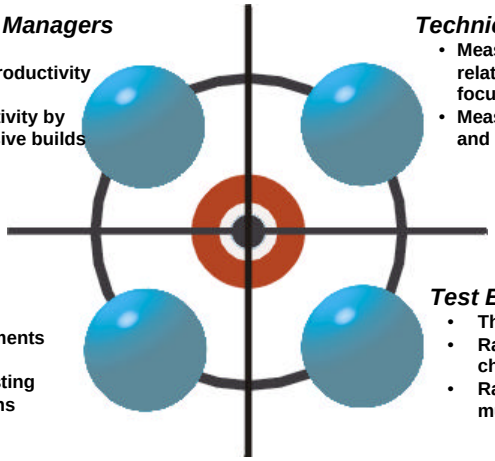


©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

TestMaster Provides a Comprehensive Test Solution for All Members of a Project Team

Program/Business Managers
- Proven solution
- Improve initial test productivity by 2x- 10x
- Improve test productivity by 10x - 20x for successive builds

Technical Managers
- Measures test coverage relative to requirements to focus test investment
- Measures test progress and completeness



System Engineers
- Eliminates requirements ambiguity
- Enables explicit testing for illegal conditions

Test Engineers
- Thorough test coverage
- Rapid response to changes
- Rapid test generation for multiple configurations

©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

TestMaster is Deployed in a Wide Range of Telecom Companies:

Lucent	**Ericsson**
Nortel	**AT&T**
Motorola	**Bellcore (SAIC)**

… With Proven Results:

Using TestMaster to automate our test generation can increase our productivity by over 80%

Jim Clarke
Lucent Technologies

TestMaster takes the guess-work out of defining test cases … it has paid for itself many times over with the problems it has found in our software

Neil Anderson
Ericsson Mobile Phones

The beauty of the TestMaster-Hammer approach is that as our software changes we can keep thousands of regression tests in sync with just a small change to the TestMaster model --- that saves us months of rework for each update

Peggy Champlin
Nortel

©1998 Teradyne Software and System Test
Teradyne Confidential
TERADYNE Software & Systems Test

QW2000 Paper 8T2

Mr. W. T. Tsai & Mr. Xiaoying Bai & Mr. Baisu Huang, George Devaraj, Ray Paul
(Arizona State University)

Automatic Test Case Generation for GUI Navigation

BACK TO QW2000 PROGRAM

Presentation Abstract

GUI has been pervasive today and it has been used in safety-critical application where failures of the application can have serious consequences. Unfortunately, GUI testing is expensive and currently many of steps are manual. Currently, most GUI testing techniques focus on capture-and-replay mechanism. Even though it is a powerful technique and has been used testing numerous GUI testing projects, the initially set of test cases must be developed, possibly manually. Furthermore, it is difficult to state the coverage of GUI by using the capture-and-replay mechanism because the coverage is provided with respect to the existing test cases, rather than with respect to other criteria such as navigation.

This paper presents a technique to develop the initial set of GUI test cases. In this approach, test cases can be generated systematically thus ensuring test coverage. This approach divides testing GUI into two parts:

- Screen navigation: this part ensures that each screen will be visited, as well as all the paths that might be used to traversed from one screen to another;
- Widget logic: this part ensures that the software behind a widget will respond correctly when the corresponding widget is selected or activated by the application.

By separating GUI testing into these two parts, the testing effort and complexity can be significantly reduced. This paper will illustrate the saving that can be achieved using this approach. Since widget logic is application dependent, this paper then focuses on screen navigation.

This paper propose a model called SNet (for Screen Net) to model screen navigation. It is a directed graph model with a node represents a screen, and a link between two nodes a navigation path from one screen to another. The conditions, precondition and post-conditions, for traversing from one screen to another are also specified on the link. Each navigation step (NStep) is defined by 5 components: the departure screen, the arrival screen, the navigation event, the precondition and the post-condition. Pre-conditions and post-conditions are specified by the states of the screens, which are further specified by the combination of the values of the GUI widgets inside the screen container.

The proposed model SNet has been automated with friendly user interface. A user can use the tool to

interactively input the navigation information, and the tool will check the completeness and consistency of the input. Once the SNet has been specified, the tool can systematically generate all kinds of test cases based on various navigation paths and scenarios based on certain coverage criteria such as path coverage or node coverage. The test cases are written in a high-level description language. An example is shown below:

The tool has been constructed and used to test a variety of GUI applications successfully.

About the Authors

W. T. Tsai is currently Professor of Computer Science and Engineering at Arizona State University. He received his Ph.D. and M.S. in Computer Science from University of California at Berkeley. He has been involved in software engineering research.

Xiaoying Bai is currently a graduate student in the Department of Computer Science and Engineering at Arizona State University. She received her M.S. in Computer Science and Engineering from University of Minnesota, Minneapolis, MN 55455, 1999.

Baisu Huang is currently a Research Associate in the Department of Computer Science and Engineering at Arizona State University. She was Associate Professor at Peking University, Beijing, China. She received her Ph.D. from Northwestern Polytechnical University in China.

George Devaraj is current a graduate student in the Department of Computer Science and Engineering at University of Minnestoa, Minneapolis, MN 55455.

Ray Paul is current the Directorate at DoD OASD Y2K Office. He recently received his Ph.D. in Computer Science and Engineering.

BACK TO QW2000 PROGRAM

Test Case Generation for GUI Navigation

Wei-Tek Tsai[*+], **Xiaoying Bai**[*], **Baisu Huang**[*], **George Devaraj**[+], **Ray Paul**[-]

*Arizona State University
+University of Minnesota
- OASD Y2K Office

Outline

- Introduction
- SNet: the Model for GUI Navigation Behavior
- Test Case Generation Based on SNet
- Tool and Experiments
- Conclusion

Introduction: The Problem

- Main Problem
  - GUI is pervasive but GUI testing is difficult
    - Variety of GUI components
    - Complex GUI construction
    - Complex user interaction
  - GUI testing is expensive and many steps are done manually
  - Test case generation is an issue

Introduction: Current Approach

- Capture and Replay
  - The principal technique used in current GUI testing tools
  - Capture and store user's interactions during a test session, which can be repeated and reused
  - Initial set of test cases must be developed manually
  - Hard to evaluate the coverage of the test

Introduction: Our Approach

- Two levels of GUI testing
  - Screen navigation
    - Take the top-level container as the basic functional module and test the inter-module GUI behavior
    - Ensure screen and navigation path coverage
  - Widget logic
    - Test the correctness of the behavior of the widgets inside a functional module - the top-level container
- Advantage
  - Testing effort and complexity can be significantly reduced

5

3/31/00

Introduction: Screen Navigation

- Problem
  - A test case is a navigation through many screens. In addition to generate test data for each navigation step between two screens, we also need to identify the sequence of execution steps.
  - The pre-condition for a navigation from one screen to another is hard to define
- Solution presented in this paper
  - Model the navigation behavior by a directed graph SNet (for Screen Net)
  - Three-level test case generation based on SNet model
  - Tool support for user interactive SNet specification and automatic test case generation

6

3/31/00

SNet: Screen Network

- Screen Navigation Model
  - A directed graph to model the navigation behavior through all the screens in a GUI system
- Defined by:
  - A set of screen nodes
  - A set of directed navigation links, or navigation steps

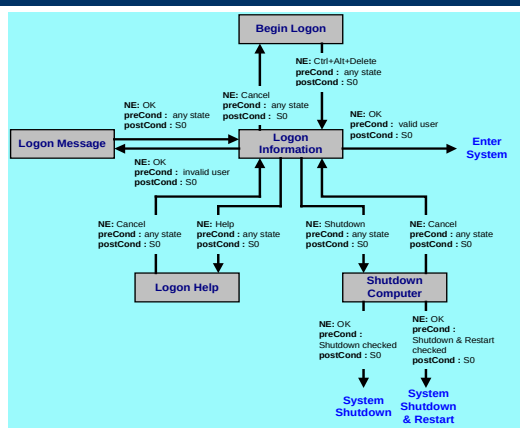
SNet: Screen Nodes

- Represent the top-level containers in the GUI system
- Specified by:
  - ID
  - A set of widgets
  - A set of states
  - A set of navigation events
- States are defined by the values of the GUI widgets that are contained in the screen and that accept user inputs

SNet: Navigation Steps (NStep)

- Directed Navigation Links:
  - Represent the navigation logic with pre and post condition information attached
- Specified by:
  - ID
  - Departure Screen
  - Arrival Screen
  - Pre Condition
  - Post Condition
  - Navigation Event
- Conditions are defined by the states of the screen. A condition is a logical expression (“and”, “or”, “not”) on a set of screen states.

An Example of SNet



Name

The screen of the GUI system

NE: event
preCond : prec
postCond : postc

A navigation step:

- **NE**: navigation event, the event that triggers the navigation
- **preCond**: the precondition of the navigation
- **postCond**: the post conditon of the navigation

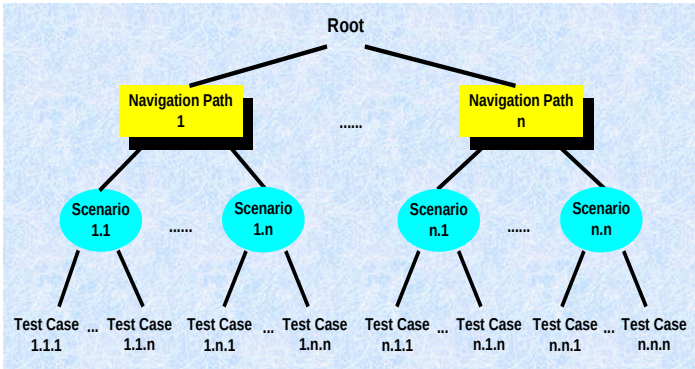
SNet Completeness and Consistency Checking

- Checking Screens
  - Ensure that all screens are represented in the SNet model and that each node in SNet uniquely represents a screen
  - Ensure that there is no conflicting information in screens
- Checking NSteps
  - Ensure that there is no conflict in the navigation conditions
  - Ensure that for each screen, each combination of pre-condition and navigation event uniquely decides the target screen
  - All screens are connected by navigation links (NSteps)

Test Case Generation by SNet

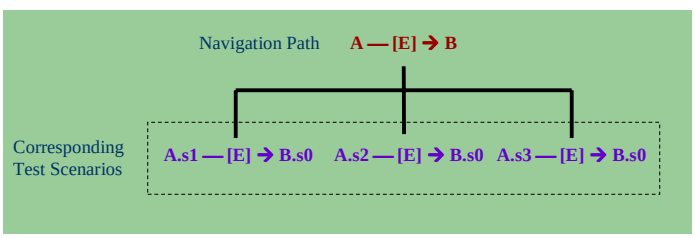
- Three levels of Test Case Generation
  - Navigation Paths: Navigation sequence between screens in terms of screens, navigation conditions and links.
  - Test Scenarios: Navigation paths specified by screen states rather than by conditions
  - Test Cases: Specification of the actual test data and testing activities

Tree Structure of the 3-Level Test Case Generation



An Example of Test Scenarios Derived from A Navigation Path

Consider two screens A and B. Under any state, A can be navigated to the default state of B on the occurrence of event E. Suppose A has 3 states: s1, s2, s3 and the default state of B is s0. The 3 scenarios derived from the navigation path A → B are illustrated in following figure:

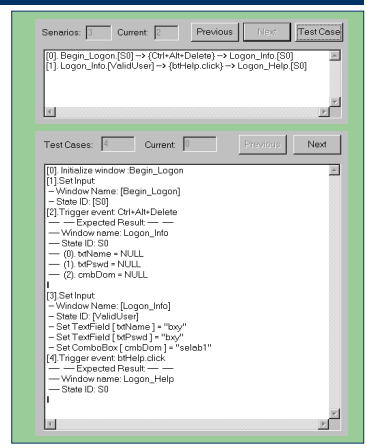


An Example of Test Case Derived from A Test Scenario

Take 3 screens of the Logon System in the example: “Begin_Logon”, “Logon_Info”, and “Logon_Help”. And take the navigation path: from “Begin_Logon” to “Logon_Info” to “Logon_Help”

Take the following Test Scenario:
• Triggered by the event “Ctrl+Alt+Delete”, screen Begin_Logon in state S0 is navigated to screen Logon_Info is state S0
• Triggered by the event “Button Help Click”, screen Logon_Info in state “Valid User” is navigated to screen Logon_Help in state S0

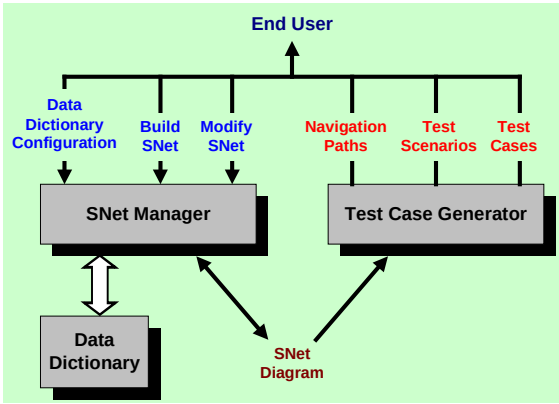
Suppose one of the 4 valid users for the logon system is: “Name = bxy, Password = bxy, Domain = selab1”



Tool Prototype

- A prototype tool is constructed to facilitate the user-interactive SNet specification process and to automate the test case generation process
- Three parts of the prototype system
  - *Data Dictionary* : Store the general information of the GUI system – the screens, their widgets and navigation events
  - *SNet Manager* : Provide an interface for user to interactively build, store and maintain the navigation diagram
  - *Test Case Generator* : Generate the navigation paths, test scenarios and test cases in a high level description language

Prototype System



Experiments

- Test cases generated by the tool for two example GUI system

Example System	Screens	Navigations	Test Scenarios	Test Cases
Logon GUI	5	11	25	71
SATM GUI	12	21	27	105

- Comparison of the time to generate the same number of test cases shown in the table above for two approaches

Example System	Automatic Generation Supported by Tool	Manual Generation
Logon GUI	20 minutes	710 minutes
SATM GUI	35 minutes	1050 minutes

* Assuming it takes average 10 minutes to develop each test case manually

Experiments

- Experiments on small systems shows that SNet is helpful for
  - Understanding the GUI system
  - Reducing testing effort and cost
  - Improving the reusability and manageability of test cases as well as the generation process
  - Test coverage analysis with respect screen nodes, navigation path and screen states

Conclusion

- This approach significantly reduce complexity and testing effort by separating GUI testing into two parts
- SNet provides a formal model to specify the GUI navigation behavior
- SNet establishes the basis for systematic test cases generation, ensuring coverage and reliability
- Multilevel test case generation approach improves the reusability of test cases at each level, facilitates test case maintenance and change management

Test Case Generation for GUI Navigation

W. T. Tsai*+, Xiaoying Bai*, Baisu Huang*, George Devaraj+

***Department of Computer Science and Engineering**
Arizona State University
Tempe, AZ 85259

+Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455

Ray Paul
OASD Y2K Office
Washington, D.C.

Abstract

GUI is an important feature enabling human-computer interaction to be more user friendly. It has been used in safety-critical applications where failure of the application can have serious consequences. Unfortunately, GUI testing is expensive and currently many of the steps are manual. This paper proposes a model called SNet (for Screen Network) to model screen navigation and presents a technique to develop the initial set of GUI navigation test cases based on various navigation paths and test scenarios. The model also allows coverage analysis such as path coverage. In this way, test cases can be generated systematically while at the same time ensuring coverage. This paper then presents a prototype tool, which can help user to interactively input the navigation information, check the completeness and consistency of the input, and automatically generate test cases.

1 Introduction

GUI (Graphical User interface) is a common feature in most modern software systems. It is widely used as the front-end control sub-system in many applications including safety-critical systems. GUI is where the end-users look, feel and interact with the underlying application system. Due to the increasing importance of a GUI system, GUI testing is gaining attention. But unfortunately, because of the variety of GUI components as well as the complexity of GUI construction and user interaction, testing a GUI is difficult. Currently, GUI testing is expensive and many of steps are done manually.

Many tools are available today to facilitate GUI testing, such as QA Parterner [QA], JavaStar [Star], XRunner [XR], TestMaster [TM] and so on. The basic technique used in these tools is capture-and-replay. Capture-and-replay tools can capture the user's interactions on the GUI objects, which are under test during a test session, store them in a test script which can be interpreted and executed by the tool. Once captured and stored, testing can be repeated at any time, thus addressing the reusability problem. Even though capture-and-replay is a powerful technique and has been used for testing numerous GUI projects, the initial set of test cases must be developed. All this has to be

possibly done manually. Furthermore, it is difficult to state the coverage of the GUI test by using the capture-and-replay mechanism because the coverage is provided with respect to the existing test cases, rather than with respect to other criteria such as navigation.

Test case generation in GUI testing is an issue. This paper presents a technique to automatic generate the initial set of GUI test cases based on a GUI behavior model. In this approach, test cases can be generated systematically thus ensuring proper test coverage. This approach divides testing GUI into two parts:

- *Screen navigation*: take the top-level container as the basic functional module and test the inter-module GUI behavior. This part ensures that each screen will be visited, as well as all the paths that might be used to traversed from one screen to another

- *Widget logic*: test the correctness of the behavior of the widgets inside a functional module - the top-level container. This part ensures that the software behind a widget will respond correctly when the corresponding widget is selected or activated by the application

By separating GUI testing into these two parts, the testing effort and complexity can be significantly reduced. This paper illustrates the savings in cost and effort that can be achieved using this approach. Since widget logic is application dependent, this paper focuses on screen navigation.

Top-level containers, called screens are the building blocks of a GUI system. As an interactive system, each function in a GUI system is performed by navigation through a set of screens. A screen can take input from users, produce output to display, and consists of all the underlying logic to process user requests based on the user's inputs. Navigation through screens describes the high-level functionality of the system and the inter-relationship among screens.

For large applications, navigation through screens can be quite complex. Each Screen in the GUI system can be in several states depending on the data, which it contains in its GUI widgets. There is a set of navigation events, which would enable transitions between screens. Pre-condition is the current status of the screen when a particular event is triggered. Post-condition is the expected status of the target screen after it becomes the focus of the GUI system. Each screen can be navigated to different targets under different combinations of pre-conditions and the triggering events. The system may also navigate to a particular screen but with different post conditions.

For example, consider a GUI system with 10 screens. Assume each screen has 3 states and 2 navigation events. Then, at least 60 ($3 \times 2 \times 10 = 60$) test cases are needed. With the increase in complexity of the GUI system, the number of test cases required to test the states grows exponentially.

Therefore, automating the process of test case generation for GUI navigation can significantly save the time and effort involved in testing. This paper presents an approach for automatic test case generation based a model specification of the GUI navigation behavior.

Extensive research has been done on test cases generation [Clarke 1984, Boehm 1984, Tsai 1990, Weber 1994, Paradkar 1996, Yin 1997]. [Clarke 1984] describes a test case generation method based on symbolic execution technique. [Tsai 1990] discusses the automatic test case generation technique

for relational algebra specifications expressed as queries. [Yin 1997] presents an automatic test generation scheme for black box testing, using checkpoint encoding and anti-random testing schemes. One particular problem for GUI navigation testing is that, in addition to generate the test data for each input, we also need to generate the sequence of execution steps.

This paper proposes a model called SNet (for Screen Network) to model screen navigation. It is a directed graph model where a node represents a screen and a link between two nodes represents a navigation path from one screen to another. The precondition and post-conditions for traversing from one screen to another are also specified on the link. Each navigation link (NStep) is a six tuplet and is defined by 6 components: the ID for the navigation step, the departure screen, the arrival screen, the navigation event, the precondition and the post-condition. Conditions are specified in the states of the screens, which are further specified by the combinations of the values of the input GUI widgets inside the screen container.

Based on the model, test cases can be generated systematically from various navigation paths and scenarios based on certain coverage criteria such as path coverage. The navigation paths, test scenarios and test cases construct the three levels of the test elements. They are organized into a tree structure, which can be maintained and reused at different levels.

A prototype tool is developed to help user interactively input the navigation information. It will check the completeness and consistency of the input. Once a SNet has been specified and checked, the tool can then generate all kinds of test cases in a description language.

The paper is organized in the following manner:

- Section 2 briefly introduces GUI components and GUI system structure
- Section 3 presents the navigation model *SNet* and the rules for its completeness and consistency checking
- Section 4 discusses the technique for test case generation based on SNet.
- Section 5 presents the prototype tool and some experimental results.

2 GUI Components and System Structure

A GUI system is built from GUI components by containment. For example, the containment hierarchy of swing components is illustrated by the following figure [Swing]:

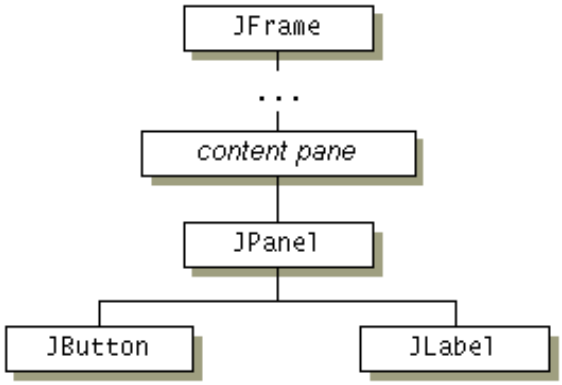


Figure 1. Swing GUI component structure

Viewed in a top-down manner, a GUI system is constructed by following a hierarchical containment structure using containers and widgets.

2.1 GUI container

Containers serve as the carriers of atomic GUI controls. These containers decompose

the complex GUI system into loosely coupled functional modules layer by layer. Containers, such as Windows (VC), Frame (Java), Form (VB), Dialog, can also be divided into two categories:

- The top-level container

  Such containers lie at the outermost level of the containment structure of the GUI system. They separate GUI widgets into loosely coupled groups, representing relatively independent functional modules. GUI components in this category include Windows (VC), Dialog, Applet (Java), Frame (Java), and Form (VB) and so on.

- The intermediate container

  Components in this category such as menubar, toolbar and panel (Java) are embedded within top-level containers and contain atomic GUI components or other intermediate containers. Doing so simplifies the positioning problem of atomic GUI widgets. Such a container plays a weak role in functional decomposition compared to the top-level container.

2.2 Atomic GUI control - GUI widget

In contrast to containers, atomic GUI controls or GUI widgets such as buttons, textfields and checkboxes are the simplest elements inside GUI containers. For example, a logon dialog can be made up of two input text fields for NAME and PASSWORD and two buttons for OK and CANCEL.

GUI widgets are the real carrier of system functionality. They accept user's interactions as events and associate events with specific program logic. For example, a click event on a button A will trigger the procedure called onButtonAClicked ().

3 Screen Navigation Model

Each top-level container is called a screen and is a relatively independent functional module in the GUI system. The current focus of the GUI system shifts from one screen to another by specific events. Such logic is called screen navigation and it reflects the relationship between the functional modules and behavior of the system when the functional modules are integrated. Testing the correctness of the navigation logic is an important issue in GUI system testing.

A model SNet (Screen Network) is proposed to model the navigation behavior though all the screens in a GUI system. It uses a directed graph model where nodes represent screens and directed links represent the navigation logic. Conditions, which include both preconditions and post-conditions of the navigation, are also specified on the link.

The following sections will introduce the definition of screen navigation, navigation step, navigation diagrams, their identification process and C&C checking on the SNet model.

3.1 Screen Navigation

In a GUI system, at each time, a particular active screen is interacting with the user. Triggered by specific events, the current screen may be deactivated and another screen will be loaded in or activated. This is an example of screen navigation, and these two screens are logically connected by the event that caused the navigation.

For example, consider the logon system in the Windows NT workstation 4.0. The first screen is a dialog titled "Begin logon". Triggered by the event "Ctrl+Alt+Delete", the "Begin Logon " disappears and another dialog titled "Logon Information" is loaded in. Then, when the event "click the button Help" is fired, the dialog "Logon Information" is deactivated and another dialog titled "Logon Help" is loaded in and becomes the focus of the system. To specify the relationship between these dialogs, we say that:

- Screen "Begin Logon" is navigated to screen "Logon Information" by event "Ctrl+Alt+Delete"
- Screen " Logon Information " is navigated to screen "Logon Help" by event "Help button click"

3.2 Navigation Step: NStep

Each direct connection between two screens is called a navigation step or NStep. There are usually two kinds of navigation: unconditioned and conditioned. In case of unconditioned navigation, a screen, which is triggered by the same event, always arrives in the same target screen, no matter what state the departure screen is in. For conditioned navigation, a screen, which is triggered by the same event, will arrive at different target screens depending on the status of the departure screen.

For example, consider the “Logon Information” screen in the logon system. No matter what name and password the user put in, whenever the “Help” button is clicked, the “Logon Help” screen appears. This is an example of unconditioned navigation.

For the case of conditioned navigation, consider the "Logon Information" screen. The "OK button click" event will result in different screens under different conditions. In case the user puts in the correct name and password, the interface for Windows NT operating system will appear. Otherwise, a dialog "Logon Message" will appear, instructing the user to re-enter the logon information. This is a typical example of conditioned navigation. In this case, the status of the departure screen or the condition is the key to distinguish navigation steps from each other.

In general, a navigation step is decided by the following elements:

- The unique identification of the navigation step called ID;
- The screen the navigation departs from, called the departure screen DS;
- The screen the navigation arrives at, called the arrival screen AS;
- The pre-condition of the navigation, called preCond;
- The post-condition of the navigation, called postCond; and
- The event that triggers the navigation is a navigation event or NE.

Thus, a navigation step NStep in this model is defined as a 6-tuplet:

$$NStep = (ID, DS, AS, preCond, postCond, NE)$$

Most navigation steps are conditioned, and they need to be specified. Pre-condition is the current status of the screen when a particular event is triggered. Post-condition is the expected status of the target screen after it is activated. Each screen can be in a number of states depending on the values associated with the widgets inside the screen. A navigation condition is defined by a group of states of the screen. For example, in the Windows NT logon example mentioned earlier, a navigation condition for the “Logon Information” screen can be “invalid user”.

This condition can be specified by the following states of the screen:

- An invalid user name within a valid domain; or
- A valid user name, but with an invalid password, within a valid domain;

3.3 Navigation Diagram: SNet

The SNet (for screen network) is a graph model that describes GUI screen navigation information. In a SNet, a node represents a screen and a directional link between two nodes the navigation from the first screen to the next. In other words, a SNet is defined as a set of screens and a set of navigation steps:

$$SNet = (\{Screen\}, \{NStep\})$$

Each screen is defined by the following elements:

- A unique identification, ID
- A set of atomic GUI widgets
- A set of states
- A set of navigation events

$$Screen = (ID, \{Widget\}, \{State\}, \{Events\})$$

The state of the screen is determined by the values of the GUI widgets, which accept user's inputs, inside the screen.

Appendix A gives a visual illustration of the SNet for the example Windows NT Logon GUI system.

3.4 Completeness and Consistency Checking

Software specification is an interactive process, and it is prone to errors [Kirani 1994]. It is also subjected to continuous modifications due to the changes in the requirements or design. This section introduces rules to perform completeness and consistency (C&C) checking of a SNet model to ensure that the model is correct.

C&C checking for SNet involves checking whether each screen and navigation is represented properly in the SNet model. The C&C checking includes two parts:

- C&C check on screens; and
- C&C check on navigation steps.

Checking Screens

Screen C&C checking ensures that:

- All screens are represented in the SNet model and that each node in SNet uniquely represents a screen; and
- There is no conflicting information in each screen.

Checking NStep

NStep represents the navigation logic. C&C checking on NStep ensures that

- There is no conflict in the navigation conditions;
- For each screen, each combination of pre-condition and navigation event uniquely decides the target screen; and
- All the screens are connected by navigation links (NSteps).

These rules have been formalized and can be automatically executed on a SNet model.

4 Test Case Generation in the SNet model

Currently, most test cases are generated manually. A typical GUI application can have numerous scenarios, however, in practice few of these scenarios can be generated and tested. With a limited number of test cases generated, it is difficult to have any assurance of reliability and coverage. This paper introduces a hierarchical approach to generate test cases systematically from the SNet model, and test engineers can use the SNet model to ensure certain test coverage. Furthermore, a prototype tool has been developed that can generate templates for test scripts to test the GUI application, thus significantly reduces the time and effort to generate test cases.

The test case generation takes a 3-level approach:

- *Navigation Paths*: This step specifies the navigation sequence between two screens in terms of screens, navigation conditions and links.

- *Test Scenarios*: This step specifies the navigation conditions in terms of the states of the screen, rather than pre and post conditions.

- *Test Cases*: This step specifies the actual test input data, actions and the expected output.

The 3-level approach is illustrated as a tree structure as shown in figure 2. This tree structure promotes test case reuse and supports test case maintenance [Poonawala 1997]. Once a navigation path is obtained, one can generate a set of test scenarios based on the states of the departure screens. And for each test scenario, one can derive a set of test cases by replacing the states with actual data values. This 3-level approach follows the same approach as in [Kirani 1994, Tsai 1999].

Each of these levels will be discussed in the following subsections.

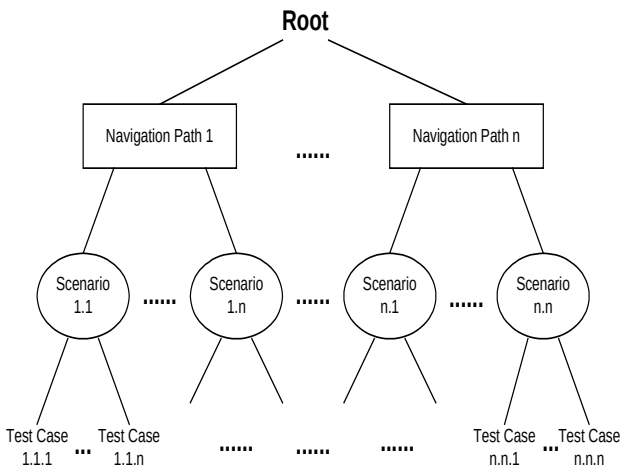


Figure 2. Tree of Three-level Test Case Generation

4.1 Navigation Path Generation

Navigation paths can be generated by tracing the SNet model. The SNet is a graph model, thus one can use various graph traversing algorithms such as the depth-first search or breadth-search search algorithms to generate navigation paths. One may also generate navigation paths based on certain test coverage criteria such as:

- All the navigation paths between two screens;
- A set of independent paths between two screens;
- All the paths between two screens that include a set of specific screens such as the beginning and ending screens; and
- All the paths between two screens that include a set of specific links;
- The set of paths that include all the screens; and

- The set of paths that include all the links;

Figure 3 shows an algorithm to generate all the navigation paths between two arbitrary screens in a SNet model using the depth-first search algorithm.

```
Data Structure:
PathArray    arrPaths;
LinkArray    aPath;

Algorithm:

1. Diagram::Search (CString from, CString to)
1)   Remove all elements in arrPaths
2)   Create a new LinkArray aPath
3)   Get the start and end nodes pFrom, pTo
4)   Call RecursiveSearch (pFrom, pTo, aPath)

2. Diagram::RecursiveSearch
(CNode* pFrom, CNode* pTo, LinkArray* aPath)
  {
     if pFrom = pTo
        add a new path, aPath, to the arrPaths
     else {
        for each link that starting from pFrom do {
           get the node that the link ends at
                 to pNextNode;
           if pNextNode is not in aPath {
              add the link to aPath;
              RecursiveSearch (pNextNode,pTo,aPath);
              Remove the last link in aPath;
           }
        }
     }
  }
```

Figure 3. A Depth-First Algorithm for Navigation Path Generation

4.2 Test Scenario Generation

The conditions to move from one screen to another are based on the state of the departure screen. Thus, once a navigation path has been obtained, one can generate specific test scenario by the examining the possible states of the departure screens in the path.

For example, consider two screens A and B. In any state, A can navigate to the default state of B on the occurrence of event E. Assume that A has 3 states: s1, s2, s3 and the default state of B is s0. Then, the 3 scenarios corresponding to the navigation path A → B, can be illustrated in figure 4:

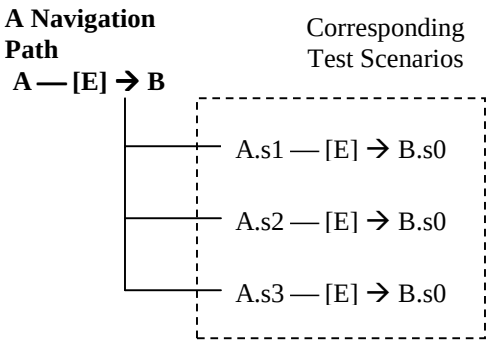


Figure 4. An Example of Deriving Test Scenarios From a Navigation Path

4.3 Test Case Generation

Once a test scenario has been generated, one can generate specific test cases to drive the test scenario.

Test data can be generated based on different considerations, such as:

- Different widget types, such as a text field which allows users to input whatever they want, and a combo box which enumerates all the possible input values
- Different data types, such as integers for which boundary value based test data generation technique can be used, strings for which random test data generation technique can be used
- Different data source, such as data defined by database

- Different data attributes, such as independent or dependent

Figure 5 illustrates test cases generated from a test scenario using the Windows NT logon as the example. The navigation path is from screen “Begin Logon” to “Logon Information”, and finally to screen “Logon Help”. The event “Ctrl+Alt+Delete” causes the screen “Begin Logon” in state S0 to navigate to the screen “Logon Information” in state S0. And the event “Button Help click” causes screen “Logon Information” in any state navigated to “Logon Help” in state S0.

The test scenario taken here is the “Valid User” state of the screen “Logon Information”. Assume one of the 4 valid users is “name bxy, password bxy, domain selab1”. Figure 5 shows the test cases corresponding to the test scenario.

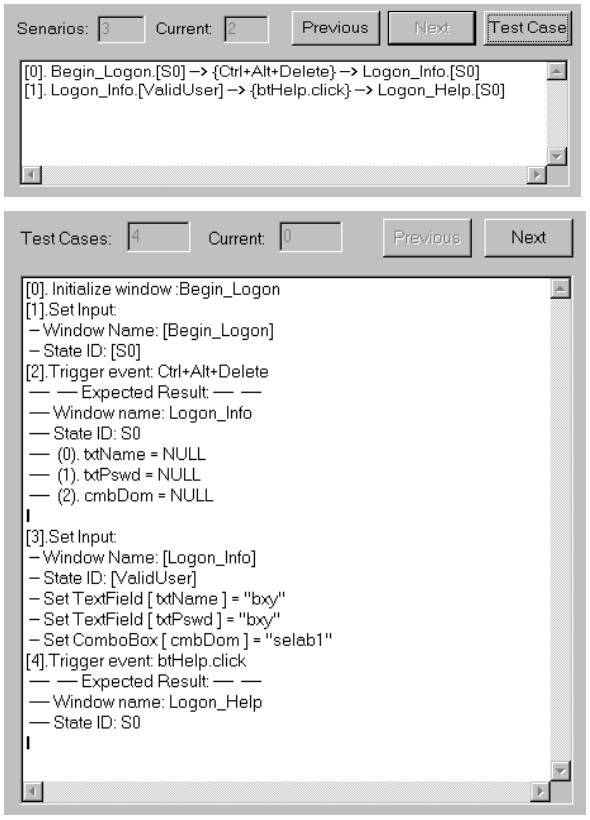


Figure 5. An Example of Generating Test Cases from a Test Scenario

5 Tools and Experimentation

A prototype tool has been constructed to facilitate the interactive specification and to automate the test case generation. The tool runs on Windows NT and is built using Microsoft’s Visual C++.

5.1 Tool Prototype

The prototype system is made up of three modules:

- *Data dictionary*: The storage of the general information of the GUI system: the screens, their widgets and navigation events.

- *SNet Manager*: Provide an interface for user to interactively construct, store and maintain the navigation diagram.

- *Automatic Test Case Generator*: Based on the SNet constructed, the test case generator generates the navigation paths, test scenarios and test cases in a description language as shown in figure 5.

5.2 Experimental Results

Experiments on several systems show that the SNet model is helpful for understanding the GUI application and simplifying test case generation. The model also promotes reusability and facilitates change management. Two examples are illustrated below.

The first example is the Windows NT Logon system, and its SNet specification is in Appendix A. Assume that there are four valid users in two domains. The following table

specifies the number of states and navigation steps of each screen.

Screen	States	Navigations
Begin Logon	1	1
Logon Information	5	5
Logon Message	1	1
Logon Help	1	1
Shutdown Computer	2	3

The following table lists the number of scenarios and test cases generated from the screen "Begin Logon" to each of the other nodes (loops are bypassed).

Destination	Scenarios	Test Cases
Logon Information	1	1
Logon Message	3	9
Logon Help	5	15
Shutdown Computer	5	14
Exit	11	32
Total	**25**	**71**

In another experiment, a Simple Automatic Teller Machine (SATM) is taken as the example system. This version (Appendix B) is a revision of an example in [Paul 95]. It has twelve screens. The following table specifies the system in terms of the number of states and navigation links.

Screen	States	Navigations
Screen 1	1	1
Screen 2	4	3
Screen 3	1	1
Screen 4	4	3
Screen 5	3	2
Screen 6	4	4
Screen 7	1	1
Screen 8	1	1
Screen 9	1	1
Screen 10	2	2
Screen 11	1	1
Screen 12	1	1

For a user account database with four accounts, 27 scenarios and 105 test cases are generated from screen 1 (the start point for all scenarios) to screen 12 (the end screen for all scenarios).

For the first example, it took 10 minutes to build the data dictionary and another 10 minutes to construct the SNet model. Once specified, the prototype tool automatically generates 71 test cases.

For the second example, it took 15 minutes to create the data dictionary and another 20 minutes to construct the SNet model. And the tool generates 105 test cases.

The manual test case generation takes much more time and effort. Even assuming each test case can be developed in 10 minutes by an experienced tester, it will take 710 minutes and 1050 minutes to generate the same number of test cases for these two examples respectively.

6 Conclusion

This paper presents a way to generate test case for GUI testing based on SNet. The SNet uses a directed graph with conditions attached to model the navigation behavior of the GUI system. The technique has the following advantages:

- The testing techniques separate GUI testing into two parts, screen navigation and widget logic. By separating GUI testing into these parts, the complexity and testing effort can be significantly reduced.

- The model is formal and allows users to extract and specify the navigation aspect of the GUI system. Once

specified, test engineers can use the specification to generate test cases.

- The test case generation using the SNet model is hierarchical and systematic, and can use the traditional test coverage criteria to ensure adequacy of testing. These features also significantly reduce the time and effort in GUI testing, and at the same time provide some assurance for reliability.

- An important by-product of the multilevel test case generation is that all the intermediate results (navigation paths, test scenarios, test cases) can be reused at their levels. This helps in maintenance of test cases as well as facilitates change management.

7 Reference

[QA] Segue Software. Inc., QA Partner, available at: http://www.segue.com.

[QES] QES Inc. Flexible, productive automated GUI testing, available at: http://www.questest.com/.

[Star] Sun Microsystem. Inc., JavaStar Overview, available at: http://www.sum. com/.

[Swing] Sun Microsystems. Inc, All about JFC and Swing, Available at: http:// java.sun.com.

[TM] Teradyne Inc., Software and System Test, available at: http://www.teradyne.com.

[XR] Mercury Interactive Corporation, XRunner, available at: http://www.merc-int.com.

[VB] Microsoft Corporation, Visual Basic, available at: http://msdn.microsoft.com.

[VC] Microsoft Corporation, Visual C++, available at: http://msdn.microsoft.com.

[Clarke 1984] L. A. Clarke, and D. J. Richardson, “Validation by Symbolic Evaluation or Symbolic Execution”, in *Software Validation*, edited by H. L. Hausen, North-Holland, 1984.

[Kirani 1994] S. Kirani and W. T. Tsai, “Specification and Verification of Object-Oriented Programs”, technical report, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, 1994.

[Paradkar 1996] A. Paradkar, K. C. Tai, M. A. Vouk, “Automatic Test Generation for Predicates”, Seventh International Symposium on Software Reliability Engineering, 1996. Proceedings, 1996, pp. 66 –75.

[Paul 1995] Paul C. Jorgensen, *Software Testing: a Craftsman’s Approach*, CRC Press, New York, New York, 1995.

[Poonawala 1997] M. Poonawala, S. Subramanian, R. Vishnuvajjala, W. T. Tsai, R. Mojdehbakhsh, and L. Elliott, "Testing Safety-Critical Systems -- A Reuse-Oriented Approach", Proc. of 9th International Conference on SEKE, 1997, pp. 271-278.

[Tsai 1990] W. T. Tsai, Volovik, and T. F. Keefe, “Automatic Test Case Generation for Programs Specified by Relational Algebra Queries”, IEEE Transactions on Software Engineering, Vol. 16, No. 3, March 1990, pp. 316-324.

[Tsai 1993] W. T. Tsai, W. Xie, I. A. Zualkernan, and S. K. Musukula, "A

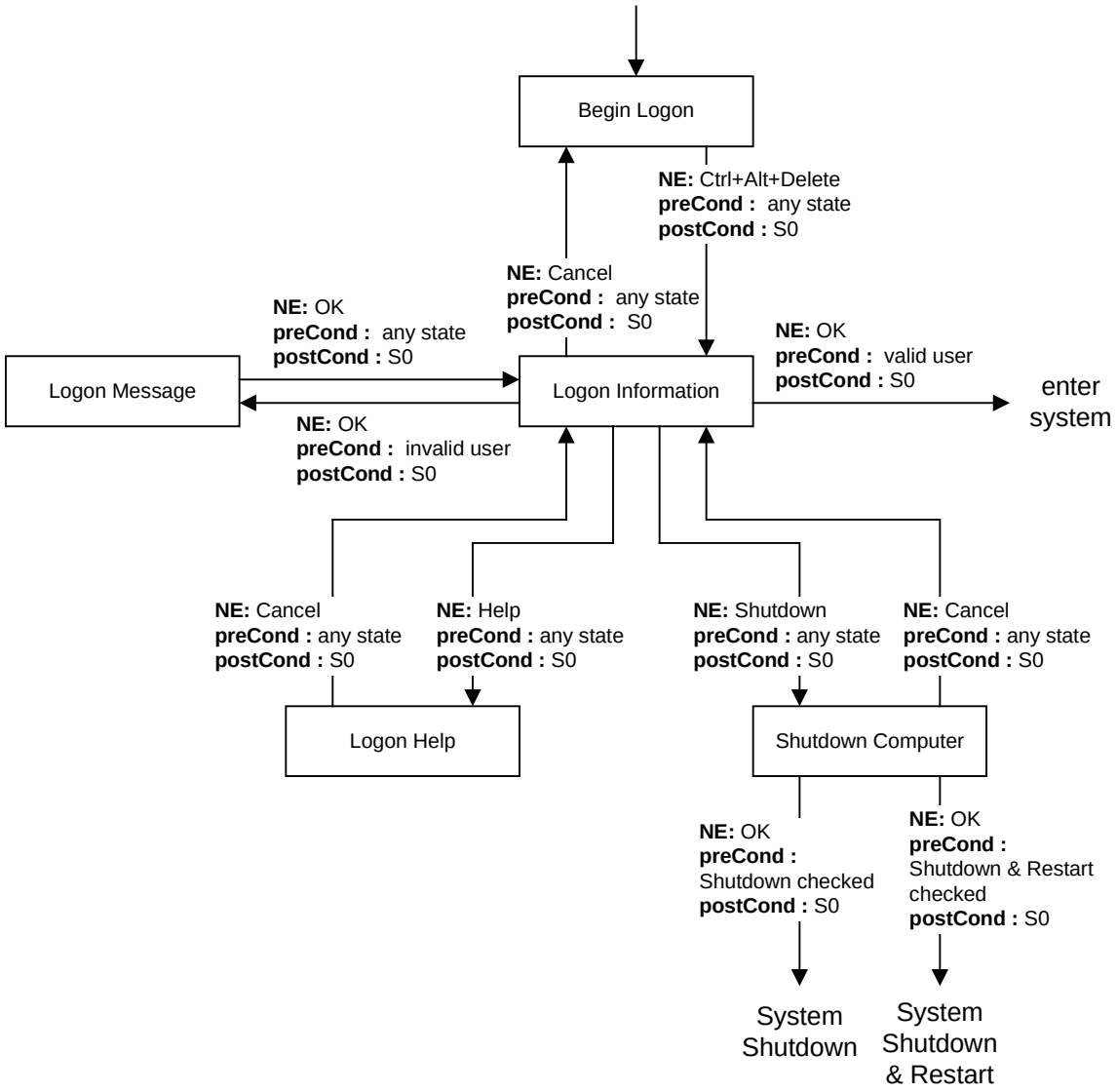
Framework for Systematic Testing of Software Specifications", Proc. of International Conference on Software Engineering and Knowledge Engineering, 1993, pp. 380-387.

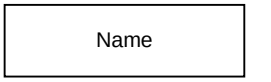
[Tsai 1999] W. T. Tsai, Y. Tu, W. Shao and E. Ebner, “Testing Extensible Design Patterns in Object-Oriented Frameworks through Hierarchical Scenario Templates”, Proc. of IEEE COMPSAC, 1999.

[Weber 1994] R. Weber, K. Thelen, A. Srivastava, and J. Krueger, “Automated Validation Test Generation”, Digital Avionics Systems Conference, 1994. 13th DASC., AIAA/IEEE , 1994 , pp. 99 –104.

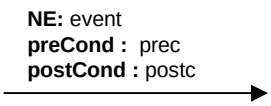
[Yin 1997] H. Yin, Z. Lebne-Dengel, and Y.K. Malaiya, “Automatic Test Generation Using Checkpoint Encoding and Antirandom Testing”, The Eighth International Symposium on Software Reliability Engineering, 1997 Proceedings, 1997, pp. 84 –95.

Appendix A: SNet Diagram for Logon GUI System





The screen of the GUI system



A navigation step:
event: the event that triggers the navigation
prec: the precondition of the navigation
postc: the post conditon of the navigation

Appendix B: the SATM system

There are 12 screens in the system, as shown in figure 6.

The navigation logic through these 12 screens are illustrated in figure 7:

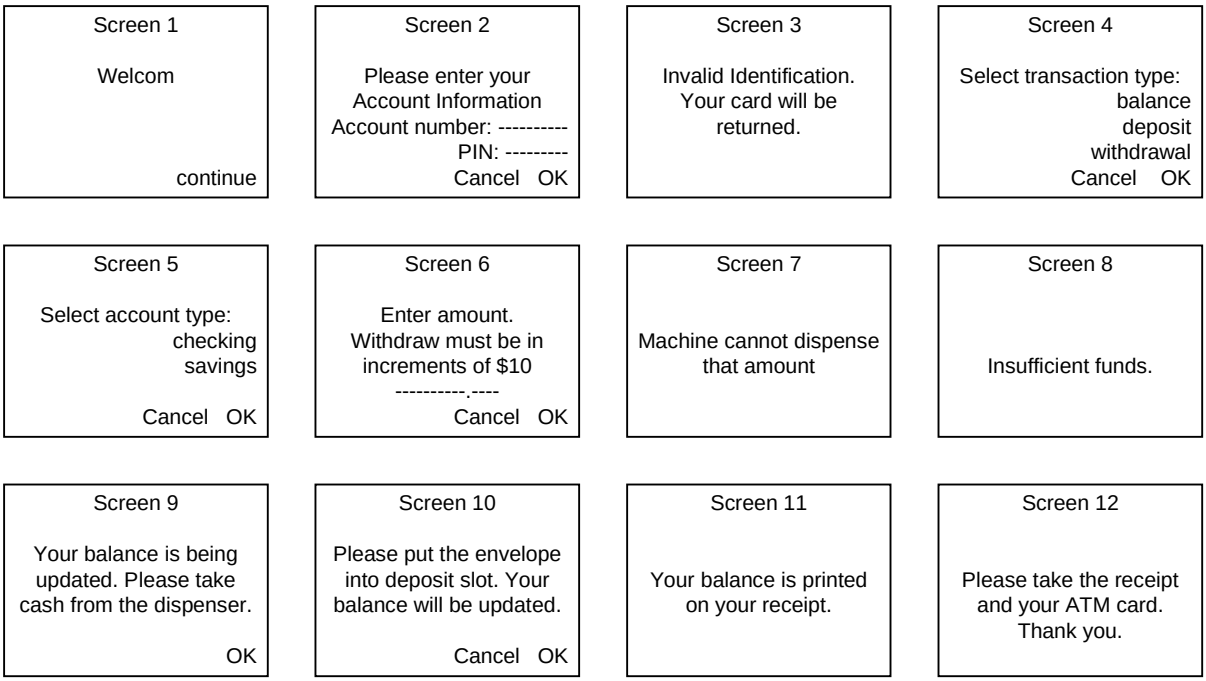


Figure 6. Screens for the SATM system

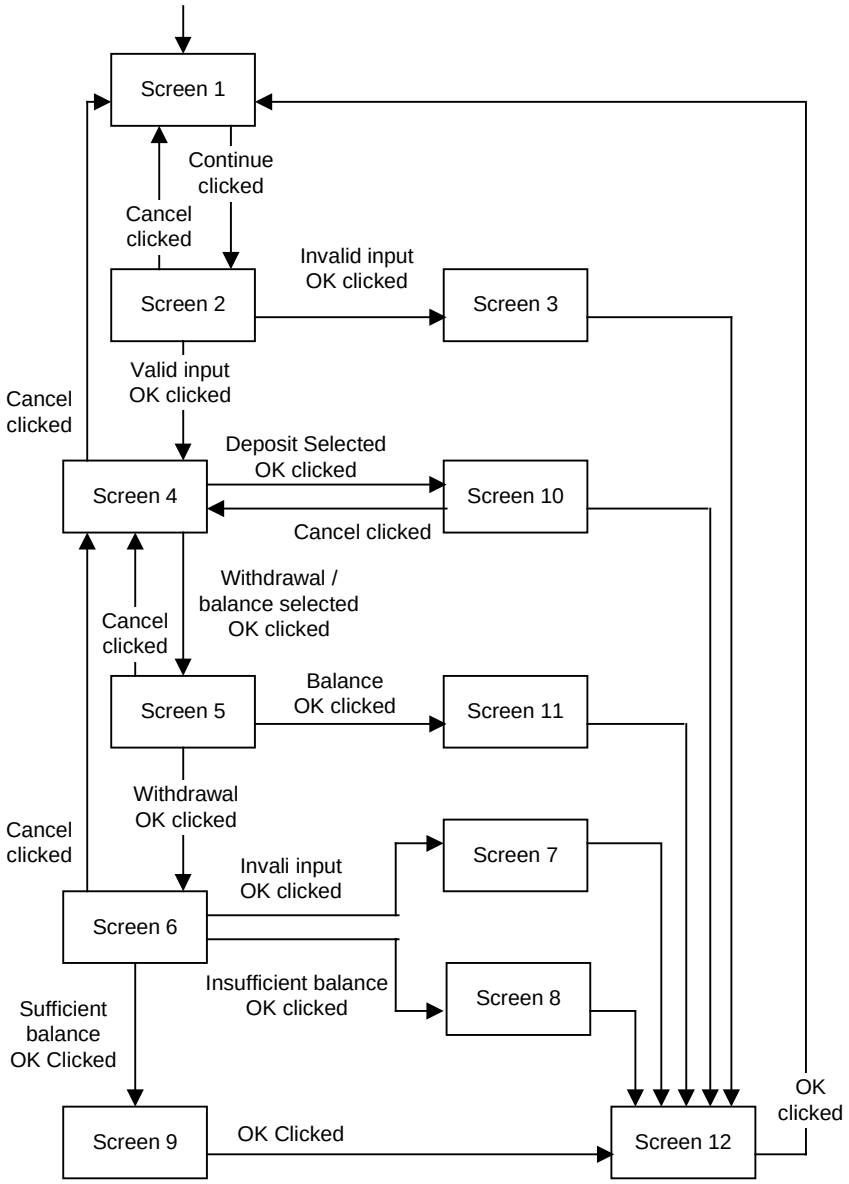


Figure 7. Screen Navigation Diagram for SATM System

QW2000 Paper 8A2

Mr. Giuseppe Lami, Ms. Stefania Gnesi, Prof. Mario Fusani & Mr. Fabrizio Fabbrini
(Istituto di Elaborazione dell'Informazione)

Quality Evaluation of Software Requirements Specifications

BACK TO QW2000 PROGRAM

Presentation Abstract

The criticality of the Software Requirements Specifications (SRS) phase is widely recognised for the success of the whole software project and the attention played on it by the software developers is more and more significant. The software science developed in the past methods and techniques for producing quality SRS: structured languages, controlled languages, formal languages.

Nevertheless the SRS phase is, especially in the Small and Medium Enterprises (SME) community, still one of the weaker steps of the whole software process. One of the principal reason of this situation is that SME, stressed by the time-to-market, do not have enough resources for establishing rigorous methods for the SRS. Furthermore such rigorous methods, should be shared among all the parties involved in a software project, customer included, asking for a further effort in terms of resources to be expended. It is not surprising then that the Natural Language (NL) SRS are, in spite of their inherent inaccuracies, still the most used technique for SRS.

With the objective of transferring methodologies and tools for improving the software development process of SMEs, the IEI/CNR started a new project, funded by CNR, aiming to realise an automatic tool supporting the analysis and quality evaluation of NL SRS. The adopted approach has been, first to define a Quality Model for SRS, then to verify this Model on real cases of SRS in order to be sure that the Quality Model provides a real contribution for solving some NLSRS related problems for SMEs. Finally we have developed an automatic tool for the verification of the requirements to guarantee their conformance to the Quality Model itself. The process for evaluating SRS by using the defined Quality Model and the corresponding Tool is shown in the figure below:

The Quality Model against which to evaluate the quality of NLSRS has been defined so that the Quality Properties included in it can be really evaluated by means of Metrics automatically and numerically computable. The Quality Model inovolves the following components:

Requirements
Specification
Process
Requirements
Document
Review
Process
Quality
Model
Expert Reviewer
TOOL
produces
supports
performs
determines
changes

Definitions of Terms:

- Implicit Subjects Sentences: A sentence is an implicit subject sentence if its subject contains a demonstrative adjective, is expressed by means of pronouns, is specified by a preposition as: above, below, or by an adjective as such: previous, next, following, last, first, ...
- Optional Sentences: A sentence is optional if it contains an option (e.g. it contains words as: possibly, eventually, if appropriate, if needed .....
- Subjective Sentences: A sentence is subjective if it refers to personal opinion or feeling
- Vague Sentences: A sentence is vague if it contains a vague adjective or derived adverb
- Weak Sentences: A sentence is weak if it contains a weak verb (Weak verbs are: can / could, may)
- Underspecified sentences: A sentence is underspecified if its subject contains a word identifying a class of objects without a modifier specifying an instance of such class.
- Multiple sentences: A Sentence is multiple if it has more than one subject or more than one main verb or more than one direct or indirect complement that specifies its subject

The above Properties, although not covering all the possible quality properties of NLSRS, are sufficiently specific to compare and verify the quality of SRS Documents. The above Metrics represent tangible aspects of SRS that can be automatically calculated and they can provide information on the related Quality Property of the Model.

In order to verify if the defined Quality Model provides significant and useful results on real cases, it has been applied on two different sets of documents of real industrial requirements specifications. The first is a set of Telecommunication Software Requirements Specification documents, the second is a case of Safety Critical Space Software Requirements Specification documents. In both cases the organizations that produced these SRS use standards for writing them, perform revision sessions and produce the related reports. The results achieved, on a set of over 800 requirements, are shown below:

The results obtained are encouraging because many possible problems, then risk identifications, have been detected and pointed out, and that has been appreciate by the firms who provided the SRS.

About the Authors

Fabrizio Fabbrini obtained his degree in Computer Science from the University of Pisa, Italy, in 1974. Since 1975 he has served as a scientific researcher at the Institute for Information Processing (IEI) of the Italian National Research Council (CNR), where now he is Senior Researcher. Fabrizio Fabbrini's present activity is focused on Software Quality, and more precisely on the development of methodologies and standards for the assessment and the evaluation of software products and processes, with particular attention to Software Engineering Standards and Software Certification. Software Process Assessment & Improvement, Software Verification & Validation, Computer Security & Data Privacy represent the main fields of application of such research activities.

Mario Fusani obtained his degree in Electrical engineering from the University of Pisa, Italy, in 1971. Since 1973 he has served as a scientific researcher at the Institute for Information Processing (IEI) of the National Research Council (CNR), where now he is Senior Research. His present activity is focused on Software Quality, including the development of methodologies and standards for the assessment and the evaluation of software products and processes.

Stefania Gnesi graduated cum laude in Computer Science at the University of Pisa, Italy. Since 1984 she has been a researcher in the Programming Languages and Operating Systems group of the Italian National Research Council (CNR), Pisa, Italy. Her current research interests include methods and tools for the high-level specification and formal verification of concurrent systems, and applications of temporal logic.

Giuseppe Lami since 1994, when he obtained the degree in Computer Science from the University of Pisa, Italy, focused his interests on Software Engineering and Software Quality issues. His current research activity at the Institute for Information Processing (IEI) of the National Research Council (CNR) includes methods and tools for software testing and requirements engineering.

BACK TO QW2000 PROGRAM

QUALITY EVALUATION OF SOFTWARE REQUIREMENTS SPECIFICATIONS

F. Fabbrini, M. Fusani, S. Gnesi, G. Lami
IEI-CNR, Pisa, Italy

REQUIREMENTS ENGINEERING

Requirements Engineering phase	Level of Detail	People involved
Requirements Definition: statement, in a natural language plus diagrams, of the services the system is expected to provide and the constraints under which it must operate. Generated using customer-supplied information.	General	Managerial Level End-users System architects Client engineers
Requirements Specification: structured document which sets out the system services in detail. This document should be precise. It may serve as a contract between the system buyer and software developer	Intermediate	End-users System architects Client engineers SW developers
Software Specification: abstract description of the software which is a basis for design and implementation.	Highly detailed	System architects SW developers

Techniques for Producing SRS

Approach to Requirements Specifications	*Description*
Natural Language	-
Structured Natural Language	restricted natural language where the terminology is limited and templates can be used. Control constructs derived from programming languages can be included.
Semi-formal Languages	usually special-purpose graphical notations with a precise syntax and a non-rigorous semantic.
Formal Languages	mathematics based languages with vocabulary, syntax and semantics formally defined.

SRS Quality Evaluation

Goal Properties

- Requirements Sentence Quality (RSQ)

  the syntactical quality of single sentences considered separately.

- Requirements Document Quality (RDQ)

  the quality of the sentences considered in the context of the whole requirements document.

SRS Quality Evaluation

RSQ related Properties

- **Non-Ambiguity**: the capability of a Requirement to have a unique interpretation.
- **Completeness**: the capability of each Requirement to make reference to precisely identified entities.
- **Understandability**: the capability of each Requirement to be fully understood when used for developing software.

SRS Quality Evaluation

RDQ related Properties

- **Completeness:** the capability of the Requirements Specification Document to avoid potential or actual discrepancies.
- **Understandability:** the capability of the Requirements Specification Document to be fully understood when read by the user.

SRS Quality Evaluation

NL SRS Quality Indicators

syntactic aspects of the requirements specifications that can be automatically calculated and that provide information on a particular Quality Property of the SRS.

A NLSRS Quality Model

Goal Properties	Properties	Quality Indicators
Requirement Sentences Quality	Non-Ambiguity	Implicit Subjects Sentences, Optional Phrases, Subjective Sentences, Vague Sentences, Weak Sentences
	Completeness	Underspecified Sentences
	Understandability	Multiple Sentences
Requirement Document Quality	Completeness	Underreferenced Sentences
	Understandability	Comments Frequency, Readability Index, Unexplained Sentences

Ambiguity Indicators

- A sentence is an **implicit subject sentence** if its subject:
  - contains a demonstrative adjective: *this, these, that, ...*
  - is expressed by means of pronouns: *it, they, ...*
  - is specified by a preposition as such: *above, below, ...*
  - is specified by an adjective as such: *previous, next, following, last, first, …*

(Example: The above requirements shall be verified by test)

Ambiguity Indicators

- A sentence is **optional** if it contains an option.

  Note: A sentence is to be considered as optional if it contains words as such: *possibly, eventually, if appropriate, if needed .....*

(Example: The system shall be such that the mission can be pursued, possibly without performance degradation.)

Ambiguity Indicators

- A sentence is **subjective** if it refers to personal opinion or feeling.

  Note:The sentences containing the following kinds of wording are to be pointed out as potentially subjective:

  – *having in mind, take (into) account, take into consideration, ...*

  – *similar, better, similarly, worse, ...*

  – *as [adjective] as possible.*

(Example: Software tasks shall be as much as possible synchronous.)

Ambiguity Indicators

A sentence is **vague** if it contains words holding inherent vagueness.

Note: Words holding inherent vagueness:

- Adjectives related to intrinsic, environmental, time, location characteristics of an object;
- Adverbs derived from the previous adjectives;
- Vague nouns or verbs (e.g. support)

(Example: The C code shall be clearly commented.)

Ambiguity Indicators

A sentence is **weak** if it contains a weak verb.

Note: Weak verbs are:

- *can, could, may*.

(Example: The results of the Initialization checks may be reported in a special file.)

Completeness Indicators

A sentence is **underspecified** if its subject contains a word identifying a class of objects without a modifier specifying an instance of such class.

(Example: The testers shall be independent of the software developers.)

Understandability Indicators

A Sentence is **multiple** if:

- it has more than one subject or more than one main verb;
- it has more than one direct or indirect complement that specifies its subject.

(Example: The mean time needed to remove a faulty board and restore service shall be less than 30 minutes.)

Doc. Completeness Indicators

- A sentence of a Requirements Document (RD) is **underreferenced** if it contains explicit references to:
  - not-numbered sentences of the RD itself,
  - documents not-referenced into the RD itself,
  - entities not defined nor described within the RD itself.

  Note: The following kinds of wording are to be considered as explicit references:
  according to, on the basis of, relatively to, compliant with, conformant to

(Example: The Software shall be designed according to the rules of the Object Oriented Design)

Doc. Understandability Indicators

The **Comment Frequency** of a Requirements Document is the value of the CFI (Comment Frequency Index).

(CFI = NC / NR where:

NC is the total number of Requirements having one or more comments,

NR is the total number of Requirements of the document).

Doc. Understandability Indicators

The **readability** index is the value of ARI (Automated Readability Index).

(ARI=WS + 9SW where:

WS is the average words per sentence,

SW is the average letters per word).

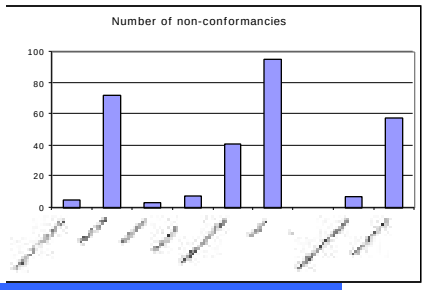
Doc. Understandability Indicators

A sentence of a Requirements Specification Document (RSD) is **unexplained** if it contains acronyms not explicitly and completely explained within the RSD itself.

(Example: The handling of any received valid TC packet shall be started in less than 1 CUT.)

Validation of the Quality Model

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	5	1,3%
Multiple Sentences	72	18,2%
Optional Sentences	3	0,8%
Subjective Sentences	8	2%
Underspecified Sentences	41	10,4%
Vague Sentences	95	24,1%
Weak Sentences	0	0%
Underreferenced Sentences	7	1,8%
Unexplained Sentences	58	14,7%



TLC Application SRS

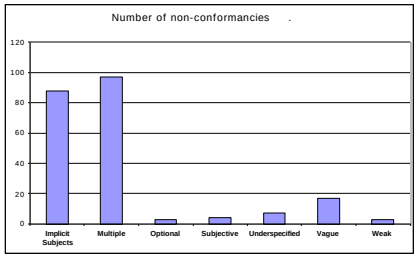
Total number of Requirements Evaluated = 395

Warnings

rate = 73%

Validation of the Quality Model

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	88	19,8%
Multiple Sentences	97	21,8%
Optional Sentences	3	0,7%
Subjective Sentences	4	0,9%
Underspecified Sentences	7	1,6%
Vague Sentences	17	3,8%
Weak Sentences	3	0,7%



Space Application SRS

Total number of Requirements Evaluated = 444

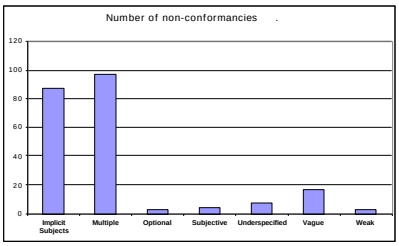
Warnings

rate = 49%

Validation of the Quality Model

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	93	11,1%
Multiple Sentences	169	20,1%
Optional Sentences	6	0,7%
Subjective Sentences	12	1,4%
Underspecified Sentences	48	5,7%
Vague Sentences	112	13,3%
Weak Sentences	3	0,4%
Underreferenced Sentences	7**	
Unexplained Sentences	58**	

***: calculated only for the Telecommunication Software Requirements Specification documents.*



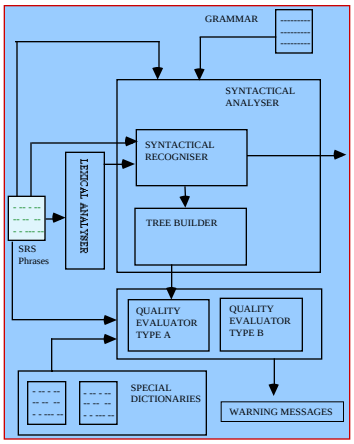
Joint Results

Total number of Requirements Evaluated = 839

Warnings

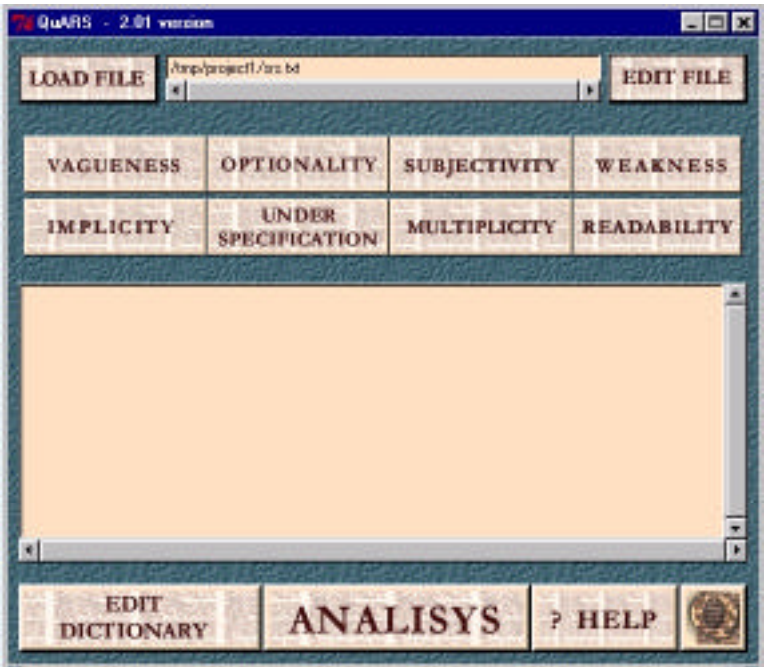
rate = 53%

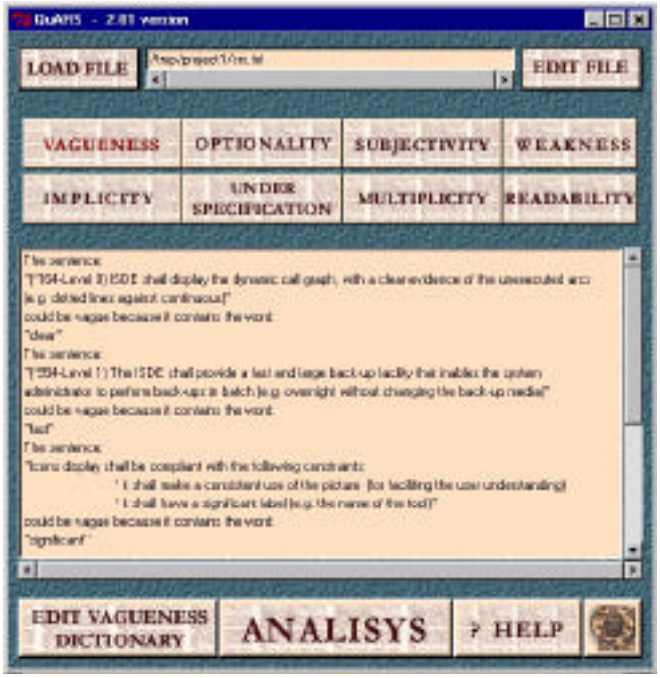
QuARS Tool

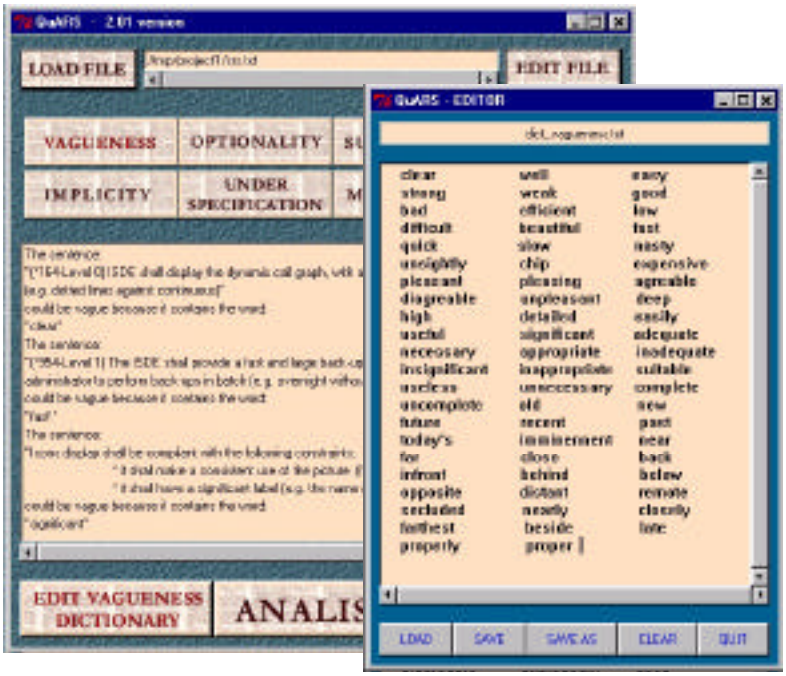


QuARS:

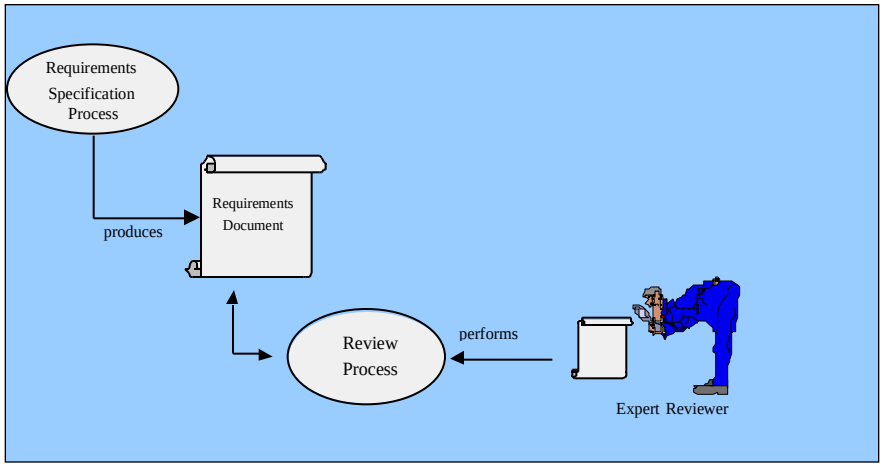
Quality Analyser of Requirements Specifications


QuARS - 2.01 version
LOAD FILE
EDIT FILE
VAGUENESS
OPTIONALITY
SUBJECTIVITY
WEAKNESS
IMPLICITY
UNDER SPECIFICATION
MULTIPLICITY
READABILITY
EDIT DICTIONARY
ANALISYS
? HELP

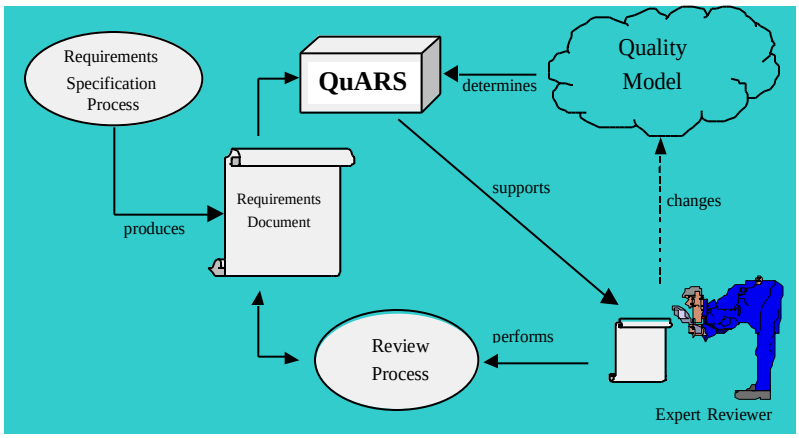

QuARS - 2.01 version
LOAD FILE
EDIT FILE
VAGUENESS
OPTIONALITY
SUBJECTIVITY
WEAKNESS
IMPLICITY
UNDER SPECIFICATION
MULTIPLICITY
READABILITY
EDIT VAGUENESS DICTIONARY
ANALISYS
? HELP



NL SRS Evaluation Process



NL SRS Evaluation Process



Conclusions

- A Model for the syntactic quality of SRS
  - incomplete
  - including indicators (metrics) numerically and automatically computable
- An automatic Tool providing support for the Quality Evaluation of SRS by means of calculation of the Quality Model metrics

Future Work

- To make QuARS able to provide automatic calculation of all the Quality Indicators of the Quality Model;
- To include an on-line SRS editor
- To provide Guide Lines for tailoring the Quality Model according to particular needs

Quality Evaluation of Software Requirements Specifications

F. Fabbrini, M. Fusani, S. Gnesi, G. Lami

I.E.I. - C.N.R. Pisa, Italy

The criticality of the Software Requirements Specifications (SRS) phase of the software life cycle for the success of the whole software project is widely recognized and the attention played on it by software developers is more and more significant. The software science developed in the past methods and techniques for producing quality SRS: structured languages, controlled languages, formal languages.
Nevertheless the SRS phase is, especially in the Small and Medium Enterprises (SME) community, still one of the weakest steps of the whole software process. One of the principal reasons of this situation is that SMEs, stressed by the time-to-market, do not have enough resources for establishing rigorous methods for the SRS. Furthermore such rigorous methods should be shared among all the parties involved in a software project, customer included, asking for a further effort in terms of resources to be expended.
It is not surprising then that Natural Language (NL) SRS are, in spite of their inherent inaccuracies, still the most used technique for SRS.
In this paper we describe the effort made at I.E.I - C.N.R. for realizing an automatic tool (called QuARS - Quality Analyzer of Requirements Specifications) supporting the analysis and quality evaluation of NL SRS. The adopted approach has been first to define a Quality Model for SRS, then to verify this Model on real cases of SRS in order to be sure that the Quality Model provides a real contribution for solving some NLSRS related problems for SMEs. Finally we have developed an automatic tool for the verification of the requirements to guarantee their conformance to the Quality Model itself.
This paper is organized as follows: in Section 1 we provide general considerations on Software Requirements Specification by considering NL techniques and other more formal techniques. In Section 2 we describe the proposed Quality Model for NL SRS. In Section 3 we describe the outcomes of the experimentation of the proposed Quality Model on real industrial cases. In Section 4 we describe the functionalities of the QuARS tool. In Section 5 we focus on the Quality Evaluation Process of NL SRS, and, finally, in Section 6 we discuss conclusions and future works.

* This work was partially supported by SSSUP S.Anna in the framework of the LINK project, MURST L. 488/92.

1. Techniques for producing Software Requirements Specifications

The Requirements Engineering is the process of establishing the services the system should provide and the constraints under which it must operate [13].

Ideally the Requirements Engineering process should be separated into three sub-processes, each needing different styles, levels of detail and people to be involved. The separation of these processes should guarantee the production of a suitable final output toward the software designer. The next table summarizes the principal aspects of these processes [13]:

Requirements Engineering phase	Level of Detail	People involved
Requirements Definition: statement, in a natural language plus diagrams, of what services the system is expected to provide and the constraints under which it must operate. It is generate using customer-supplied information.	General	Managerial Level End-users System architects Client engineers
Requirements Specification: a structured document which sets out the system services in detail. This document should be precise. It may serve as a contract between the system buyer and software developer	Intermediate	End-users System architects Client engineers SW developers
Software Specification: abstract description of the software which is a basis for design and implementation.	Highly detailed	System architects SW developers

The expected outputs of the previous phases are documents that gradually fill the gap between the conceptual understanding of the system we want to build and the detailed description of its functionalities that have to be fed to the system architects.
The phase of the Requirements Engineering Process addressed in this paper is the Requirements Specification.

The scheme presented before is an ideal scheme and often it doesn't correspond to the practices of software firms. In fact, the real software process of many firms considers the Requirements Engineering process as composed of an unique phase aimed to provide a document describing, in some way, the Requirements of the system to be developed.
There are several risks associated with this practice, as, for example, those listed below:

- The requirements related to different functionalities of the system may be not expressed with the same level of detail and precision;
- The requirements may be too abstract and then not easily usable by the software designers.
- The involvement of some people in the Requirements Engineering process may be not fully adequate (e.g. software developers, end-users, ..).

We will focus our attention on a product that we can assume will be anyway provided as an outcome of the Requirements Specifications phase: a document describing somehow and with a certain level of precision the functionalities and the constraints of the system under construction.

Several approaches to producing Requirements Specification exist and their precision and cost may vary considerably. We will concentrate our attention on one of these approaches: the use of Natural Language for writing Requirements Specification. Even though it is out of the scope of this paper to compare weaknesses and strengths of different Requirements Specifications approaches and to establish if one of them is better than another, in the next scheme we provide a brief description of the most used approaches.

Approach to Requirements Specifications	*Description*
Natural Language	-
Structured Natural Language	restricted natural language where the terminology is limited and templates can be used. Control constructs derived from programming languages can be included.
Semi-formal Languages	they are usually special-purpose graphical notations with a precise syntax and a non-rigorous semantic.
Formal Languages	mathematics based languages with vocabulary, syntax and semantics formally defined.

It is clear that the effects of the use of each of them on the quality of the Requirements Specifications are different and it is also quite clear that the order in which they are listed into the scheme correspond to an increasing "quality" of the Requirements Specification they produce. In other words, it is clear that Requirements Specifications written by using NL may contain ambiguities, while SRS written by using Formal Methods are more precisely and rigorously understandable.
However, the establishment of a formal method for the SRS needs some initial additional costs. These costs are due to the training of people in order to make them able to use the formal techniques. In other words it is necessary to make all the people involved in the SRS phase, often coming from different social, business and technical contexts, able to share these formalisms.
In spite of these considerations, many SMEs, even though they recognize both the importance and the benefits of the use of formal methods, and the risks due to the use of NL in writing SRS, continue to write SRS using natural languages unless they are not obliged to do otherwise by precise contractual constraints.

Then the fact that writing SRS in NL is not a good practice because they are prone to ambiguities and that they are neither a good input for the design phase nor a good basis for the contractual issue between the customer and the supplier doesn't mean that we have to renounce achieving an acceptable level of quality even in NL SRS. On the contrary we are persuaded that it is useful to have a pragmatic approach to the question and then it is a worth-while effort to provide a contribution in the quality improvement of NL SRS. This approach has been adopted in the definition of the Quality Model and in the consequent realization of the QuARS prototype.

2. A NLSRS Quality Model

The Quality Model defined in this paper aims to provide the means for evaluating quantitatively two aspects of the SRS quality, called Goal Properties:

- Requirements Sentences Quality (RSQ): the syntactical quality of single sentences considered separately; [4]
- Requirements Document Quality (RDQ): the quality of the sentences considered in the context of the whole requirements documents.

These two Goal Properties are obviously too abstract for being directly evaluated. For this reason some Properties have been associated to each Goal Property in order to provide quality characteristics less abstract and then easier to evaluate. Table 1. shows the Quality Model Goal Properties and the related Properties:

Goal Properties	***Properties***
Requirement Sentences Quality (RSQ)	Non-Ambiguity
	Completeness
	Understandability
Requirement Document Quality (RDQ)	Completeness
	Understandability

Table 1.

The Properties related to the RSQ Goal Property included within our Quality Model are:

- Non-Ambiguity: the capability of a Requirement to have a unique interpretation.
- Completeness: the capability of each Requirement to make reference to precisely identified entities.
- Understandability: the capability of each Requirement to be fully understood when used for developing software.

The Properties related to the RDQ Goal Property included within Quality Model are:

- Completeness: the capability of the Requirements Specification Document to avoid potential or actual discrepancies.
- Understandability: the capability of the Requirements Specification Document to be fully understood when read by the user.

The above Goal Properties and Properties have been included into the Quality Model because they can be automatically evaluated by means of tangible Indicators that can be counted/detected during the parsing of the requirements document, and because possible problems related to these Properties can be pointed out precisely so that corrective actions may be easily done.

This Quality Model is focused on linguistic properties of SRS and cannot be considered complete because other aspects of the SRS quality, as for example semantic consistency aspects, exist and they are not taken into account.

2.1 Quality Indicators

Quality Indicators are syntactic aspects of the requirements specifications [14] that can be automatically calculated and that provide information on a particular quality Property of the requirements specifications themselves. Several Indicators have been included in the Quality Model, each associated to a Property.
In the following, the Indicators are listed and described, then the association between them and the Properties of the Quality Model is shown.

The Quality Indicators included in the Quality model are divided into Indicators related to Requirement Sentences Quality (RSQ) and Requirement Document Quality (RDQ):

Requirement Sentences Quality (RSQ) related Indicators:

- Implicit Subjects Sentences
- Multiple Sentences
- Optional Sentences
- Subjective Sentences
- Underspecified Sentences
- Vague Sentences
- Weak Sentences

Requirements Document Quality (RDQ) related Indicators:
- Comment frequency,
- Readability Index
- Underreferenced Sentences
- Unexplained Sentences

In the following the definition of each Quality Indicator is provided along with some examples.

Implicit Subject Sentences:
A sentence is an implicit subject sentence if its subject:
- contains a demonstrative adjective: this, these, that, those
- is expressed by means of pronouns: it, they.
- is specified by a preposition as such: above, below, ...
- is specified by an adjective as such: previous, next, following, last, first, ...

Examples:
Implicit Subject Sentences:
- This counter shall be incremented by 1 each time a RTI occurs.
- It shall be stored in non-volatile memory.
- The above requirements shall be verified by test.

Non-implicit Subject Sentences:
- The MC counter shall be incremented by 1 each time a RTI occurs.
- The detected erroneous data shall be stored in non-volatile memory.
- The requirements number 1 to 56 shall be verified by test.

Multiple Sentences:
A Sentence is multiple if:
- it has more than one subject or more than one main verb
- it has more than one direct or indirect complement that specifies its subject

Examples:
Multiple sentence:
- The System shall provide for the generation of stub/driver skeleton and for the simulation of module interface.
- The customer perception of System Down Time and Partial System Down shall be zero.

Non-multiple sentences:
- The System shall provide for the generation of stub/driver skeleton.
- The System shall provide for the simulation of module interface.

- The customer perception of System Down Time shall be zero.

- The customer perception of Partial System Down shall be zero.

Optional Sentences:

A sentence is optional if it contains an option .

Note: A sentence is to be considered as optional if it contains words as such: possibly, eventually, if appropriate, if needed .....

Examples:

Optional sentence:

- The system shall be such that the mission can be pursued, possibly without performance degradation.

Non-optional sentence:

- The system shall be such that the mission can be pursued, with performance degradation not greater than 10%.

Subjective Sentences:

A sentence is subjective if it refers to personal opinion or feeling.

Note: sentences containing the following kinds of wording are to be pointed out as potentially subjective:

- having in mind, take (into) account, take into consideration, ...
- similar, better, similarly, worse, ...
- as [adjective] as possible.

Examples:

Subjective Sentences:

- The System should allow computation analysis results to be presented on-line during the coding phase (having in mind the basic set of metrics).
- Software tasks shall be as much as possible synchronous.

Non-subjective Sentences:

- The System should allow computation analysis results to be presented on-line during the coding phase by showing values of the basic set of metrics.
- Software tasks shall be synchronous.

Underspecified Sentences:

A sentence is underspecified if its subject contains a word identifying a class of objects without a modifier specifying an instance of such class.

Examples:

Underspecified Sentence:

- The testers shall be independent of the software developers.

Non-underspecified Sentence:

- The module testers shall be independent of the software developers.

Vague Sentences:

A sentence is vague if it includes words holding inherent vagueness.

Note: Vague adjectives are:

- adjectives related to intrinsic characteristics as such: clear, well, easy, strong, weak, good, bad, efficient, low .....
- adjectives related to environmental characteristics as such: useful, significant, adequate, fast, slow....
- adjectives related to time characteristics as such: old, new, future, recent, past, today's, ...
- adjectives related to location characteristics as such: near, far, close, back, in front, ...

Examples:

Vague Sentences:

- The C code shall be clearly commented.
- The User Manual shall provide adequate information about the installation procedure.
- The Reports shall be produced according to the today's trend.

Non-vague Sentences:

- The sentences of the comments into the code shall be no longer than 20 words.
- The User Manual shall provide step by step procedure for the installation.
- The Reports shall be produced according to the template provided in figure 1.2.

Weak Sentences:

A sentence is weak if it contains a weak verb.

Note: Weak verbs are:

- can / could
- may

Examples:

Weak Sentence:

- The results of the Initialization checks may be reported in a special file.

Non-weak Sentence:

- The results of the Initialization checks shall be reported in a special file.

Comment Frequency:

The Comment Frequency of a Requirements Document is the value of the CFI (Comment Frequency Index). (CFI=NC / NR where NC is the total number of Requirements having one or more comments, NR is the total number of Requirements of the document)

Underreferenced Sentences:

A sentence of a Requirements Document (RD) is underreferenced if it contains explicit references to:

- not-numbered sentences of the RD itself,
- documents not-referenced into the RD itself,
- entities not defined nor described within the RD itself.

Note: The following kinds of wording are to be considered as explicit references:

- *according to*
- *on the basis of*
- *relatively to*
- *compliant with*
- *conformant to*

Example:

Underreferenced Sentence:

- The Software shall be designed according to the rules of the Object Oriented Design

Non-underreferenced Sentence:

- The Software shall be designed according to the rules contained in [TD.4]. (where [TD4] is included into the list of references of the same Requirements Document).

Unexplained Sentences:

A sentence of a Requirements Specification Document (RSD) is unexplained if it contains acronyms not explicitly and completely explained within the RSD itself.

Examples:

Unexplained Sentences:

- The handling of any received valid TC packet shall be started in less than 1 CUT.
- The results of the Initialization checks shall be reported in the HK TM.

Readability Index:

The unreadability index is the value of ARI (Automated Readability Index) [6] (ARI=WS + 9SW where WS is the average words per sentence, SW is the average letters per word)

Each of the above Quality Indicators has been associated with a Property and the Quality Model has been then completed. The table 2 shows the Quality Model obtained by including the previously defined Quality Indicators.

Goal Properties	Properties	Quality Indicator
Requirement Sentences Quality	Non-Ambiguity	Implicit Subjects Sentences, Optional Phrases, Subjective Sentences, Vague Sentences, Weak Sentences
	Completeness	Underspecified Sentences
	Understandability	Multiple Sentences
Requirement Document Quality	Consistency	Underreferenced Sentences
	Understandability	Comment Frequency, Readability Index, Unexplained Sentences

Table 2.

2.2 Quality Requirements for Software Requirements Specifications

The Quality Indicators can be automatically detected and counted during the analysis of the requirements specification document. The results of this analysis have to be provided in order to make the user able to point out the sentences that are potentially incorrect. Then the user, on the basis of his experience and skill has to judge if the pointed out sentences are really incorrect or if they can be accepted by taking into account the following quality requirements.

The quality requirements for each Indicator are listed below.

A Requirements Document shall not contain:

- Implicit Subjects Sentences ;
- Multiple Sentences;
- Optional Sentences;
- Subjective Sentences;
- Underspecified Sentences;
- Vague Sentences;
- Weak Sentences;
- Underreferenced Sentences;
- Unexplained Sentences;

A Requirements Document shall:

- have the Comments frequency value greater than an established target value X*;
- have the Readability Index value greater than an established target value Y*.

*: the target values determination is project dependent.

3. Validation of the Quality Model

In order to verify if the defined Quality Model provides significant and useful results on real cases, it has been applied to two different sets of documents of real industrial requirements specifications.
The first is a set of Telecommunication Software Requirements Specification documents, the second is a case of Safety Critical Space Software Requirements Specification documents. In both cases the organizations that produced these requirements specifications use standards for writing them, perform review sessions and produce the related reports.
The results achieved are shown below:

Telecommunication Software Requirements Specification documents analysis and evaluation results

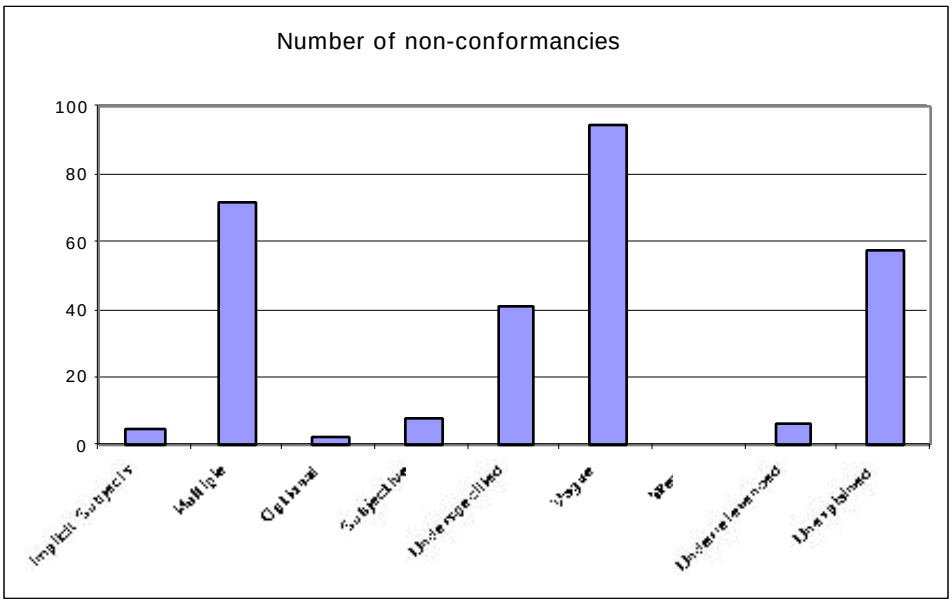
Indicators taken into account:

a) Implicit Subjects Sentences
b) Multiple Sentences
c) Optional Sentences
d) Subjective Sentences
e) Underspecified Sentences
f) Vague Sentences
g) Weak Sentences
h) Underreferenced Sentences
i) Unexplained Sentences

Total number of evaluated requirements: **395**

The following table shows the number of requirements that don’t satisfy the quality requirements:

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	5	1,3%
Multiple Sentences	72	18,2%
Optional Sentences	3	0,8%
Subjective Sentences	8	2%
Underspecified Sentences	41	10,4%
Vague Sentences	95	24,1%
Weak Sentences	0	0%
Underreferenced Sentences	7	1,8%
Unexplained Sentences	58	14,7%



Faults rate = 73%
Note: Faults rate = (number of non-conformances) / (number of requirements)

Safety Critical Space Software Requirements Specification documents analysis and evaluation results:

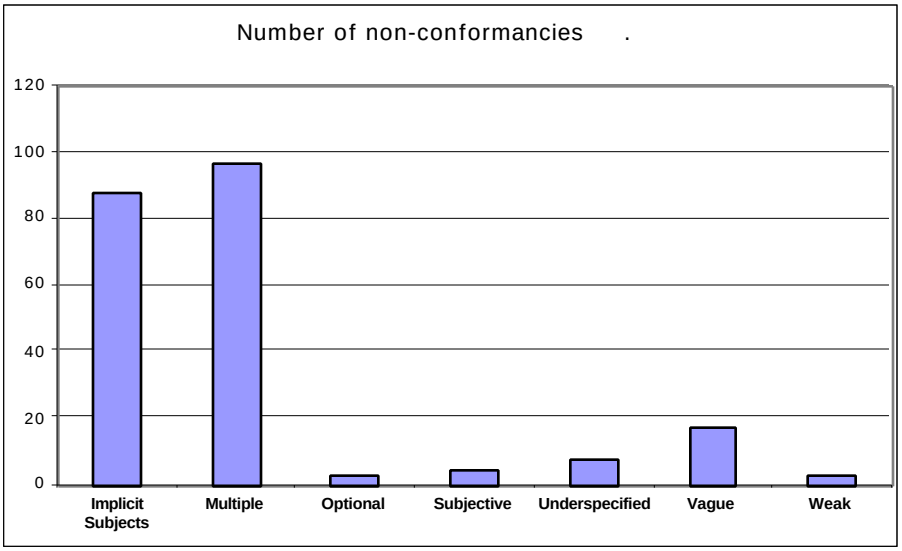
Indicators taken into account:

a) Implicit Subjects Sentences
b) Multiple Sentences
c) Optional Sentences
d) Subjective Sentences
e) Underspecified Sentences
f) Vague Sentences
g) Weak Sentences

Total number of evaluated requirements: **444**

The following table shows the number of requirements don't satisfy the quality requirements:

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	88	19,8%
Multiple Sentences	97	21,8%
Optional Sentences	3	0,7%
Subjective Sentences	4	0,9%
Underspecified Sentences	7	1,6%
Vague Sentences	17	3,8%
Weak Sentences	3	0,7%



Faults rate = 49%

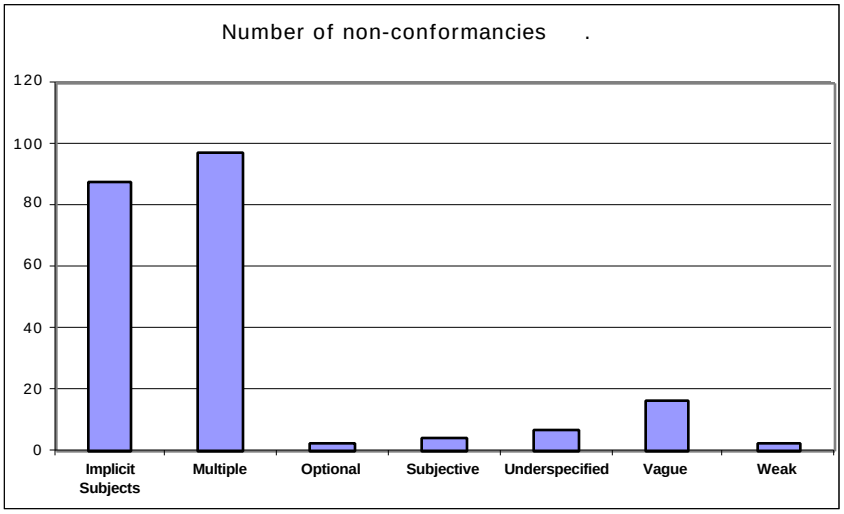
note: Faults rate = (number of non-conformances) / (number of requirements)

Joint results:

Total number of evaluated requirements: **839**

Quality Indicator	number of not conformances	frequency
Implicit Subjects Sentences	93	11,1%
Multiple Sentences	169	20,1%
Optional Sentences	6	0,7%
Subjective Sentences	12	1,4%
Underspecified Sentences	48	5,7%
Vague Sentences	112	13,3%
Weak Sentences	3	0,4%
Underreferenced Sentences	7**	
Unexplained Sentences	58**	

***: calculated only for the Telecommunication Software Requirements Specification documents.*



Faults rate = 53%
note: Faults rate = (number of non-conformances) / (number of requirements)

The firms that provided the analyser with the requirements were presented these results in order to achieve their comments and, in particular, to understand if the potential lacks of quality outlighted by QuARS are considered actually significant and noteworthy by their Requirements Engineers. The outcomes of the discussion with the Requirements Engineering have been encouraging, since QuARS has been considered a valid support for solving part of the problems they encounter during the SRS phase.

4. QuARS (Quality Analyzer of Requirements Specifications) tool

In order to make the analysis of NL SRS automatic, a prototype has been realized at I.E.I.. This prototype named QuARS (Quality Analyzer of Requirements Specifications) has been conceived to perform a parsing of the SRS phrases written in Natural Language and point out a number of potential sources of errors into the SRS themselves.

The principal functionalities provided by QuARS are listed below:

- Lexical Analysis of the SRS sentences;
- Syntactical Analysis of the SRS sentences based on a special-purpose grammar and construction of the derivation trees (structures that represents the possible derivations of the analyzed sentence by using the grammar);
- Analysis of each sentence of the SRS against the Quality Properties by using the Quality Indicators provided within the Quality Model;
- Provision to the user of warnings messages indicating those sentences having potential problems and explanation of these problems;

We don't want to provide implementation details of QuARS in this paper, but figure 1 can help understand how the prototype works.

The structure of QuARS is composed of the following key components:

- Lexical Analyser: it reads the input SRS sentences, performs a morphological analysis of the items of the sentences, verifies if they are in the English Dictionary and produces on output file the appropriate lexical category associated to each word of the sentence.
- Syntactical Analyser: it is composed of two sub-components: the Syntactical Recogniser and the Tree Builder. The first is a parser that on the basis of a special grammar and of the output of the Lexical Analyser recognise if a sentence is syntactically correct. The second component builds the derivation trees for each sentence. A derivation tree is a tree representing a possible derivation of the sentence by using rules of the grammar.
- Properties Evaluator: it is composed of two sub-components: the Evaluator of Indicators of Class A and the Evaluator of Indicators of Class B. The first performs the computation of those Indicators of the Quality Model for which information on the syntactical structure and semantic value of the items of the sentence is needed (e.g. the Underspecified Sentences Indicator). The second component performs the compilation of those indicators that don't require information on the syntactical structure and semantic value of the items of the sentence (e.g. the Vague Sentences Indicator).
- Special Dictionaries: each Indicator has a special dictionary associated that contains the special words necessary for its computation. These Dictionaries have to be considered modifiable when it is necessary to make the evaluation more suitable for particular SRS.

QuARS then has a Grammar, Special Dictionaries and the SRS Sentences as input data and provides the user with messages where the sentences having some potential problems are shown and the problems are outlighted.

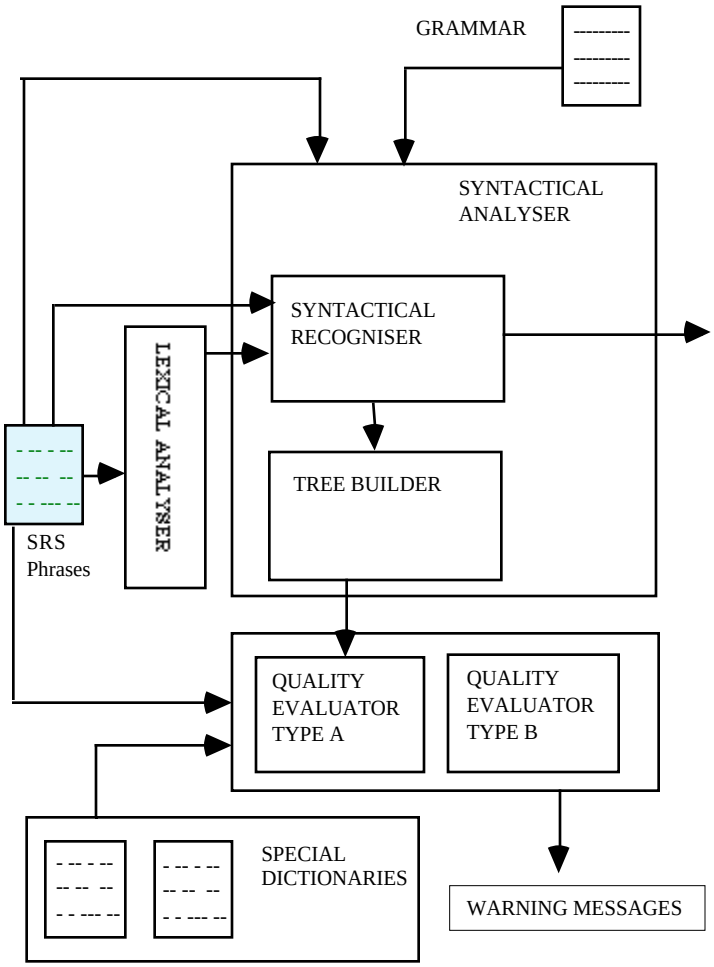
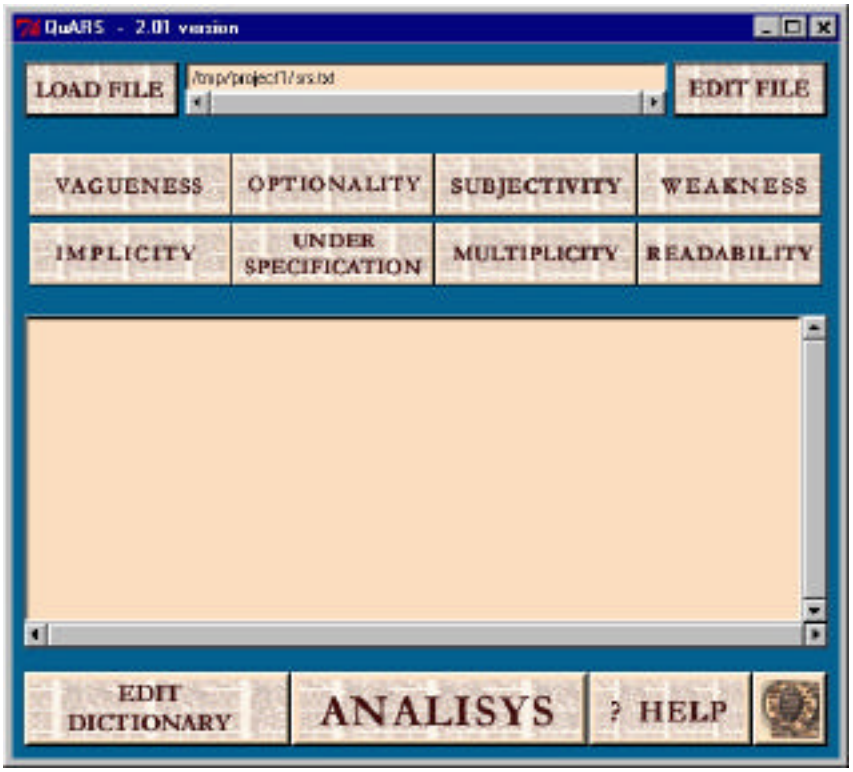


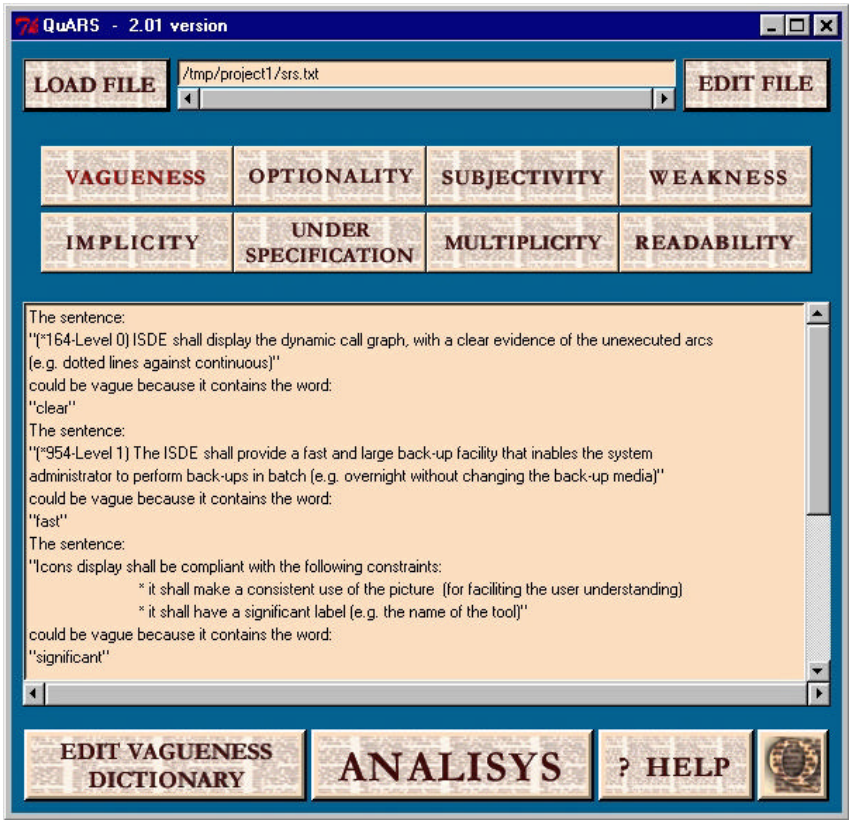
Figure 1

In the rest of this section we present the user interface of QuARS in three phases of the SRS Quality Evaluation:

- The main menu that ask against which Property we need the SRS Evaluation.



- A possible output in case of potential Vagueness problems detected.



5. The NL SRS Quality Evaluation Process

The Quality Model discussed in Section 2 drove the realization of the QuARS prototype, in fact this tool provides the real calculation of the Indicators of the Quality Model on SRS documents.
In this section we discuss how to integrate QuARS within the SRS process and in particular the SRS Evaluation process.
As mentioned in Section 1, the object of the evaluation by means of QuARS is the SRS, that, in the framework of the 'ideal' Requirements Engineering process, are precisely identified as the intermediate level between Requirements Definition and Software Specifications, or they are elsewhere simply a more or less precise definition of the tasks to be performed by the system (or by the Software component of the system). In this second case the SRS document may be the only outcome of the Requirements phase of the software life-cycle and then it may be closer to Requirements Definition or Software Design. In any case, since we aim to perform a linguistic analysis of the SRS sentences, we can, in this context, consider any document containing SRS sentences the object of the Quality Evaluation.

A typical evaluation of SRS written in NL is performed by an expert reviewer (or a team of expert reviewers) who by reading the SRS documents outlights ambiguities, inconsistencies or, in general, inaccuracies. Other actors may be involved in the preparation of these documents, but at evaluation (review) time the reviewer is the key actor.
QuARS can be helpfully included in this scheme (see figure 2). The document on which the Reviewer performs the Evaluation is analysed by QuARS that with its outputs supports the Reviewer in the detection of inaccuracies, ambiguities and linguistic inconsistencies in the document.

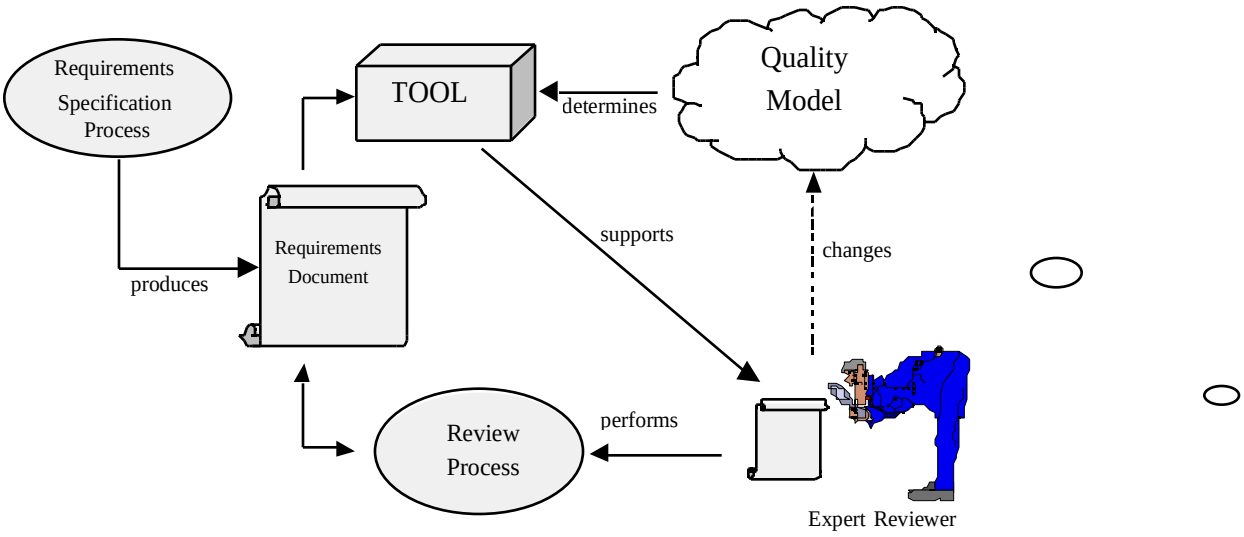


figure 2

The QuARS Tool, for its nature, can be changed in order to make it more adequate to particular context or type of SRS. Then the inclusion of the QuARS tool in the SRS Quality Evaluation process allows the reviewer to change the Quality Model if necessary in order to make it more respondent to the particular SRS under evaluation. Typically, these evolutive changes concern the contents of the special dictionaries that are the basis for the calculation of the Indicators (figure 1). The conceptual changes on the Quality Model can be easily transferred on QuARS for its modularity. Finally, the corrective actions are made on the SRS document and the process can be re-started on the modified document.

The role of the expert reviewer is then crucial in the SRS Quality Evaluation process because, not only he performs the analysis the outputs of QuARS in order to determine if the potential errors pointed out are or not concrete errors to be removed from the SRS document but he has to make also the evaluation environment suitable for the particular SRS under evaluation, by performing evolutive changes to the Quality Model and then to QuARS, if needed.

6. Conclusions and Future Work

This paper reports the work made for realizing an analyser of Software Requirements Specifications (SRS) using a Quality Model for SRS sentences based on linguistic quality properties of the Requirements. In order to automatize the application of the Quality Model to real cases, we have defined, for each Property, a set of Indicators that are tangible aspects of the requirements specifications, can be automatically calculated and provide information on the related Property.
The Quality Model has been experimented on industrial real cases of SRS in order to verify the usefulness and significance of its basis. The results of the experimentation are encouraging, since a significant number of inaccuracies and ambiguities have been detected and pointed out, and the firms that provided us with these real SRS have judged these results very interesting and useful.
Then, on the basis of this experience, we have developed a prototypal tool, named QuARS (Quality Analyzer of Requirements Specifications), that automatically performs the quality evaluation of SRS by measuring the Indicators and pointing out the sentences containing potential inaccuracies and ambiguities. The underlying idea is that avoiding these inaccuracies decreases the risk of errors in the software product due to misunderstandings or incompleteness of the SRS.

Several possible evolutions of this work may be foreseen. Engineering QuARS in order to make it a free tool for the Enterprises interested in this area will be the next step. Another evolutionary direction is to use this Quality Model and this approach to the quality of NL SRS for developing an on-line editor for the SRS. In this case the Quality Model will be used for guiding the production of SRS and not for their ex-post evaluation. Another interesting area of study can be the definition of guidelines for tailoring the Quality Model (and consequently the tool) according to particular Application Areas of the software to be developed or other criteria. These guidelines should be used by the Reviewer for the evolutionary changes to perform on the Quality Model.

7. References

[1] J. Allen: Natural Language Understanding. The Benjamin/Cummings Publishing Company, 1987.

[2] F.Fabbrini, M.Fusani, V.Gervasi, S.Gnesi, S.Ruggieri. On linguistic quality of Natural Language Requirements. In 4th International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'98, Pisa, June 1998.

[3] A. Fantechi, M.Fusani, S.Gnesi, G.Ristori. Expressing properties of software requirements through syntactical rules. Technical Report. IEI-CNR, 1997.

[4] J. Krogstie, O.I. Lindland, G. Sindre. Towards a deeper understanding of quality in requirements engineering. In 7th International CAiSE Conference, vol. 932 of Lecture Notes in Computer Science, pages 82-95, 1995.

[5] G. Lami. Towards an Automatic Quality Evaluation of Natural Language Software Specifications. Technical Report. B4-25-11-99. IEI-CNR, 1999.

[6] F.Lehner. Quality control in software documentation based on measurement of text comprehension and text comprehensibility. Information Processing & Management, vol; 29, No. 5, pp 551-568, 1993.

[7] O.Lindland, G. Sindre, A. Sølvberg. Understanding quality in conceptual modeling. IEEE Software, 11 (2): 42-49, March 1994.

[8] B.Macias, S.G. Pulman. Natural Language processing for requirements specifications. In Redmill and Anderson, *Safety Critical Systems*, pages 57-89. Chapman and Hall, 1993.

[9] M.Mannion, B.Keepence, D.Harper. Using viewpoints to define domain requirements. IEEE Software, January-February 1998, pages 95-102.

[10] K.Matsumura, H.Mizutani, M.Arai. An application of structural modelling to software requirements analysis and design. IEEE Transactions on Software Engineering, vol. SE-13, No. 4, April 1987.

[11] B.Meyer. On formalism in specifications. IEEE Software. January 1985, pages 6-26.

[12] P. Sawyer, I. Sommerville, S.Viller. Capturing the benefits of requirements engineering. IEEE Software, March/April 1999, pages 78-85.

[13] I.Sommerville *Software Engineering*, Fifth Edition, Addison-Wesley, 1995.

[14] W.M.Wilson, L.H. Rosenberg, L.E. Hyatt. Automated quality analysis of Natural Language requirement specifications. PNSQC Conference, October 1996.

QW2000 Paper 8W2

Mr. Steven Watson
(CNET Inc.)

Quality Assurance Challenges in the Internet Industry

BACK TO QW2000 PROGRAM

Key Points

- Personal experience in developing a QA group in an internet startup is a strong basis.
- Which QA methods work and which ones don't work in the internet industry.
- The challenges -- some unique -- which a QA professional will face in the internet industry.

Presentation Abstract

As we begin the new century, it is quite apparent that the QA professional will continue to be faced with many challenges. One area of technology in particular, the internet, has proven to be and will continue to be the one of the most challenging. This talk will focus on the challenges I have encountered, during the implementation of QA processes at both an early stage internet start-up and a later stage internet start-up company. Examined in more detail, I will identify some of the key issues a QA professional will be faced with and why the challenges are amplified when compared to more traditional software development industries. Training, defect tracking, management support, high turnover, experience, documentation control, release management, project management and statistics are just some of the subjects to be discussed. Drawing upon my experience with two successful internet companies and ten years of experience in the medical device industry, methods to overcome these somewhat unique challenges will be presented. What worked, what didn't work.

About the Author

Steven Watson has 15 years experience in Quality Assurance/Test Engineering. He began his career as a test engineer with Underwriters' Laboratories, spent 10 years in QA within the medical device industry, and the last 2-1/2 years developing and managing Quality Assurance departments for two leading internet companies. Preview Travel and CNET. Steve has a B.S.E.E. from the University of the Pacific.

BACK TO QW2000 PROGRAM

QA CHALLENGES IN THE INTERNET INDUSTRY

by
Steven Watson

Let me start off by saying that the target audience for my paper is the QA professional who is thinking about joining the Internet Industry. Perhaps you've been employed in a more traditional software industry and you're looking for something different. Or you've been employed with companies in regulated industries such as the aerospace industry or medical device industry, and you've been hearing about the more relaxed working environments in Internet companies. Whichever industry you're coming from, you've made a decision to join an Internet company, more than likely an Internet start-up. This paper is for you.

Dot com this, dot com that. We really are in the middle of a revolution. Over the past few years, the Internet industry has exploded with a plethora of companies. Living in the SF Bay Area, it's all around you. You can't escape it. So, in October of 1997, I decided it was time for me to depart the "quality critical" medical device industry. After 10 years, I felt it was time for a change. I was also tired of the commute. If you've seen the opening scene in the movie OfficeSpace, you have a good idea of what it's like to commute to work by car around here. For those of you who haven't seen the movie, there's a guy in his car stuck in morning rush hour traffic. Everytime he changes lanes, the lane he was in

previously, starts moving. After several lane changes, he looks on the side of the street and sees an old man with a walker. After a while, he sees the old man walking further and further away. Pretty funny.

Well, going back to my change. Back in late 1997, I was discussing the Internet with several fellow engineers in the last medical device company I worked at. Some of the comments I heard were, "That Internet stuff will never succeed!" or "We're way too biased living here. Most people in this country don't even know how to turn on a computer!" But I was also hearing comments like, "My friend has a friend that is doing some web stuff. All I know is that she's making a ton of money!" Hmmm. Well, a comment like that will catch the attention of most people. So, there I was. I made the decision that, after all my years in QA within a regulated industry. FDA, ISO9000, CE, IEEE, etc.. I was going to jump into the Internet Industry. Didn't have a clue as to what I would be up against, but I knew I wanted to join an Internet company and I didn't want to be the guy in OfficeSpace anymore. I live in San Francisco and "Damn it", I thought, "I'm going to work in San Francisco!" Well, back in late '97, here's where I have to remind everyone that that was quite a while ago in web years, I didn't find a lot of classified ads for QA Managers in Internet companies. Nothing at all like today. But I found one. A start-up! And it was in San Francisco! Sounded good to me. Sent in my resume, got a call and went in for the interview.

Good sounding business model, management team with good credentials and a VERY different working environment from the medical device companies I was used to working at. Hey look! There's two guys putting golf balls down the hall! Was that a dog? Is that a punching bag hanging from the ceiling? Wow, a pool table! Everyone sure looks young! So I asked, during the interview, "What are you expecting from the person who comes on board as QA Manager?" In so many words, the answer was, "We need you to tell us what QA is?" Little did I know what challenges I was in for ahead.

QA Challenge Number One: **Lack of QA Knowledge.** Within the medical device industry, I didn't hear the term Q & A very much. Well, really not at all. After joining my first internet company, I was hearing it on a fairly regular basis. During the interview, it was communicated to me that there was no QA group in the company. Just two guys in Marketing who were in charge of "banging" on the site. "Banging?" After hearing that, I was waiting for them to tell me that their idea of a test tool was a hammer. I never thought I'd "banged" on any systems I've tested. Well, maybe when I worked at Underwriters Laboratories.

So we've got the term "banging" and the misnomer Q&A. Hmmm. 2 years and 8 months later. Guess what I've discovered? Within the Internet Industry there is very little knowledge of what Quality Assurance really is and what a QA group does. Due to this lack of "QA knowledge",

you'll get a lot of blank stares if you start using terms like Unit testing, Integration testing, System testing or Regression testing. "Uh, dude, we just want you guys to "bang" on it and let us know what breaks." Or you'll hear some business or development managers in a meeting using the terms incorrectly, after they heard you mention them in a previous meeting. Why is there such a lack of "QA knowledge"?

In Internet companies, especially startups, you've got people from all over the place. And a majority of them are in their 20's or early 30's. All jumping onto the pre-IPO Internet bandwagon. And all hoping to get rich in the process! But guess what? The supply can't fill the demand. What you end up with are companies filled with some people that have never even been exposed to a traditional software development process, and therefore, don't have a clue as to what the QA test process is all about.

So how do you implement QA processes in an industry where QA is called Q&A? And they just want you to "bang" on the site. What's my answer? Selectively and with training! What? Yes, training. A QA professional joining an internet company will constantly be training the company's employees. You will need to evangelize the QA process within the organization. If you've had experience implementing ISO9000 in any of your past companies, you know that training is critical to obtaining your certification. Is there time to train? Ha! In the Internet Industry, things move as fast as the guy in the OfficeSpace movie was moving slow.

But you'd better find the time. Take any opportunity to train. Going out to lunch with some co-workers? Going out for a beer after work? Seize the moment. Of course if you have management's support, you'll be in good shape.

But that's Challenge Number Two: **Management Support.**
One of the most critical, if not THE most critical, needs of a QA professional forming a group and implementing QA processes in an Internet startup. If you don't have management support, then why are you even working at the company, right? Well, guess what. It happens. It happens because of the first challenge, Lack of QA Knowledge. You can ask all the right questions, during your interview, and hear all the right answers and you may still hear silence when you ask for support. So why do internet start-ups hire QA Managers? Well, to be fair, some Internet companies have management teams that are sort of familiar with QA processes, but some just have an investor on the board who wants a QA group in place. So it's possible that some managers don't know anything about QA. And as soon as they hear wind from a business manager that QA delayed the schedule, you'll be in trouble. But then we're back to training again. It's up to you to train the management team on the QA process, and if it sinks in, you're in better shape. They'll understand how unrealistic schedules contribute to a low quality release, which results in dissatisfied users. And they'll feel more

comfortable when they hear the term "Regression testing". Hopefully, then, management will understand the need to train the rest of the company on QA processes. But don't think your training job will be over though.

Because of Challenge Number Three: **High Turnover**. With so many Internet companies out there. It's extremely difficult for Internet companies to retain employees these days. Technical and business types alike, right now, can literally walk across the street from their existing company and land a new job over their lunch hour. What? Your company's stock didn't split in the last year? Your company hasn't IPO'd yet. Dude, your laggin! Or you'll receive emails like, "It's been a great two years! Bye everyone! I'm off to my next pre-IPO dot com!" What this means is that you'll be continuously training new "QA deficient" employees. If you're in a later stage Internet company, you may have the luxury of having a formal Training department in place to help you. Either way, a formalized and appropriately scheduled QA training class for all new hires, mandated by upper management, is essential to effectively communicate the QA process throughout a company with a high turnover rate. Did I say communicate?

Because that's Challenge Number Four: **Communication**. Now here is a BIG challenge for QA. You know the cartoon that shows a bunch of

drawings of a tire swing, along with captions under each drawing? The first one shows a regular tire swing attached normally to a tree branch. The caption under this drawing reads, "What the customer wanted". Then there's one with a caption, "What Marketing wanted." Then one that reads, "What Engineering designed", and so on. Well, I saw that cartoon about 10 years ago. Boy does it apply to the Internet Industry. In such a rapid paced industry, you'd think that most Internet companies would have some system to aid in communication from one group to another. Communication is so important! Well, the reality is they've got a system, but it doesn't aid in maintaining communication between groups. In fact, it's probably the main cause of inadequate communication. It's email. Email is only effective if all the appropriate people on a project team receive copies. For example, if a Product Manager on a project, doesn't know what QA is or does, the manager won't include QA on any important emails related to a project. The current Internet company I'm working for uses mailing lists to aid in maintaining communication between project team members. This helps. But email really can't take the place of verbal communication. If an initial email is unclear, it can result in a long thread of replies from people asking questions on the content in the first email. Then you'll see replies to replies to replies. The main thing a QA professional will need to do is maintain communication by taking your own initiative to meet with other project team members. Tell them how important it is for you

to be kept in the loop, and badger them if they don't. It's the squeaky wheel gets the grease technique and in my experience, it's you're best alternative in this fast paced industry. Tight schedules mean people are extremely busy. If they don't know that you need to be communicated to, you won't be. Let them know that communication is required!

Which brings us to Challenge Number Five: **Requirements**. Right before I left the medical device industry, the FDA was communicating the importance of proper design controls. How patient injuries and deaths caused by a malfunctioning medical device were continuously being traced back to the device manufacturer's lack of proper design controls. For software controlled devices, they were enforcing the documentation of a Software Requirements Specification and Software Design Specification. In the Internet Industry it is rare to be involved in a project where a Requirements document exists. You'll hear statements like, "The requirements on my project are changing on a daily basis. Why should they be documented?" And it's true. They can change on a daily basis or even more frequently. Why? I think a big reason is that a limited number of people are involved at the early stages of defining a project's requirements. Because someone who should have been involved much earlier on the project, wasn't, the requirements change, after that person provides their input at a later stage in the project's schedule. Then someone else who was left out wants to. And so on. At

the end of the project, changes to requirements could have been greatly reduced, if a requirements gathering or brainstorming meeting was held early on with all the appropriate people in attendance. So be prepared to be involved in many projects without a Requirements document. Or any other semi-formal documentation for that matter. You may need to gather and piece together emails that you weren't copied on. You'll need to speak with the developer or developers involved to get their understanding of the requirements. My suggestion to overcome this challenge is to let the business and project managers know how much longer it takes to prepare a test plan without clearly documented requirements. As soon as you point out how time in the schedule can be saved, they'll start listening. IF, and that's a big IF, you are lucky enough to be involved in a project that has a Requirements document. More than likely you'll be faced with the next challenge. That being the issue of testability. I could go on and on with examples, but suffice it to say that I have read very few requirements in a Requirements specification that are written in a testable manner. Phrases like, "The page should load fast and be user friendly", come to mind. So let's say you're really lucky and find an Internet company, which does document Requirements AND they're stated in a testable manner. The next question to ask is, "Are revisions to the document controlled".

Aha! Challenge Number Six: **Documentation Control**. Again, I have to draw upon my experience in the medical device industry and with ISO9000 standards.

Documentation control processes are so critical in the medical device industry and truly critical if a company wants to obtain ISO9000 certification. Well, in my experience so far in the Internet Industry, almost every document I've read related to a project I was testing, was not maintained throughout the life of the project. A first version of the document would be created, but it was never updated. Or if it was updated, not on a regular enough basis to be of any help to the "customers" of the document. A Requirements specification is the perfect example. A first draft may be created, but by the time coding is started. The document may be totally inaccurate in it's representation of the latest project requirements. By the time QA is ready to test, guess what, the document may as well not exist. So how to overcome this challenge? Well, knowing what you'll be up against will help. But educating the document originators on the importance of maintaining version control on their documentation is key. Again, the fast pace of this industry, is the main cause here. I've heard both Business Managers and Developers complain that they don't have time to maintain their documentation. At CNET, we are slowly moving toward the direction of having technical writers work along side the Business Managers and Developers. The thinking is that technical writers are the experts at creating and

maintaining documentation. They will be responsible for translating verbally communicated requirements from the Business Manager into a Requirements document. The same would go for Design requirements communicated by a developer or technical lead. At this point, I can't say that I've actually seen this approach work. It's in the early stages of implementation. We'll see how it goes. Hopefully there won't be any bugs.

Challenge Number Seven: **Bug Reporting/Tracking.** One of the more common tasks a QA professional will be faced with in most any software related industry is setting up and maintaining a bug tracking system. But in the Internet Industry, one of the challenges is getting developers and others to just use the system. From what I've experienced so far, don't waste your time setting up a non-web based bug tracking system. Most everyone in this industry is pro-web. They work for a web company, so they want to use web-based applications. You also won't be faced with platform dependency issues. You'll eliminate excuses like, "It doesn't work on my Sparc station so I don't use it." Then there's that evil email system again. Guess what? That's how bugs are most commonly being reported prior to the implementation of a bug tracking system. It's faster than logging a bug into the bug tracking system and more people are familiar with using email than using the bug tracking system. How to overcome this challenge? Well, hopefully, all the

managers are aware of the need for a bug tracking system to be implemented in the company. And, hopefully, they each had a say in the system selected for implementation. If not, you'll undoubtedly experience some resistance in it's usage. But if you've conquered Challenge Number One and Challenge Number Two, you're in good shape. I know! Perhaps you could try automating the process!

Challenge Number Eight: **Automated Testing.** Don't know when I'll stop hearing managers say. "Wow, we can automate all the front end GUI testing and we'll be able to just concentrate on testing the back end stuff! This is challenge number eight. Getting managers to understand the limitations of automating the testing of a website. Especially if your company's website contains a lot of content which is constantly changing. The automated test tools out there are getting better, but they still have their fair share of bugs. Which is the last thing you want to deal with. Using an automated test tool to test for bugs on your company's website, which contains bugs itself! Communicate to these pro-automation managers the limitations of automation testing. Which test cases make the most sense to automate. But bring them back down to earth. Functional regression test tools and load test tools will be the common tools you hear about in the Internet Industry. But in the early stage start-ups, you'll see System Admins scrambling to set up another two or three NT servers. "Forget load testing!" is what you may hear from

the CTO. We're growing way too fast. Just keep adding boxes and pray that the air conditioning system in the server room doesn't break down.

Challenge Number Nine: **Release Management.** Guess what? A clearly defined release management process doesn't exist in an Internet start-up and I can say that even some later stage Internet companies are still struggling with this issue. First is the question of who is responsible for release management. Once the environments are set up: Development, QA, Staging and Production. Which can be a challenge on it's own. Then there is the deployment of code between these environments. Who deploys the code from Development to QA? How should the process work? Then from QA to Staging. Who deploys and what's the process? But actually, I'm describing a later stage Internet company. Walking into a 30 person start-up, you can expect no defined process whatsoever for code deployment. And even worse, you may see developers making changes to the code, directly in the Production environment. An early stage start-up, that's a bit more together, will at least have separate development and production environments. Here you'll experience the fun of testing within the development environment, at least until a separate QA environment is established. Wonder at the number of bugs logged against a software release that were actually not bugs with the software under test, but were bugs caused by developers making changes to the environment your testing in. So back to the companies

who have all the proper environments set up. You'll still deal with the lack of a defined deployment process. And believe me, proper deployment from the Development environment to the QA environment will be the major challenge. So my advice is to work together with the Development managers, Operations managers, etc. to get a deployment process defined ASAP. I've seen some companies recently, which are lumping Release Management responsibilities into QA. But I've heard the phrases "I'm burned out" and "I gave up my life" from QA Managers who are also responsible for Release Management. The later stage Internet companies may form a separate Release Management team composed of one or two Release Engineers and possibly a Release Manager. But in a small start-up, be ready to be involved to some extent with the release process.

Last, but certainly not least, is Challenge Number Ten: **The Rapid Pace**. I've been eluding to this challenge so far. Really, the reason for many of the previous challenges I've just discussed, is the rapid pace of development within this industry. Where the average time for a project's development cycle in a traditional software company might take one to several months, in an Internet company, it's common to release code on a daily basis. In this industry, you don't have to wait for tech pubs to complete the User's Manual or go through the process of packaging design. It's plan, code, test and launch! The BIG misconception is that

any attempts to add order to a process, in this rapid development environment, will delay the schedule. In my opinion, the fact that this industry's development pace is so rapid, is all the more reason to review the processes involved and increase efficiency by refining the processes appropriately. My advice is to communicate the need for efficient processes to the other managers. Let them know that you are aware of the need for speed and that it's everyone's job to agree on the processes, which will ensure the highest level of quality in this rapid paced environment. Communicate the importance of defining and agreeing upon these processes. How time spent now, will save immeasurable amounts of time on every future project in the company. Lastly, make sure they are aware that process adjustments will be needed as changes occur within the company. In a regulated industry, formal audits of established processes are conducted on a regular basis. Not so in the Internet Industry, but just knowing this can help. If a process is in need of revision, call a meeting with the appropriate managers, explain why and continue to speed along.

So now that I've listed some of the challenges a QA professional will be faced with in the Internet Industry, let me say this. In the midst of all the chaos you'll find that the general feel of the environment in these companies, is exciting. It's fast paced and always changing. There are no real rules established yet. It's still very much the wild, wild, west.

Remember that this industry is still in it's infancy stage. With time, we should hear less people calling QA, Q&A, and hopefully less people asking us to "just bang on the site". Also, competition in this industry is building. With competition comes increased awareness of the importance of quality. Like the Big 3 US auto manufacturers who got caught by the quality conscious Japanese auto manufacturers, back in the 1970's, there are internet companies out there concentrating on doing it right the first time. Those companies will rise and the industry will mature.

But for now, it's full speed ahead and hold on to your test plans. Whoever gets to market first is perceived as the winner. The Internet Industry, like the old man with the walker in OfficeSpace, is moving much faster than the more traditional software industries, which comparatively appear to be the ones in the rush hour traffic. Only, at this stage, I'd replace that old man with an infant. An infant who is crawling very fast. Let's just hope that this infant doesn't skin it's knees too badly on it's way to learning how to walk.

QW2000 Paper 8M2

Mr. Marc Zasada
(VeriTest)

What Does "Application Certification" Mean in the Software Industry?

BACK TO QW2000 PROGRAM

Key Points

- Industry-wide Application and ASP Certification programs are growing in popularity
- The history of Certification programs
- What Certification programs mean
- Where Certification is headed

Presentation Abstract

Market-leading companies have long seen the value in providing a logo program for the third-party products that integrate properly their products. Standards for these programs have varied wildly, and customers have not always understood them. As the software industry matures, standards for software and hardware are more critical than ever. End users are demanding higher levels of reliability and manageability. As a result, there is a trend to upgrade logo programs to more rigorous certification programs, and there is even a trend to "Certify" ASPs and ISPs.

An overview of "Certification" within the industry will be presented, along with a focus on Microsoft's efforts over the years. Created by Microsoft and VeriTest in 1994, the "Designed for Windows 95" logo test was comprised of a relatively simply array of tests performed against a small set of requirements devised by the Windows logo team at Microsoft. With the advent of Windows NT 4.0 came the "Designed for Windows NT and Windows 95" logo program in which the requirements, written by the Microsoft Windows team with input and edits from the logo team at VeriTest, became more stringent. This evolved into the "Designed for Windows NT and Windows 98" logo test, and the testing process became even more in depth and demanding.

To license the new "Certified for Windows 2000" Logo, products must undergo extensive functionality testing in addition to compatibility testing. Applications qualifying for this logo must be shown to provide the highest level of reliability, security, and manageability available. Compliance with the Application Specification for Microsoft Windows 2000 for desktop applications is strictly enforced and the Windows logo team at Microsoft closely monitors the needs of developers and consumers before allowing any variance or making changes to the requirements.

About the Author

For 9 years, Marc Zasada has been the Vice President of VeriTest, one of the world's most respected independent labs. Since 1994, Marc has helped craft the global software certification strategies for the Microsoft Windows, Windows CE and BackOffice logo programs, and is now working closely with Microsoft in rolling out the "Windows 2000 Application Spec" testing program at VeriTest worldwide. VeriTest was acquired by Lionbridge in 1999.

BACK TO QW2000 PROGRAM

What Does It Mean to "Certify" a Software Product in the Commercial Marketplace?

Marc Zasada
V.P., Certification Programs
VeriTest, a Service of Lionbridge

VeriTest.

What Is Software "Certification"?

- Introduction
- Goals
- Role of the Lab
- Levels of Certification
- Methods & Approaches
- Case Study: Certified for Windows 2000
- The Global E-Certification Challenge

VeriTest.

Introduction

- Who is VeriTest?
- Marketing or Testing?
- A Little History
  - Logos, logos, everywhere
  - Failure Rates
  - Worth the ink it's printed with?
  - Microsoft
- Redefining Certification
- The Current Situation
  - Open Source
  - ASPs

VeriTest.

VeriTest Certification Programs

Designed for Microsoft® Windows NT® Windows® 98

Powered by Microsoft® Windows® CE

ORACLE e-certified warehouse

Microsoft® OFFICE Compatible

Certified For Microsoft® Windows®

Certified For Microsoft® Windows®

Built with ObjectARX AutoCAD®

TERMINAL SERVER TESTING PROGRAM

Microsoft ® Windows NT® Server 4.0 Terminal Server Edition

CERTIFIED FOR PATROL PATROL BMC SOFTWARE

Designed for Microsoft® BackOffice®

Featuring Microsoft® Visual Basic® for Applications

VeriTest.

Certification Program Goals

Goals for Sponsor

- **Delivers brand extension**
- **Defines Partner "solar system"**
- **Provides technical direction to Partners & End-Users**
- **Promises consistent quality to End-Users**

Goals for ISVs

- **Gating for "Partnership" Benefits**
- **End-User Recognition**
- Internal discipline

VeriTest.

Certification Goals

End-User Benefits

- Follows Standards
- Has "joined the technical wagon train" with Sponsor
- "Stamp of Approval"
- Guarantees support for certain technologies or features

VeriTest.

Programmatic Goals

- Operate fairly and independently.
- Create consistent, auditable, and repeatable results.
- Track exemptions and changes.
- Anticipate technology.

VeriTest.

Role of the independent Lab

- "Cooperative Competition" in the marketplace
- "Middleman" role of the lab
- Global consistency

VeriTest.

Levels of Testing

Scalability, Stress, Boundary
Functionality & Stability
Verification with Standards
Demo of Compat & Compliance
Self-Certification with Audit
Self-Certification with Standards

VeriTest.

Approaches

- Creating Compliance Criteria
  - Livability
  - Viability
  - Testability
  - Survivability

VeriTest.

Approaches

- Creating a Test Plan
  - Definitions
  - Convenience
  - Public Distribution
  - Auditability

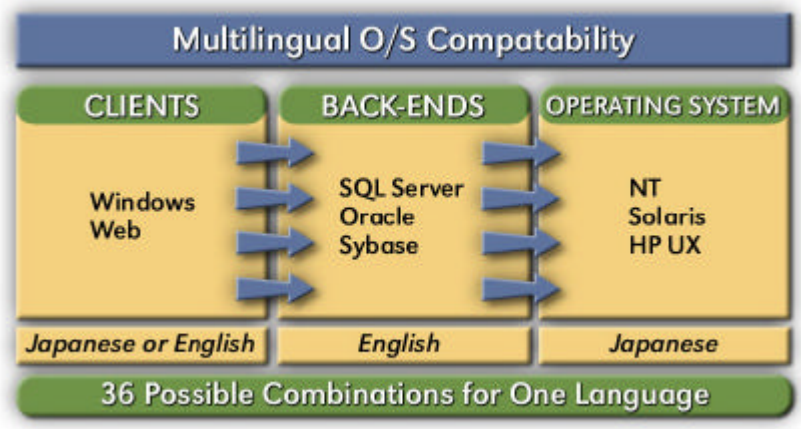
VeriTest.

Case Study: Certified for Windows 2000

- Defining the Criteria
- Discussions with Customers & Analysts
- Creating the Test Plan
- Automated Test Tools

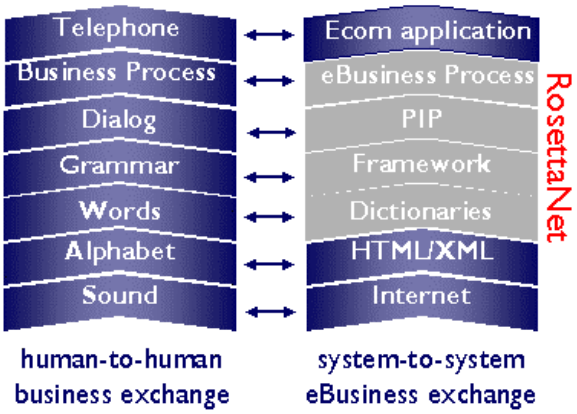
VeriTest.

The Global Certification Challenge



VeriTest.

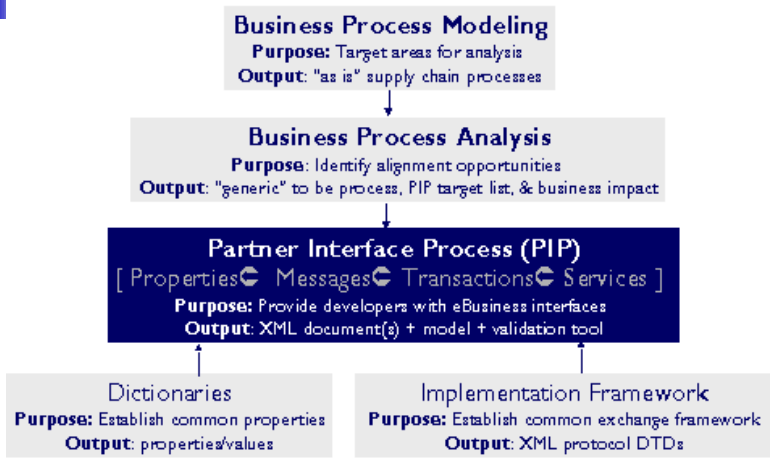
The "E-Certification" Challenge



Source: Rosettanet.org

VeriTest.

The "E-Certification" Challenge



Source: Rosettanet.org

VeriTest.

The "E-Certification" Challenge

- The Needs of ASPs
  - Scalability
  - Provisioning
  - Multi-tier
- Open-Source

VeriTest.

Conclusion

- Standards, standards, standards

VeriTest.

QW2000 Panel 8P
Thursday, 1 June 2000
5:00 - 6:30?

Mr. Nick Borelli
(Microsoft Corporation)

Ask The Quality Experts!

Stump the Quality Experts If You Can!
QW2000 Advisory Board Members Will Answer All!

Post Your Questions LIVE on the Web!
(Live question collection and voting provided by Microsoft Corporation)

BACK TO QW2000 PROGRAM

How The *Ask The Quality Experts* Panel Works

This special QW2000 panel session works **interactively** with you to get your key questions answered! If you have a burning question about any aspect of Software or Internet Quality, click on the Ask The Quality Experts! page.

You'll see the current set of questions posed to the Panel Of Experts, rank ordered based on the number of votes each question has received.

Are The Questions Moderated?

Yes, the questions posted are moderated. From time to time the Ask The Quality Experts! Panel Moderator, Nick Borelli, will review the current set of questions. He'll remove off-topic questions, consolidate obvious duplicates, and make other necessary corrections. If there are high-scoring questions that seem to be outside the range of the experts currently on the panel we will **add more experts to the panel**.

About The Panelists

The panelists are chosen from among the International Quality Week Advisory Board. You can see who the panelists are on the current Ask The Quality Experts! page.

How Often Can I Vote?

You can vote as often as you like, but, **please** we ask that you only vote for your favorite question(s). P.S. QW2000 will have Web workstations available where you can vote on-site.

How Will I Get Answers to My Question?

Yes. During this special session the Advisory Board Experts will answer the top ranking questions -- using the data from the web as of Noon on Thursday 1 June 2000. QW2000 will have Web workstations available where you can vote on-site before then.

Brief summaries of the answers will be posted on the Web shortly after the conference is over.

About the Moderator

Nick Borelli is currently a Group Test Manager at Microsoft Corporation and is responsible for the World-Wide releases of the award-winning application, Microsoft Word.

Nick has over 15 years experience in both Software Testing and Development and has worked in both small start-ups such as Pensoft, Go and EO, as well as working at Triad Systems, Apple and Software Publishing Corporation.

BACK TO QW2000 PROGRAM

QW2000 Panel Session 9P

Mr. Brian Lawrence
(Coyote Valley Software)

How Can I Tell When My Project's in Trouble?

Panelists:
Maureen A. O'Hara
Esther Derby
Johanna Rothman

BACK TO QW2000 PROGRAM

Key Points

- Identifying clues that projects are in trouble
- Interventions once risk is identified
- experienced views of turning around software projects

Presentation Abstract

Many software projects fail. Many might not fail if we could only recognize the signs of trouble in time.

This is a 90 minute panel proposal to discuss real circumstances where projects have started to go south, how the panelists recognized the signs, and what they did. I've chosen experienced and thoughtful participants who have seen many software projects, and are experts at intervention.

Each panelist will take a few minutes to present their top three techniques for identifying project trouble, and how they might intervene. Then the chairmain will open the rest of the session to audience questions.

About the Panel Chair and Panelists

Panel Chair: Brian Lawrence - Coyote Valley Software

Brian has moderated panels in a number of conferences, including several past Quality Weeks. He has served as a program chair for the SEPG'97 Conference as well as the IEEE Computer Society's 1998 International Conference on Requirements Engineering. In his consulting practice, Brian teaches and facilitates requirements analysis, peer reviews, project planning, risk management, life cycles, and design specification techniques. Brian is the technical editor of Software Testing and Quality Engineering Magazine and serves on the editorial board of IEEE Software. Brian Lawrence Home Page

Panelist: Johanna Rothman (Rothman Consulting) observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people. Johanna is the founder and principal of Rothman Consulting Group, Inc., and is ASQ certified as a Quality Auditor and Software Quality Engineer.

Panelist: Esther Derby (Esther Derby Associates) has over twenty years experience in software development. She's been a programmer, systems manager, project manager, and internal consultant. She currently runs her own consulting firm, Esther Derby Associates, Inc., in Minneapolis, Minnesota. Esther works with people to increase their effectiveness in understanding and managing complex systems -- like software development organizations and software development projects.

Panelist: Maureen A. O'Hara has an extensive and diverse background in software project management. Through her 22 years in the software industry, and most recent seven years at Microsoft, Maureen has worked in software project development methodologies including team management and crisis management. Recently at Microsoft, she founded a group dedicated to improving the software development process and overall product quality.

BACK TO QW2000 PROGRAM

Quality Week 2000 Panel Proposal

"How Can I Tell When My Project's In Trouble."

Many software projects fail. Many might not fail if we could only recognize the signs of trouble in time.

This is a 90 minute panel proposal to discuss real circumstances where projects have started to go south, how the panelists recognized the signs, and what they did. I've chosen experienced and thoughtful participants who have seen many software projects, and are experts at intervention.

Each panelist will take a few minutes to present their top three techniques for identifying project trouble, and how they might intervene. Then I will open the rest of the session to audience questions.

Panel Moderator

Brian Lawrence – Coyote Valley Software

Brian has moderated panels in a number of conferences, including several past Quality Weeks. He has served as a program chair for the SEPG'97 Conference as well as the IEEE Computer Society's 1998 International Conference on Requirements Engineering. In his consulting practice, Brian teaches and facilitates requirements analysis, peer reviews, project planning, risk management, life cycles, and design specification techniques. Brian is the technical editor of *Software Testing and Quality Engineering Magazine* and serves on the editorial board of *IEEE Software*.

Contact info: http://www.coyotevalley.com/
brian@coyotevalley.com
(408) 578-9661

Panelist

Johanna Rothman – Rothman Consulting, Inc.

Johanna Rothman observes and consults on managing high technology product development. She works with her clients to find the leverage points that will increase their effectiveness as organizations and as managers, helping them ship the right product at the right time, and recruit and retain the best people. Johanna is the founder and principal of Rothman Consulting Group, Inc., and is ASQ certified as a Quality Auditor and Software Quality Engineer.

Panelist

Esther Derby – Esther Derby Associates, Inc.

Esther Derby has over twenty years experience in software development. She's been a programmer, systems manager, project manager, and internal consultant. She currently runs her own consulting firm, Esther Derby Associates, Inc., in Minneapolis, Minnesota. Esther works with people to increase their effectiveness in understanding and managing complex systems -- like software development organizations and software development projects.

Panelist

Maureen A. O'Hara

Maureen O'Hara has an extensive and diverse background in software project management. Through her 22 years in the software industry, and most recent seven years at Microsoft, Maureen has worked in software project development methodologies including team management and crisis management. Recently at Microsoft, she founded a group dedicated to improving the software development process and overall product quality.

SUBMISSION INFORMATION:

Target Audience is:

- Managerial

Please indicate the level:

- Could be Beginner, Intermediate, or Advanced, depending on audience questions

Also, indicate if the basis of your paper is:

- Opinions/Perspectives based on Work Experience

Give three bullets describing your presentation

- Identifying clues that projects are in trouble
- Interventions once risk is identified
- Experienced views of turning around software projects

Lessons to be learned from this presentation

The audience will have the chance to ask questions of these three experts on software project management, which they may take back and use in their own circumstances.

How Can I Tell When My Project's in Trouble?

© 2000 Johanna Rothman

I look for quantitative and qualitative cues for when a project is in trouble.

Quantitative Clues

I like to take some specific measurements to see what's going on:

- Defect find and close rates by week, normalized against level of testing effort. Are the testers keeping up with the developers? Are the developers keeping up with the testers?
- Defects found per activity vs. total defects found. Where am I finding the most defects? Could I find them in other ways, with other techniques
- Schedule estimates vs. schedule achieved, by major milestone. Am I keeping up with what I thought I could achieve?
- People-hours on the project, planned vs. actual by week or month. Do I have the staff to do the work?
- Major and minor changes to requirements by week over the project. How often are we changing what we think this project is about?
- Number of tests planned, run, passed by week. Are the testers making progress, or is the work ever-expanding,

Qualitative Clues

It might be easier if all we had to do was measure things and know about our projects. Measuring things isn't adequate; looking at what people are doing is as important as the hard data.

One clue that something is wrong on a project is when the people start coming in later and leave even later. When people come in a little early and leave a little late, they're excited about their work. When people come in later and leave even later, they're tired. They're trying to make up schedule overruns with overtime. This generally means they are making more mistakes than they would normally.

Another cue to a project in trouble is to see what's not talked about. Are there issues that run into walls of silence? Anything that's not talked about is generally the thing that's a disaster.

Sometimes, people know what's wrong on a project, but they don't feel as if they have permission to tell anyone.

How Do I Know a Project is in Trouble?

© 2000 Esther Derby

Basic measurement data is a good place to start looking for signs of trouble -- when you can get it. Number of defects opened and closed, planned to actual staff hours, milestones met, number of requirements changes, test cases planned, executed, and passed can provide useful clues.

But I see a lot of projects that don't collect much data. That's generally enough to tell me there's a problem -- with the project and with the organizational system.

I look at management and the organizational context:

- Does the organization have an established project management practice?
- Does the organization have an overall project agenda and a way to track that agenda?
- Does the organization have the process and cultural capability to do the type of project they're undertaking?
- Is the reward structure designed to support building good quality software?

On the project level I look for a reasonable set of processes to track progress and control change:

- Are there clear, written, testable requirements?
- Is there a mechanism to control change?
- Is there a plan?
- Is status visible?
- Is there a way to track and problems (issues, risks, defects)?

These don't have to be fancy or sophisticated, paper will do when an organization is starting to bring their project practices under control.

And I look at the relationships between people and groups:

- What are the relationships like?
- Do the business and development teams get along?
- Is there a climate of respect and teamwork?
- Is there a climate of blame?

Sometimes I can tell there's a problem without looking at any of these things. Sometimes the way the project manager talks or react tells the story: "We don't have time to do reviews!" or "There's no way you can stop this project!"

Sometimes it's the unspoken climate: the humor is dark, developers and testers are working lots of overtime, people are leaving the project.

All of these seem pretty obvious. And they can be hard to see when you are part of the system. They can be even harder to talk about when you are in a climate that kills the messenger or refuses to hear bad news.

How to Tell When Your Project Is Going South

Copyright © 2000, Maureen A. O'Hara

"Management" considers a project failing if the project is behind schedule; and they sometimes even consider being over-budget, or not delivering what was expected in terms of content or quality failing. The signs of pending failure are not elusive or mysterious, and if you know where to look, they are apparent before management reads your progress report. You can monitor specific areas to identify problems early. The cost of the additional monitoring, including educating the participants in what to monitor, is far more cost efficient then to remedy problems. The types of problems found in projects are usually endemic to an organization. Your management's lack of commitment to process quality is a telltale sign that there are projects in trouble.

The core areas to watch are:

- **Documentation** – if your project does not have the "standard" set of documents that are not disseminated to the team and are not kept current, send up a red flag. Signs of documentation woes include discontinuous progress, staffing/resource needs in flux, and continuous changes in end-user product documentation. Other warning signs include team members do not having the "big-picture" or do not understanding where their work fits into the end product or the implementation plan. The band-aid solution of doing a one-time documentation update is not the solution; you should put in place procedures for creating, reviewing, up dating, and disseminating documentation.

- **Project Life Cycle** – is the team using a "reasonable" method or process for implementing similar projects? Warning signs are dusty copies of general industry methods, and explanations that all projects are unique and methods are not needed, as the team is small and experienced. Beware when team members say that all projects always have the same problems. The fix is to take the painful step to stop the project, develop a plan, and then to use it!

- **Team and team management** – this includes issues of staffing, training, communication, individual effectiveness, and morale. Projects with issues in this area see people working long hours for too long a time, as well as people working minimal hours. Warning signs are team members not speaking about their work, peers or management with any sense of belonging or pride. If you see too little communication or interaction among team members, be it formal or informal, this is a sign that there are fundamental problems. Don't underestimate the impact of this issue -- a "together" team can make a troubled project succeed.

- **Quality** – project quality problems are the easiest to find -- if you look. Some of the simplest metrics are the best tell-tale signs. Large pieces of the project not fitting together, or too much retrofitting are also signs of poor QA during the design phase. Usually if there is "quality smoke", there is fire. Remedies include a QA plan and commitment to the plan that starts from senior management and goes down to the "little fishes". This commitment includes funding with people and other resources, education, and commitment not ship a low quality product. This usually starts with an appreciation that quality is more than just testing, and bad quality always costs more in the end.

- Technology – surprisingly enough, projects infrequently fail due to technology problems. Warning signs are: if only one or two team members grasp key technical areas, or if there is little or no detail design for areas where new or "to be developed" technology is to be used. Problems here are usually a sign of delivery promises made by people who do not understand the technical hurdles, or management not understanding how to deal with technical unknowns. Remedies start with telling the truth about the problems; first within the team and then to management.

QW2000 Paper 9T1

Mr. Sam Guckenheimer
(Rational Software Corporation)

Enabling Testable Architectures with UML

BACK TO QW2000 PROGRAM

Key Points

- Overview of the industry-standard Unified Modeling Language (UML) for testers
- Examples of using UML to capture test design as part of system design and to provide access methods for testing the AUT
- Guidelines for an architectural process around UML to enable testing from the beginning of the software development lifecycle

Abstract...

Architectural process and patterns for testability will have a significant impact on software quality. Yet testing technology has largely ignored the use of architectural artifacts as a source of test design and instead focused on processes of deriving test requirements or testability solely from the as-built system. This is backwards, as it is architectural process and standards that are the single greatest factor determining the testability of software, whether at component, integration or system levels. The Unified Modeling Language (UML) has become the industry standard for capturing software architectures and elaborating system design. This talk provides an overview of the use of the UML to capture test design and specify accessbility for testing the AUT.

Author Bio...

Sam Guckenheimer is the senior director of technology for Rational's Automated Testing products. In this role, he is responsible for the product vision, strategy and implementation of Rational's software testing products and the formulation of architectural standards for testability. He joined the engineering team at SQA Inc. in 1995 as director of technology integration, assumed responsible for marketing of test products when SQA merged with Rational in early 1997, and assumed his current position in autumn 1999.

Sam has held several marketing, engineering, and general management positions in US and European software companies over the last eighteen years. At Rational, Sam has spearheaded the integration of the load testing products, the development of Java and Web server testing technologies, the internationalization of SQA Suite, and the introduction of OEM versions of these products. Prior to joining SQA, he spent six years with Softbridge, Inc., ending his time there as managing director of Softbridge Capital Markets, a subsidiary based in London, England. A Phi Beta Kappa graduate of

Harvard University, he is a frequent speaker on software development and application topics at industry conferences and seminars, and has spoken as a guest lecturer at the management schools of MIT, Harvard, and Yale.

BACK TO QW2000 PROGRAM

Rational®
the e-development company™

Enabling Testable Architectures with UML

Sam Guckenheimer
samg@rational.com

Agenda

- What is Architecture?
- What is UML?
- What is a testable architecture?
- What does UML provide for testing?
- What process does this imply?
- Some predictions

Agenda

- **What is Architecture?**
- What is UML?
- What is a testable architecture?
- What does UML provide for testing?
- What process does this imply?
- Some predictions

Rational
the e-development company™

What is Architecture? (1/2)

- Easy to plan
- Easy to build
- Easy to test

Rational
the e-development company™

What is Architecture? (2/2)

- Require architecture and process
- Testing is different
- Can't reverse engineer as-built structure to derive intent

Rational
the e-development company™

Agenda

- What is Architecture?
- **What is UML?**
- What is a testable architecture?
- What does UML provide for testing?
- What process does this imply?
- Some predictions

Rational
the e-development company™

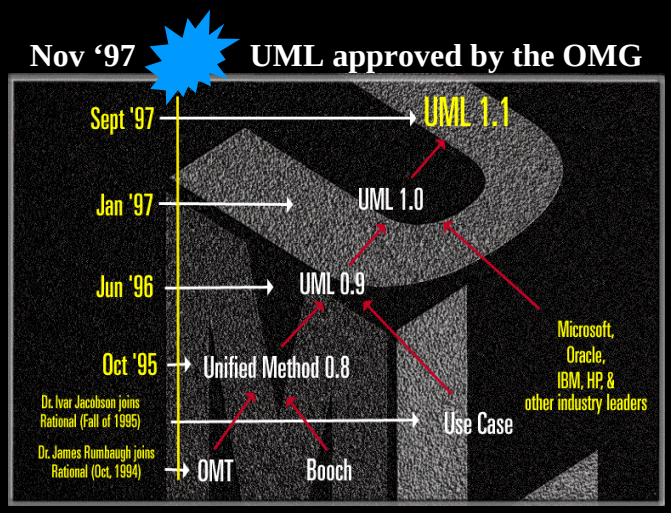
Unified Modeling Language

- The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system
- The UML combines the best of the best from
  - Data Modeling concepts (Entity Relationship Diagrams)
  - Business Modeling (work flow)
  - Object Modeling
  - Component Modeling
- OMG Standard

UNIFIED MODELING LANGUAGE

Rational® the e-development company™

History of the UML

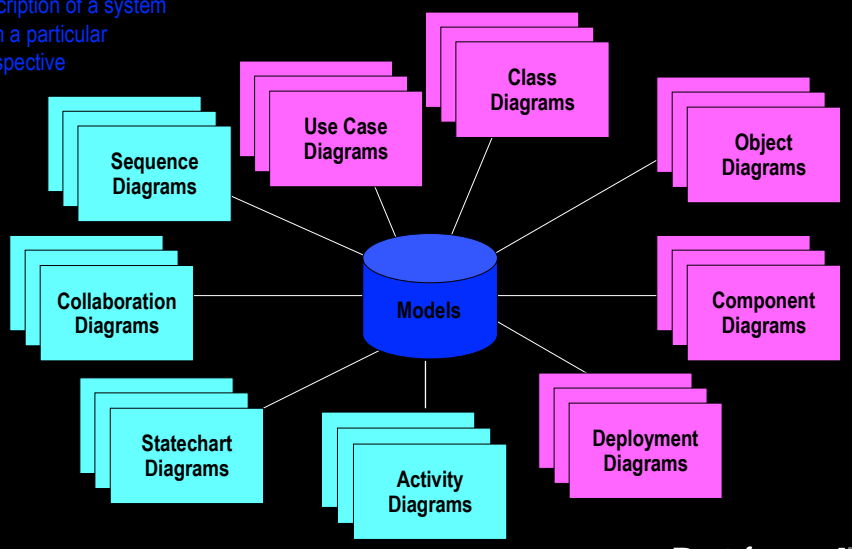


Rational® the e-development company™

Rational® the e-development company™

What UML Can Offer

A ***model*** is a complete description of a system from a particular perspective



Rational® the e-development company™

UML Concepts

- The UML may be used to visually model:
  - The interaction of your application with the outside world
  - The behavior of your application
  - The structure of your system
  - The architecture of your enterprise
  - The components in your system

Rational® the e-development company™

Agenda

- What is Architecture?
- What is UML?
- **What is a testable architecture?**
- What does UML provide for testing?
- What process does this imply?
- Some predictions

Rational
the e-development company

Testable Architecture

Three criteria determine testability of an architecture:

- Test Design Captured
- Runtime Test Access Provided
- Traceability Measured

Rational
the e-development company

Test Patterns Capture Repeatable Test Design

Binder's Template:

- Intent
- Context
- Fault Model
- Strategy
- Entry Criteria
- Exit Criteria
- Consequences
- Known Uses
- Related Patterns

Rational the e-development company

Test Patterns Apply At All Phases

- Unit Test of Class
  - Method: Pre-/poscondition
  - Invariant boundaries
    - Positive/negative
    - Equivalence classes
  - Inheritance
- Integration Test
  - Sequences
  - Transaction integrity
  - Exceptions
- System Test
  - "Login"
  - "Shopping cart"

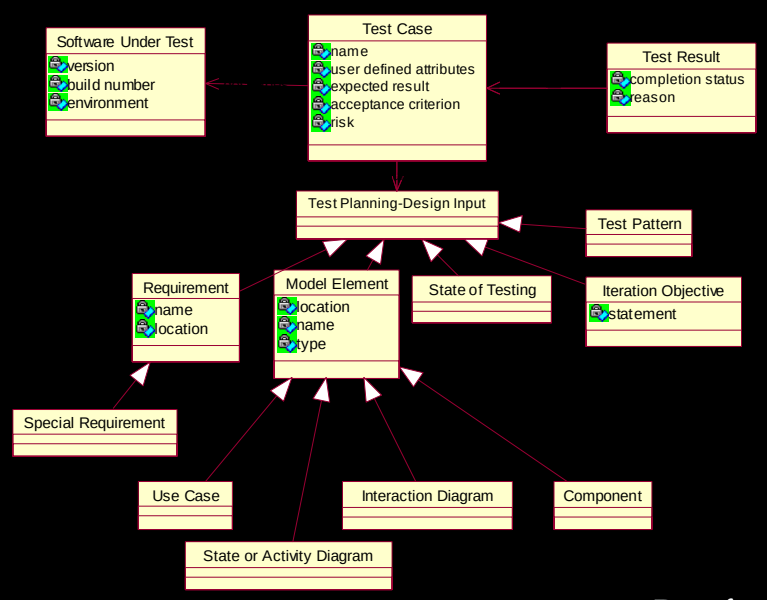
Rational the e-development company

Test Access

- Mechanism to allow nonintrusive automation
- Interface to provide
  - Preconditions
  - Postconditions
  - Built-In Self-Test
    - Invariant boundaries
    - Equivalence conditions
- May differ in debug and production

Rational
the e-development company

Traceability



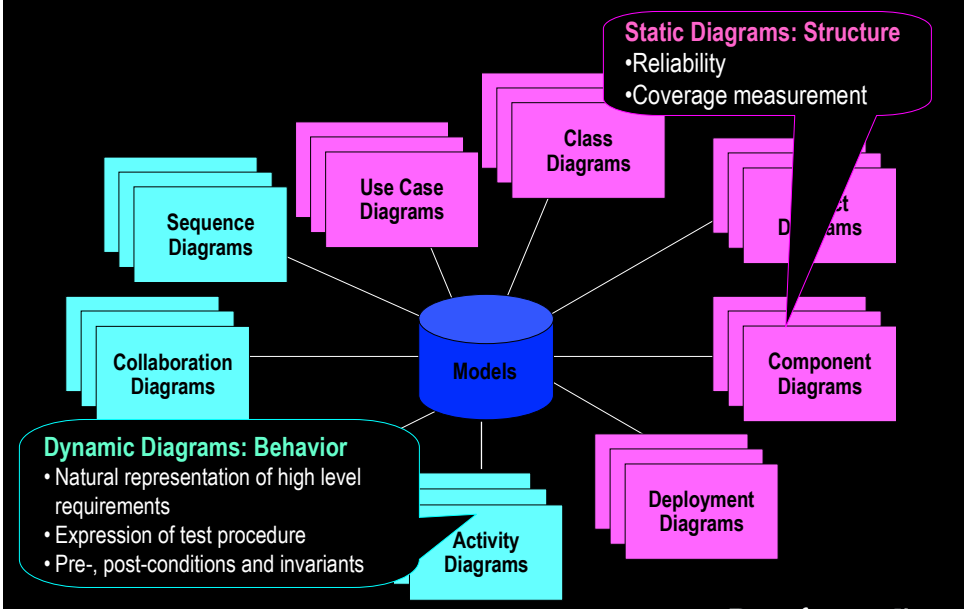
Rational
the e-development company

Agenda

- What is Architecture?
- What is UML?
- What is a testable architecture?
- **What does UML provide for testing?**
- What process does this imply?
- Some predictions

Rational
the e-development company™

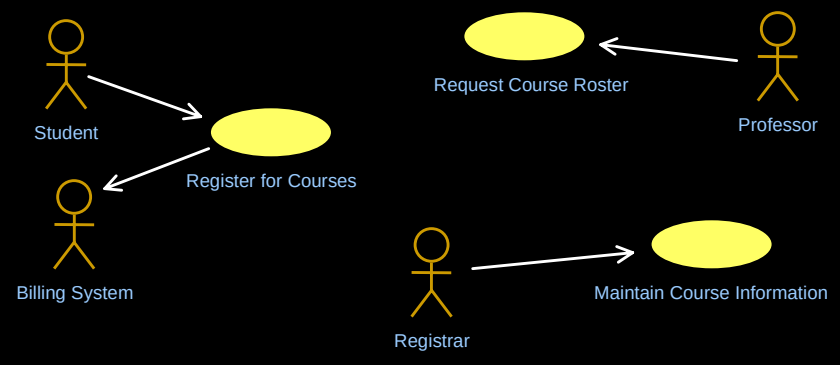
UML and Testing



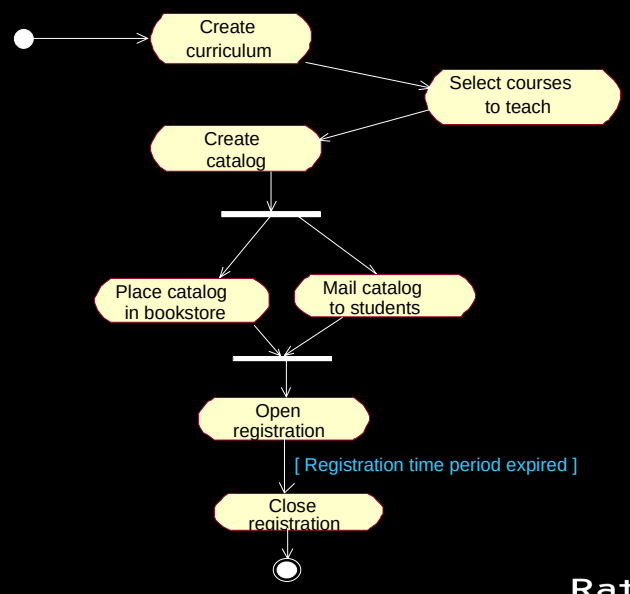
Rational
the e-development company™

Use Case Diagram

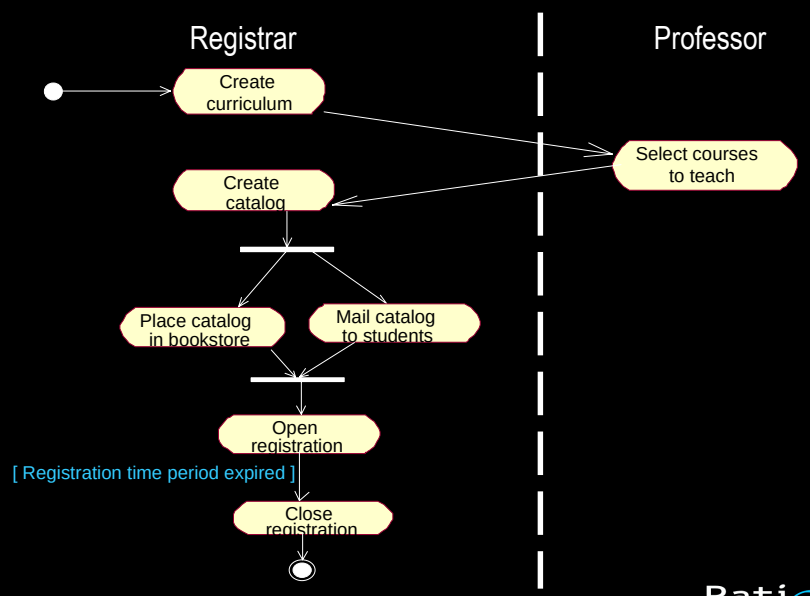
- Use case diagrams are created to visualize the interaction of your system with the outside world



Activity Diagram



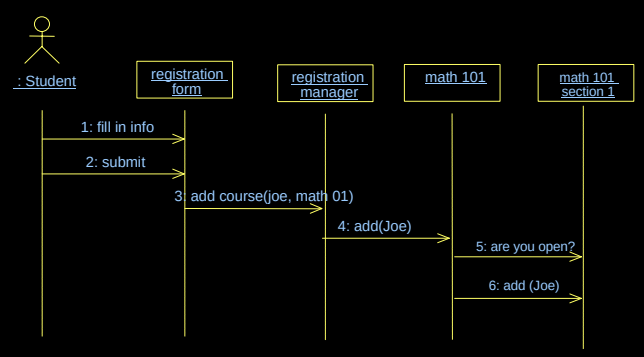
Swimlanes



Rational
the e-development company™

Sequence Diagram

- A sequence diagram shows step-by-step what has to happen to accomplish a piece of functionality provided by the system

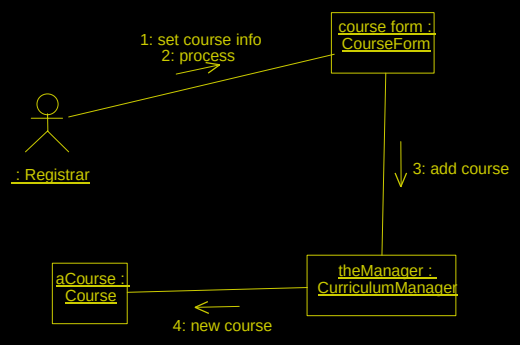


Rational
the e-development company™

Rational
the e-development company™

Collaboration Diagram

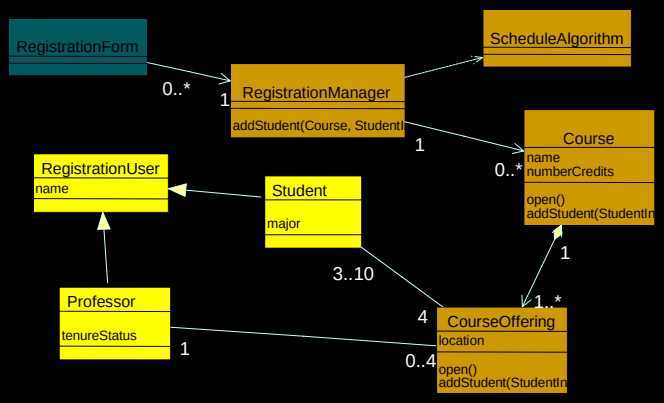
- A collaboration diagram displays object interactions organized around objects and their links to one another



Rational
the e-development company™

Class Diagram

- A class diagram shows the structure of your software

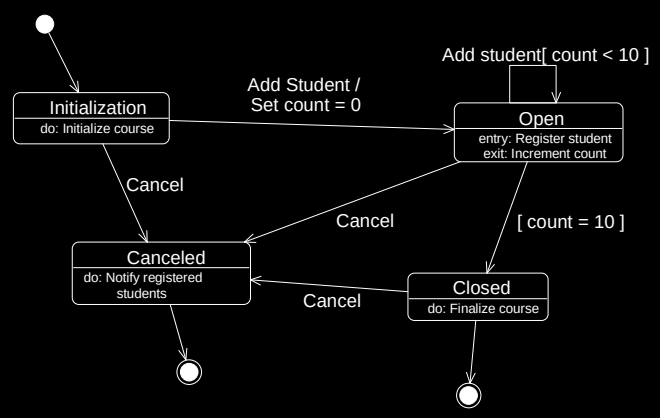


Rational
the e-development company™

Rational®
the e-development company™

The State of an Object

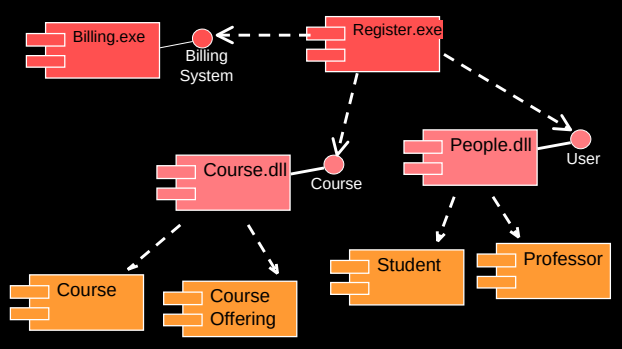
- A state transition diagram shows the lifecycle of a single class



Rational
the e-development company

The Physical World

- Component diagrams illustrate the organization and dependencies among software components

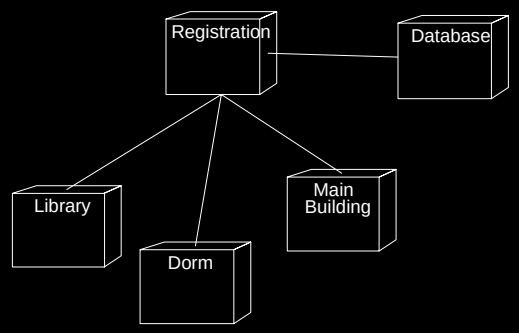


Rational
the e-development company

Rational
the e-development company™

Deploying the System

- The deployment diagram visualizes the distribution of components across the enterprise.



Rational
the e-development company

Extending the UML

- Stereotypes can be used to extend the UML notational elements
- Stereotypes may be used to classify and extend associations, inheritance relationships, classes, and components
- Examples:
  - Class stereotypes: interface, exception
  - Inheritance stereotypes: uses, extends
  - Component stereotypes: subsystem

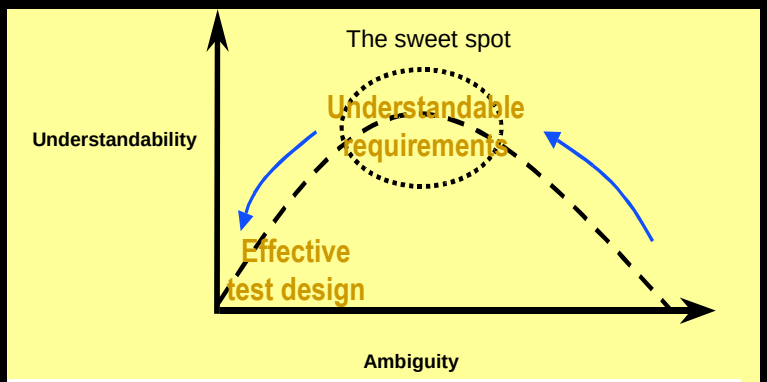
Rational
the e-development company

Agenda

- What is Architecture?
- What is UML?
- What is a testable architecture?
- What does UML provide for testing?
- **What process does this imply?**
- Some predictions

Rational
the e-development company™

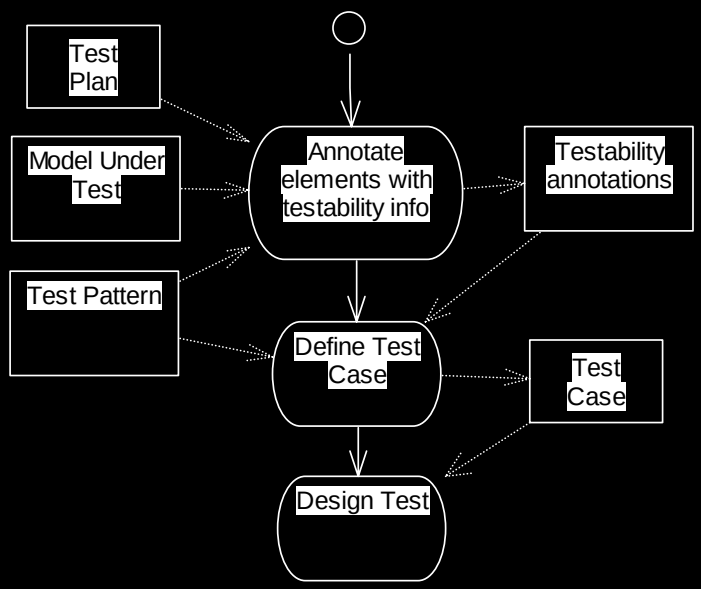
Test Design Paradox



Techniques that reduce ambiguity in a requirement often decrease understandability and alienate customers and users. Our goal is to find the "sweet spot" at which we attain the greatest amount of understandability with the least amount of ambiguity: *- Leffingwell*

Rational
the e-development company™

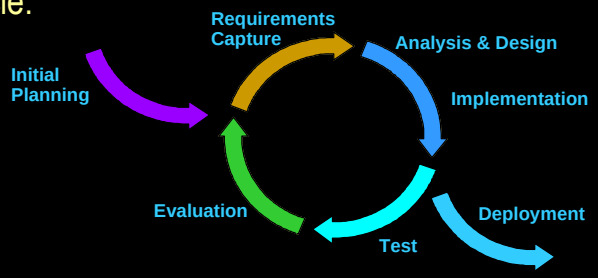
Using the Model for Design Capture



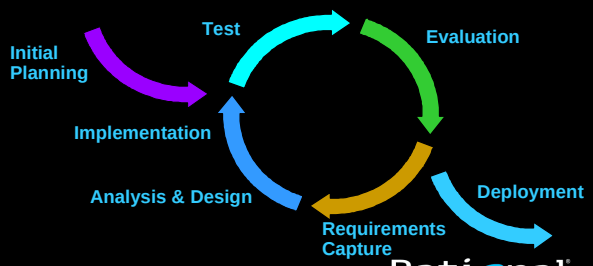
Rational
the e-development company™

Test Design Has To Be Roundtrip

- The theoretical development lifecycle: Requirements first



- The reality: Requirements not specified, but **discovered** through testing



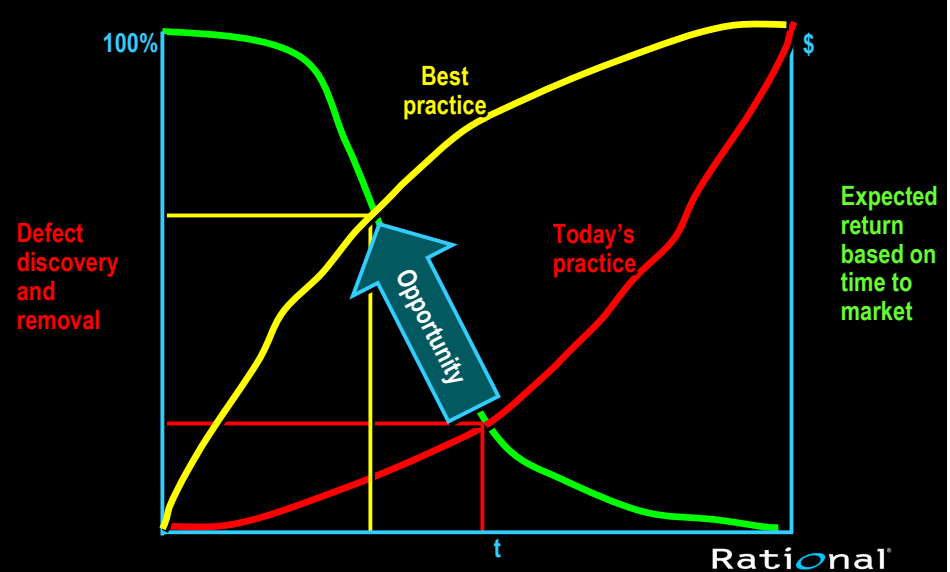
Rational
the e-development company™

Agenda

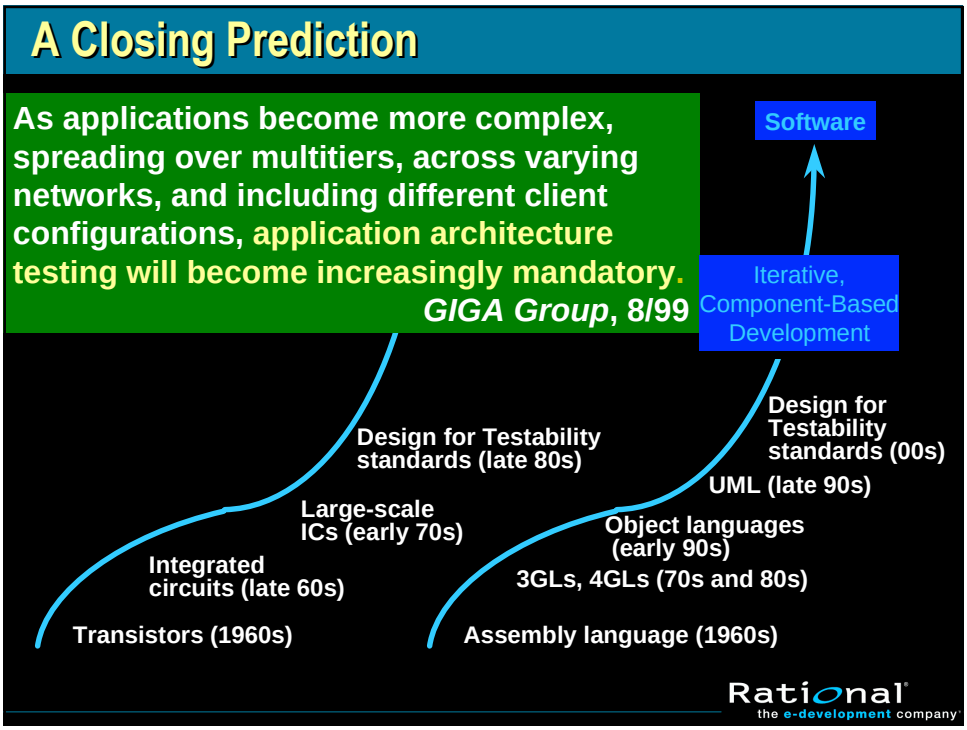
- What is Architecture?
- What is UML?
- What is a testable architecture?
- What does UML provide for testing?
- What process does this imply?
- **Some predictions**

Rational
the e-development company™

How We Can Improve Software Quality



Rational
the e-development company™


A Closing Prediction
As applications become more complex, spreading over multitiers, across varying networks, and including different client configurations, application architecture testing will become increasingly mandatory.
GIGA Group, 8/99
Software
Iterative, Component-Based Development
Design for Testability standards (late 80s)
Large-scale ICs (early 70s)
Integrated circuits (late 60s)
Transistors (1960s)
Design for Testability standards (00s)
UML (late 90s)
Object languages (early 90s)
3GLs, 4GLs (70s and 80s)
Assembly language (1960s)
Rational
the e-development company

QW2000 Paper 9A1

Mr. Rob Baarda
(IQUIP Informatica BV)

Risk Based Test Strategy

BACK TO QW2000 PROGRAM

Key Points

- Well defined steps from business risks to test coverage
- Early and stronger test involvement of all parties concerned
- Useful for all tests

Presentation Abstract

Testing of an information system should be based on the business risks for the organisation in using that information system. In practice, the test manager often takes the steps to go from risks to test coverage in an intuitive way. In this presentation, the steps to define a testing strategy are made explicit. This gives all parties involved better insight and provides a sound basis for negotiating testing depth.

A good risk assessment is a part of these steps. Very important is that this explicit way of looking at risks clearly shows that a test manager or tester can't do this alone. The involvement of users and managers of the client organisation and of project people like the developers, testers, QA'ers and project manager is necessary. Discussing risks and testing in the above way proves in practice to be real eye-openers for all parties concerned. This also enables negotiating about testing depth by letting the customer choose what should be tested how thoroughly.

The stepwise defining of the test strategy can be used for any test level and also for an overall strategy, including and co-ordinating all test levels and even inspections.

About the Author

Rob Baarda is an information systems professional for more than 20 years, following the path from programmer to consultant. Since 1986 he specialised in the field of testing. Starting with developing and implementing automated testtools, Rob moved after a few years to the methodology of testing. He is now part-time researching various test subjects in the R&D department of IQUIP, besides working as an international test consultant and teaching TMap« and TPI«. He also presented in QWE'99 about Risk Based Test Strategy.

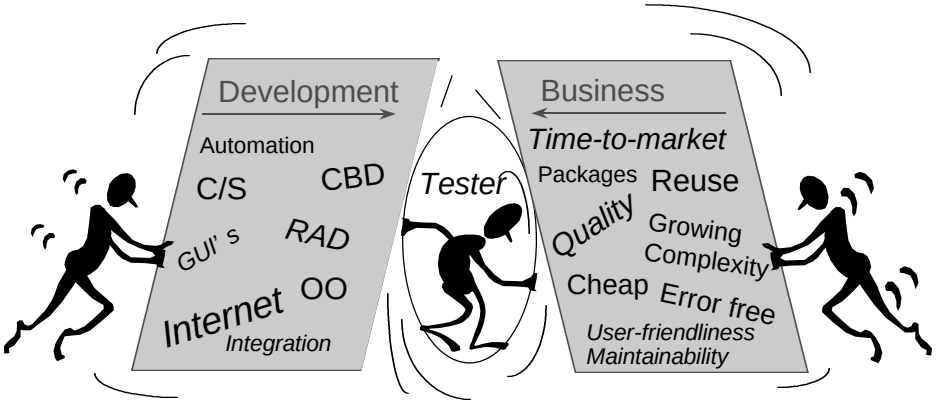
BACK TO QW2000 PROGRAM

Risk Based Test Strategy

Rob Baarda
baardaro@iquip.nl
IQUIP Informatica B.V.
The Netherlands

IQUIP
00 nr Be 1

Tester under pressure



Needed: communication between
Development, Business and Tester

IQUIP
00 nr Be 2

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Risk Based Test Strategy helps!

Agenda
- Test strategy
- Risk analysis
- Procedural steps
- Conclusion

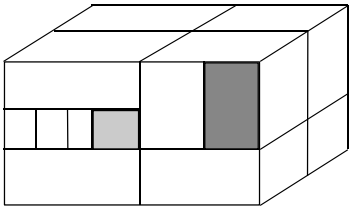
IQUIP
00 nr Be 3

What is Test Strategy?

Test Strategy =
the distribution of the effort for testing
-based on the test coverage-
over parts and quality characteristics
of the system under test

Quality characteristics:
- Functionality
- Performance
- Usability
- Security

Etc.



IQUIP
00 nr Be 4

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Aim of Test Strategy

Aim: To detect the most important defects as early as possible at the lowest costs!

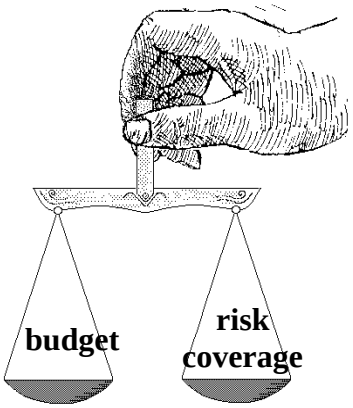
What are the most important defects?

Risk thinking gives the answer!

IQUIP
00 nr Be 5

Risk thinking: an introduction

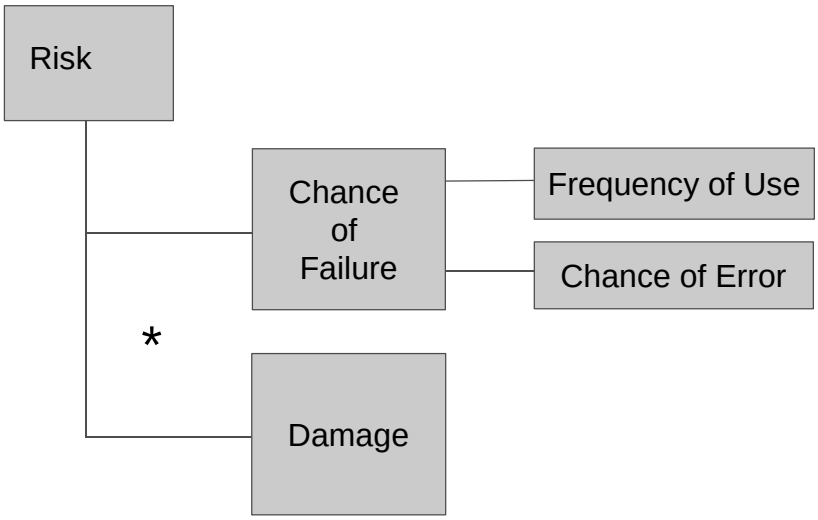
- Business reasons
- No risk, no test
- Risk analysis approach



IQUIP
00 nr Be 6

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Risk: definition



IQUIP
00 nr Be 7

Risk definition Chance of Error

- Chance of Error
  - globally
    = size * complexity
  - in detail
    = knowledge of development quality
- Possible development problems
  (partly based on Schaefer 1996)
  - Badly documented
  - Completely new functions
  - Frequently adjusted functions
  - Tools or techniques used for first time
  - Insufficient quality of low-level tests
  - Extreme time pressure

IQUIP
00 nr Be 8

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Risk definition Damage

- Damage
  - direct financial
  - loss of faith of customers
  - damage to corporate identity
  - impact on other functions and/or systems
  - detection and repair time

IQUIP
00 nr Be 9

Risk analysis applicable-1

- Applicable on the level of:
  - system
  - subsystem
  - individual function (e.g. interest calculation)
- Applicable:
  - New developments
  - Maintenance

IQUIP
00 nr Be 10

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Applied in maintenance

- Risk =
  Chance of failure * Damage
- Chance of failure =
  Frequency of use * Chance of error

IQUIP
00 nr Be 11

Risk analysis applicable-2

Applicable on quality characteristics

- What is the chance that a failure will happen and what will be the damage for:
  - functional defects
  - low performance
  - low maintainability
  - ...

IQUIP
00 nr Be 12

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Example

System	Failure 1-5	Damage 1-5	Risk 1-25
Debt management	5	5	25
Cast management	3	2	6
Stock transactions	1	5	5

Risk classes:

- Great risk (25)
- Middle risk (6,5)

IQUIP
00 nr Be 13

Important hints

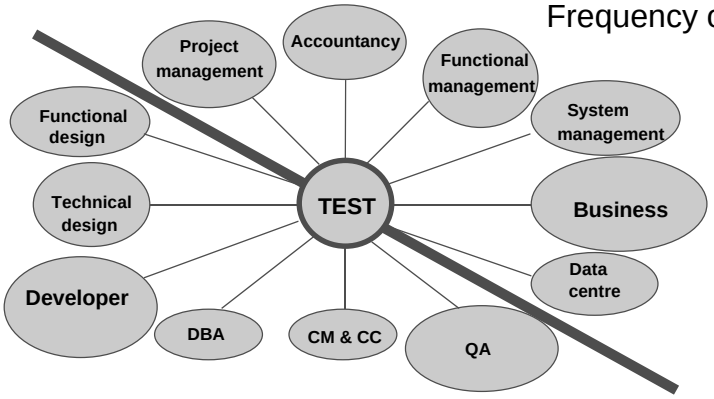
- Risk analysis should lead to a limited number of classes of (more or less) equal risks
- Pay attention to the scales of measuring
- Get the right parties involved

IQUIP
00 nr Be 14

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Parties involved

Damage
Frequency of use



Chance of error

IQUIP
00 nr Be 15

Risk Based Test Strategy helps!

Agenda

- Test strategy
- Risk analysis
- Procedural steps
- Conclusion

Test Strategy =
the distribution of the effort for testing

-based on the test coverage-

over parts and quality characteristics

of the system under test

IQUIP
00 nr Be 16

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Procedure visualized

Quality characteristics	Relative importance %	System Debt	Cash	Stock
Security	5			+ TT2
Functionality	75	+++ TT1	++ TT2	+ TT2
Usability	20	+ TT3		+ TT3
…	0			
	100	70	15	15

IQUIP
00 nr Be 17

Steps

- Master Test Plan
  - selection of quality characteristics
  - relative importance of quality characteristics based on risks
  - quality characteristics allocated to the test levels
- For one Test Level Plan
  - determine (relative importance) of system parts based on risks
  - specify test importance per system part and quality characteristic
  - choose test design techniques (TT) and testing depth

IQUIP
00 nr Be 18

Copyright (C) 2000, IQUIP Informatica B.V.
Diemen, The Netherlands

Test Strategy supports communication

A documented Risk Based Test Strategy helps:

- To be complete in assessing the risks
- Discussions with the parties involved
- To get commitment from the customer
- In the future to:
  - have documented proof if needed
  - make a switch in test manager position easier
  - do a re-planning
  - use in the maintenance situation

IQUIP
00 nr Be 19

Conclusion

Risk Based Test Strategy

- one line from risks to test design techniques
- less intuition, more business reasons
- applicable for different test levels
- supports customers to make choices
- is a medium to have communications between users, developers and testers
- makes the test process more manageable

QUESTIONS?

IQUIP
00 nr Be 20

Copyright (C) 2000, IQUIP Informatica B.V.

Risk Based Test Strategy

Rob Baarda

IQUIP Informatica B.V.

P.O. Box 263, 1110 AG Diemen, The Netherlands

Tel: +31 20 660 6600

Fax: +31 20 695 3298

E-mail: baardaro@iquip.nl

1. Introduction

Nowadays the pressure on the tester increasing due to demands from the business on one side and faster development of software on the other hand. A good approach for the tester to establish a good communication between all the parties is the use of a test strategy.. A test strategy is, in short, the choice which aspects of a (sub)system will be tested with what testing depth. In recent years we have had splendid opportunities for the further development and implementation of test strategies

We were able to combine theory on **business risks** with the existing theory of test strategy as described in TMap® (see references). This lead to a straight line of thinking from business risks into detailed testing depth per function. Some organizations started in parallel to change from the intuitive way of making a test strategy into a more rational one based on business risks.

In this paper we first describe the theory from business risk unto test techniques, followed by two concise case-stories of the previous described situations.

We will make clear that the described manner to define a test strategy has the advantage of:

- improved communication between the tester and other parties concerned;
- better test coverage on the right spot.

2. Test strategy and risks

The development of a test strategy is a means of communication with the customer commissioning the test on such matters as the organization of testing and the strategic choices that go with it. The test strategy indicates how testing is to be carried out. In order to make the best possible use of resources and time, it is decided on which parts and aspects of the system the emphasis should fall. The test strategy forms an important basis for a structured approach to testing and makes a major contribution to a manageable test process.

The customer who commissions the test will expect specific qualities of the system when in production, and wants to know whether the released system will meet these requirements. If the system qualitatively does not meet the requirements or only to a limited extent, this implies high damage for the organisation, for instance since high rework costs will be needed or clients/users will be unsatisfied. Therefore, this situation forms a risk for the organization. 'Risk' in this paper is defined as:

A risk is the chance of an error[1] occurring (chance of failure) related to the damage expected when this error does occur

---

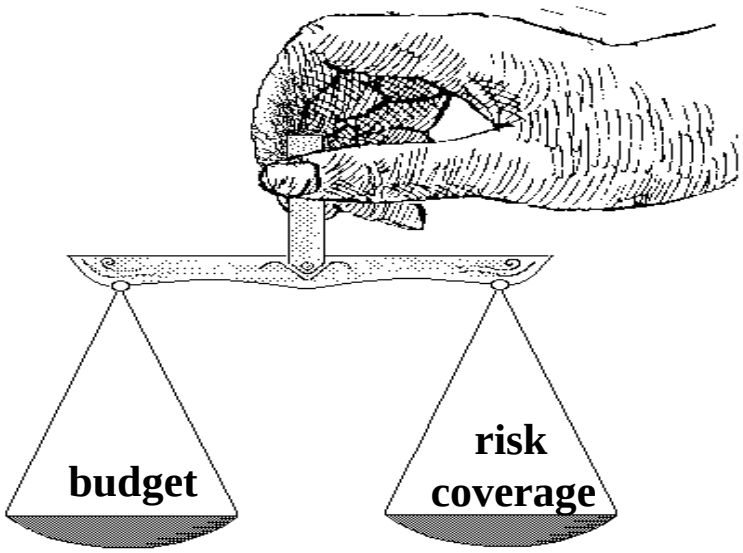
Testing covers such risks by giving insight into the extent to which the system meets the quality demands. When quality turns out to be insufficient timely measures can be taken, e.g. rework by developers. If the shipping of the system implies many risks for the organization, better testing is obvious as a solution. And the reverse also holds:

No risk, no test

Although in the above we refer to quality and risks in a general sense, there may be large differences depending on the situation. It is of great importance to discuss this with the customer, and to translate the customer's wishes in this respect into the way testing will be performed. Thus, the test strategy is directed towards finding the optimal balance between the test effort to be exerted and the coverage required for the risks. To this purpose the risks are specified up to the level of quality characteristics and separate subsystems. In doing so it becomes possible to find a suitable test coverage for the assessed risks. Here a higher test coverage usually results in more test effort. In order to reach at the variation in test coverage needed, the use of more than one test specification technique (test design technique), each offering a specified test coverage, is crucial.

An analogy with insurance industry may clarify this matter a bit more. A person wants to cover a relevant risk and takes an insurance with a coverage fitting this risk as best as possible. This insurance takes a certain premium. If the person wants to pay less, an insurance with a lower coverage is bought. The consequence is that there will be no payment if the uncovered risk occurs. On the other hand, if coverage was too large, then too much premium is paid, since a situation has been insured which is unlikely to occur for this person.

[1] The terms error, defect and failure are not used as exactly as IEEE advocates. In this paper error = fault or mistake; failure = the result or manifestation of one or more errors.



The balance between budget and risk coverage

3. Risk Assessment

Test strategy is based on risk assessment. This means assessing the damage of the consequences of failures, both undetected prior to operation and occurring during operation.

Risk assessment takes place on the basis of quality characteristics and subsystems. For instance, if the system is insufficiently user-friendly, what will be the negative consequences. And what will be the damage when the salary calculation module in a payroll system does not work correctly.

In order to be able to perform this assessment well, the separate aspects of a risk are considered:

*Risk = chance of failure * damage,*

where chance of failure is related to aspects including frequency of use and the chance of an error occurring.

These aspects are listed below:

- Frequency of use

In a function which is used dozens of times each day the chance of an error demonstrating itself is much bigger than with a function used once a year.

- Chance of error

For the assessment of the chance of errors the following list can be helpful. It presents the locations where errors tend to cluster. It is partly based on H. Schaefer, 1996 (*Surviving under time and budget pressure*, in: Conference Proceeding EuroSTAR1996, Amsterdam, the Netherlands):

  ◊ Complex functions;
  ◊ Completely new functions;
  ◊ (Especially frequently) adjusted functions;
  ◊ Functions for which certain tools or techniques were employed for the first time;

- ◊ Functions which were transferred from one developer to another during development;
- ◊ Functions that were realized under extreme time pressure;
- ◊ Functions which had to be optimized more frequently than on average;
- ◊ Functions in which many errors were found earlier (e.g. in previous releases or during earlier reviews);
- ◊ Functions with many interfaces;
- ◊ Inexperienced developers;
- ◊ Insufficient involvement of users;
- ◊ Insufficient quality assurance during development;
- ◊ Insufficient quality of low-level tests;
- ◊ New development tools and development environment;
- ◊ Large development teams;
- ◊ Development teams with sub-optimal communication (e.g. owing to geographical spread or personal causes);

- Damage

If and when the error manifests itself, what will be the damage for the organization. Aspects are costs of repair (both of the system and of the consequences), forgone income and loss of clients or of confidence. Usually the damage increases if the error has its impact on other functions or systems. In the case of errors occurring in batch processes there may be a possibility to prevent them from hampering users, so that the eventual damage will be smaller than with similar on-line processes. Of course, this only holds if errors are detected on time.

Because of the complexity of the matter, it is impossible to assess risks with complete objectivity and in detail: it is a global assessment. It is therefore important for the risk assessment not to be carried out by the test manager alone. A large number of people involved in the scheme should contribute: customer, users, development team, accountants, IT auditors and so on. This not only increases the quality of the strategy, but it also has the advantage that the different parties are more aware of the risks and the extent to which testing contributes to making these risks manageable in a better way.

The developer of the test strategy should realize that 'users' are the best people to assess the damage and the frequency of use when valuing the risks (end-users, system managers and application managers, line management), whereas project team members are best to assess the chance of error (project managers, designers, programmers, project quality staff, test manager).

The focus in risk assessment is on product risks, or, in other words, what is the risk for the organization if the product does not demonstrate the expected quality. In addition to this, there are also (test) project risks. If the system must be in production on January 1st, if functional specifications are produced too late, if no experienced testers are available, or if the test infrastructure is not ready on time, then we speak of (test) project risks. These are not taken into account in determining the test strategy; they do play a role in the test plan.

In developing a test strategy the aim is to see to it that the test will be organized in such a way that with a certain extent of reliability

- the most important problems will be found;
- the problems will be found in an early stage;
- the problems that require the most rework time will be found first:
- efficient use is made of resources;
- and eventually an accurate quality advice can be given.

This can be summarized as:

Test strategy aims at finding the **most important** errors as **early** as possible against the **lowest costs**

In practice, the development of a test strategy is often planned to coincide with preparing the budget, for example with the help of test point analysis. The advantage is that the consequences of the adopted strategy are immediately translated into time required for testing, and consequently the cost of testing, which makes the strategic choices manageable. If the time available for testing is more or less fixed, it is also possible to use test strategy combined with test point analysis to determine what is achievable within the time limits. It is probably even more important to make it clear at this time which parts cannot be tested, or cannot be fully tested, and what risks will therefore be incurred.

4. Quality Characteristics

The quality characteristics we distinguish can be divided into dynamic and static quality characteristics. The dynamic quality characteristics deal with features of the information system in use; examples are security, usability, continuity, traceability, functionality, userfriendliness, suitability, efficiency, performance. The static are concerned with intrinsic characteristics of the information system and the documentation, as considered from the standpoint of developers and future system managers. Examples are manageability, maintainability, connectivity, reusability, portability, testability.

5. Steps in different situations

The development of a test strategy is not something that can be done purely methodically or formally. The below steps are aids and indicators. Experience and skills of the performer of this activity in the area of testing is a major success factor for a sound test strategy.

One should also realize that test strategies arise as a result of iterative processes and in connection with other activities for a test plan. If the first test strategy produces an amount of test effort needed or a certain time schedule which is unacceptable for the customer, the strategy should be adjusted. The lack of test skills or suitable infrastructure can also result in adjustments of the test strategy.

The stepwise defining of the test strategy can be used for any test level and also for an overall strategy (master test plan), including and coordinating all test levels and even inspections. The steps differ for both situations.

Also we describe shortly considerations managing the test process and for the maintenance situation.

5.1 Strategy in Master Test Planning

The steps to be taken for a test strategy are:

- Decide on the quality characteristics;

- Determine relative importance of quality characteristics;
- Attribute quality characteristics to test levels.

5.1.1 Step 1: Selection of Quality Characteristics

In close liaison with the customer and other parties involved a selection of quality characteristics is made on which the tests must focus. In doing so one should take risks for the business into account as well as aspects including system requirements, business objectives concerning the information system, directions and standards set by the computer center. These quality characteristics are also used for reporting to the customer during test execution and completion.

Some characteristics are difficult to test. There may be a wish for a system to be user-friendly and flexible, for instance, but these wishes turn out not to have been translated into measurable requirements. That is why a substantial part of the effort here is devoted to formulating the relevant quality demands as measurably and unambiguously as possible. It is also the case that some quality characteristics demand relatively much effort in testing. Since it is not useful to offer possibilities which cannot be fulfilled, it should be determined beforehand what will be the estimated effort needed for a decision made.

For non-IT people our quality characteristics may be hard to handle. It helps when we translate them to the conceptual environment of our conversational partners. This can be done by finding illustrative examples of problems or errors that may occur in production and the damage that would be caused by this. This is one of the most difficult aspects of the formulation of a test strategy.

5.1.2 Step 2: Relative Importance of Quality Characteristics

On the basis of the results from Step 1 the importance of the selected quality characteristics is determined in relation to one another. This is done in the Matrix of Weights (see below), by weighing the relative risks per quality characteristic. Here the relative importance is indicated (in percentages). Note that it is not of importance to have exact percentages: the objective is to arrive at a general picture of the relative importance of the various quality characteristics. The filling in of the matrix helps evaluating the risks.

The customer should be forced to make choices. Therefore, as a directive we ask for a percentage of 5 as the minimum. The sum of all percentages should not exceed 100. An example of a Matrix of Weights is given below:

Quality characteristic	Relative importance
Manageability	5
Security	5
Usability	-
Connectivity	-
Continuity	10
Traceability	-
Flexibility	-

Functionality	50
Userfriendliness	10
Reusability	-
Infrastructure	-
Suitability	10
Maintainability	5
Performance	5
Portability	-
Testability	-
Efficiency	-
Total	100%

The Matrix of Weights

The high percentage for functionality in this matrix may strike the reader. This is in conformance with practical experience: generally 50% of the importance or more is attributed to this characteristic. The reason for this is that risks usually are larger for incorrect performing systems (Functionality) than for slow systems (Performance) or awkward systems (Userfriendliness).

5.1.3 Step 3: Quality Characteristics Attributed to Test Levels

With the aim of spending the total test effort as efficiently as possible, during test strategy development it is decided with which test level or combination of test levels the various selected quality characteristics will be tested. Also inspections may fall under the scope of the master test plan and under the test strategy. In the remaining sections when 'test' is used, inspections are also included.

In this way the various test levels within a project are brought into balance. It is obvious that the different responsibilities and authorities remain intact.

A +-sign in a matrix (for an example, see matrix below) indicates whether the test strategy takes a quality characteristic into account. '++' or '+++' indicate that relatively much attention is to be paid to the quality characteristic for the specified test level. It is obvious that one quality characteristic can be in effect for more than one test level, but depth will often vary. If structured test specification techniques are used, the acceptance test, for example, may use results of previous tests levels, on the basis of which it may be decided to test with less depth.

	Insp RQMS	Insp Specs	Insp Design	PT	IT	ST	FAT	PAT	Relative importance
Manageability	+	+	++					+	5
Security	+	+	+				+	+	5
Usability									-
Connectivity									-
Continuity	+		+					++	10
Traceability									-
Flexibility									-
Functionality	++	++		+	+	+++	++		50
Userfriendliness		++					++		10
Reusability									-
Infrastructure									-
Suitability	+	++					++		10
Maintainability		+				+			5
Performance			+				+	+	5
Portability									-
Testability									-
Efficiency									-
									100%

Example of a Test Strategy for Test Levels

Legenda:

Insp RQMS	Inspection of Requirements
Insp Specs	Inspection of Functional Specification
Insp Design	Inspection of Technical Design
PT	Program Test
IT	Integration Test
ST	System Test
FAT	Functional Acceptance Test
PAT	Production Acceptance Test

5.2 Strategy for a Test Level

The steps to be taken for a test strategy for a specific test level are:

1. Decide on the quality characteristics;
2. Determine relative importance of quality characteristics;
3. Divide the system into subsystems;
4. Determine relative importance of subsystems;
5. Specify test importance per subsystem and quality characteristic;
6. Establish test techniques to be used.

The strategy determination for a specific test level naturally has the master test plan strategy as a precondition and a starting point. If a master test plan, including a test strategy, is there, step 1 can be omitted and step 2 will be an easy and fast performed activity. Nevertheless, all steps are worked out below.

5.2.1 Step 1: Decide on Quality Characteristics

In collaboration with the customer and perhaps other parties concerned the quality characteristics are determined on which the test will focus, in relation to business risks. During the test and in the completion phase, results are reported on the basis of these quality characteristics.

5.2.2 Step 2: Determine Relative Importance of Quality Characteristics

Based on the results of step 1 the relative importance of the selected quality characteristics is determined. Determination of the importance takes place by weighing the risks per quality characteristic. This is shown in a Matrix of Weights by a percentage in the column Relative importance. In order to force the customer to make choices, a percentage of 5 is the minimum.

An example of a matrix for a functional acceptance test is given below:

Quality characteristic	Relative importance
Security	5
Functionality	60
Userfriendliness	10
Performance	5
Suitability	20
Total	100%

The Matrix of Weights for a Functional Acceptance Test (Example)

5.2.3 Step 3: Divide System into Subsystems

During this step and the following steps the test strategy is refined more and more. This implies that the quality characteristics and their relative importance as indicated in the Matrix of Weights are to be broken down for the combination of test specification technique and subsystem, later even for test specification technique and test unit.

The information system is divided into subsystems. The reason for this is that the same quality demands do not have to be valid for each subsystem. Moreover, the various subsystems may have different risks for the organization. In principle the division is the same as given in the design documentation. If we deviate from this one, we must clearly indicate the motivation for this. Examples of alternative divisions are on the basis of extent of risk or on the basis of order of release by the developer. If a conversion module is there, this is to be treated as a separate subsystem. Often the subsystem 'Total system' is distinguished. This serves the purpose of indicating that some quality characteristics can be evaluated effectively only with the help of an integral test, testing the coherence of the various subsystems.

In a later stage the various subsystems are further divided into independent test units. E.g. in a logistics system the subsystem Sales may be divided into the test units Quotations (all functions regarding quotations) and Orders.

5.2.4 Step 4: Determine Relative Importance of Subsystems

On the basis of the result of the previous step the relative importance (in percentages) of the subsystems should be indicated in the Matrix of Weight. This is done by weighing the risks per subsystem. It is not a matter of exact percentages; rather it is a matter of getting a general image of the importance of the subsystems as seen through the eyes of the customer and other parties concerned. This step helps in asking people to form an opinion of this.

The relative importance is determined of each subsystem within the information system. In the Matrix of Weights this is indicated with a percentage in the column Relative importance.

An example of a Matrix of Weight for a functional acceptance test (based on the Strategy Matrix for test levels in the master test plan, shown above) is given here:

	Relative importance
Subsystem 1	30
Subsystem 2	15
Subsystem 3	20
Conversion	15
System	20
Total	100%

Relative Importance of Subsystems for a Functional Acceptance Test (Example)

5.2.5 Step 5: Specify Test Importance per Subsystem and Quality Characteristic

Finally a refinement is made by assessing the importance of the combination quality characteristic - subsystem. E.g., a refinement may be that userfriendliness is important (relative importance of 10), but this holds predominantly for subsystem 1 and not at all for subsystem 3. Again it is emphasized that test strategy determination is not a mathematical affair: it is meant to get an image of the relative test importance of the various subsystems and quality characteristics. This is also the reason why we choose +, ++ and +++ as notation symbols, rather than opting for the pseudo-certainty of a mathematical formula. An example of this is the following: suppose both userfriendliness and a specific batch subsystem are very important, a mathematical formula would probably result in large test effort to be spent on the userfriendliness of the batch procedure. The Matrix of Weight may look like this:

	Subsystem 1	Subsystem 2	Subsystem 3	Con-version	Total system	Relative importance
Security	+	+				5
Usability						-
Continuity						-
Traceability						-
Functionality	++	+	+	++	+	60
Userfriendliness	++	+				10
Performance	+		+			5
Suitability	+	+	+		++	20
Efficiency						-
Relative importance	30	15	20	15	20	100%

Relative Importance of Subsystem x Quality Characteristic (Example)

5.2.6 Step 6: Establish Test Specification Techniques to be Used

The final step in test strategy involves the selection of the techniques that will be used to test the combination of the selected quality characteristics and subsystems. A high importance implies the use of techniques with a high coverage or the use of more techniques, a low importance implies the use of techniques with a lower coverage or the use of fewer techniques.

In choosing the techniques one should also take into account various other factors, a number of which are listed below.

- Quality characteristic to be tested

A technique is fit for testing one or more quality characteristics. Some quality characteristics can best be tested with one (set of) techniques, others with another one.

- Area of application

Some techniques are specifically suitable for testing the interaction (screens, reports, on-line) between the system and the users, others are better in testing the processing of systems (batch processes). There is a relation with the type of error to be found with a technique, e.g. false input checks, incorrect processing or errors of integration.

- Availability of test basis

Each techniques starts from a certain test basis. This may be the functional specification, the technical design, program code or descriptions of the end-user organization. The exact form of the test basis is also relevant to the choice of a technique, e.g. decision tables, pseudo-code, structured language or unstructured prose.

- Extent of formality

Informal test specification techniques offer more freedom for the tester in making the test cases than do formal techniques.

- Use of resources

The application of a techniques requires a specific amount of resources, in terms of man capacity as well as machine capacity. The use of resources has a direct relation with costs.

- Required knowledge and skills

Not each tester is equipped for each technique. For the useful application of some techniques much business knowledge is needed. For other techniques more analytic talent is required. Therefore, the knowledge and skills of the test staff also influences the choice for techniques.

For practical reasons one should attempt to cover all selected quality characteristics with a minimal set of test specification techniques.

The selection of the test specification techniques should be done in an early stage of the test process, for then the test team can take the appropriate actions in training for techniques and the necessary checklists can be made or adjusted for the specific situation.

As a result of this step the techniques that will be used per subsystem are defined. Optionally, especially with large test projects, this last step in the test strategy is performed slightly later in the process, namely during the preparation phase. As a part of this the priority order of the tests to be performed is determined. The aim of this is to have the most important tests take place as early as possible.

5.3 Strategy during the Test Process

The test strategy determined in advance often will be put under pressure in a later stage of the test project. In such a situation the test manager is asked to perform less tests or shorter tests in order to conform to the adjusted schedule. The consequences are to be seen mainly in the last step of strategy development: suddenly some tests must be cancelled or must be carried out with less depth. Using the test strategy as a basis the test manager may discuss with the customer which tests can be dropped or where less thorough testing can be done. By indicating which parts are to be tested less in relation to the risks assessed (translated into importance levels in the strategy), the test manager can report in a solid fashion on the increased risks after the testing phase. Therefore it is essential not to change the steps 1 to 5: the risks and the importance levels do not change. The result is that, when testing is reduced, there will be more risks after the system has been implemented.

Apart from this there is also the situation that during testing it turns out that part of the system contains an excessive number of errors or excessively few errors.

Both cases justify adjustments in the test strategy, namely the increase and decrease of test effort, respectively. Contrary to the situation in the previous paragraph here risks will remain the same after implementation of the system. The correction can be summarized as follows:

Testing should continue as long as the costs for finding and correcting errors during testing are lower than the costs connected to the error occurring in production

---

In 'finding and correcting errors' more costs than just test costs play a role; other, extensive costs may be concerned with the delay in shipping the product. For 'the error occurring in production' one should also take into account the chance that the error will actually occur: an error that will never occur is no error (defect)!

5.4 Strategy during Maintenance

The main difference between the development of new systems and maintenance for the test strategy is the chance of error. In the case of maintenance changes are made to an existing information system. These changes should be tested. During maintenance there is a risk that new errors are introduced, with a decrease in quality for the system as a result (regression).

The implication of this different chance of error in the case of maintenance implies for the strategy that the relative importance of the subsystems may change: a subsystem which had a high importance when it was developed, may be unchanged in maintenance. Since the chance of regression is the only risk in this case, the test importance is much lower. Therefore, test strategy development for a test level can be modified by changing the concept of 'subsystem' in the steps to 'change'. For each change it is analyzed which system parts were mutated, which parts may have been influenced by the change, and which quality characteristics are relevant. There are various possibilities for testing each change, dependent on the risks:

- A limited test, only focused on the change;
- A complete (re)test of the function which had been changed;
- A test of the coherence of the changed function and the adjacent functions.

There should also be a regression test for the system as a whole. This test focuses on the coherence of the changed and unchanged parts of the system, since the chance of regression is largest here. If the test strategy for the new developed system is available, the importance levels attributed to the subsystems here can be of use.

Apart from the changed chance of error there are more differences between a system developed and a system under maintenance. However, they have no influence on the *technique* of test strategy development. Examples of these differences include:

- Test basis is missing, incomplete or not up-to-date

This situation, frequent in maintenance, has consequences for the test specification techniques to be chosen.

- Predictable versus ad-hoc maintenance

The majority of maintenance situations are predictable and can therefore be planned. The strategy determination is easily to be applied to this type of maintenance. The situation is more difficult in the case of ad-hoc maintenance, where the focus is on putting right a production break-down and getting the system in the air as soon as possible. A formal strategy determination costs too much time here. It is feasible,

however, to have some test strategy scenarios available: if program x goes wrong and is repaired, what should be tested? These scenarios support an optimal test for ad-hoc maintenance.

6. Conclusion

The testing of information systems can be based on the business risks which the organization will experience in using these information systems if the system is not tested. In practice, test managers often take the steps to come from risks to test coverage in an intuitive manner. In this paper, the steps needed for the definition of a test strategy are made explicit. The result of such a test strategy is better insight for all parties involved and a sound basis for negotiating testing depth.

Good risk assessment is a part of these steps. It is essential to realize that this explicit way of looking at risks cannot be done by a test manager or tester alone. It is necessary to ascertain for the involvement of users and managers of the client organization, of auditors, and of project people such as developers, testers, QA staff and project managers. In practice, the discussion of risks and related testing strategies in this way proves to be a real eye-opener for all parties concerned. It also enables negotiation of testing depth by having the customer decide which elements should be tested how thoroughly.

The stepwise definition of the test strategy can be used for any test level (e.g., system test, acceptance test) and also for an overall strategy (master) test plan, including and coordinating all test levels and inspections/reviews.

7. References

In English with a short description of test strategy
Pol, Martin and Veenendaal, Erik van, Structured Testing of Informations Systems (1998), Kluwer Deventer The Netherlands, ISBN 90-267-2910-3,

In Dutch with an extensive description of test strategy
Pol, Martin, Teunissen, Ruud and Veenendaal, Erik van, Testen volgens TMap (1999), Tutein Nolthenius, s' Hertogenbosch The Netherlands, ISBN 90-72194-58-6.
The translation into English is planned for the end of 2000.

QW2000 Paper 9W1

Dr. Wen-Kui Chang & Mr. Shing-Kai Hon (Tunghai University)

A Systematic Framework for Ensuring Link Validity under Web Browsing Environments

BACK TO QW2000 PROGRAM

Key Points

This paper will present a methodology to test web links, and walk you through a practical software certification process It takes a user-oriented software validation approach.

- Introduction of a framework for website quality assurance
- Investigation application of statistical usage testing
- Consideration from the end-user operations

Presentation Abstract

Web applications are increasingly developed in many domains and most were implemented by the ad hoc approach. However, various quality metrics become progressively more difficult to measure and mange as the developing application size and scope grows. Among the typical non-functional characteristics, in this research, factors of link validity will be deeply investigated. Essentially link validity includes various characteristics such as correctness, relevance, completeness and integrity implicitly, which will deeply affect the effectiveness and efficiency on information retrieval and browsing.

In most web applications, it is unlikely that the application developers may validate all possible navigation paths. In this paper, a framework of quantitative certification on link validity will be proposed to ensure that all linked paths provide consistent and reasonable information streams and appropriate contexts.

In this research, the rational for statistical usage testing is investigated and employed to certify links for a website. In principle, a software usage model is first established, which characterizes various operational uses of a software system. Suppose that an operational use is a skeleton for the intended use of the software in an intended environment. Thus, all possible operational uses of a software system will constitute a population with a huge size. If a usage sample of test cases is drawn statistically from the usage population, performance on this sample may then be used as a basis for the evaluation of software quality.

Under the web-browsing environments, all possible navigation paths are first formulated to represent a usage model with the Markov chain property, that is then analyzed and used to generate test script file

statistically. The proposed mechanism is not only systematic on certifying effectively hyper links, but also efficient highly for those complex information structures. A real application of the proposed approach to a web application will also be demonstrated quantitatively through a certification tool ToolCertify.

About the Author

Associate Professor at Software Engineering Laboratory in Tunghai University

Points of interest: software engineering, software quality assurance, software quality certification, project management

BACK TO QW2000 PROGRAM

A System Framework for Ensuring Link Validity under Web Browsing Environments

Wen-Kui Chang and Shing-Kai Hon
Dept. of Computer & Information Sciences
Tunghai University, Taiwan ROC

outlines

- 1. introduction
- 2. web links and information structure
- 3. software quality certification
- 4. certification on link correctness
- 5. conclusions

introduction

- this paper investigates
  - issue of link validity
    - under web browsing environment
- rationale of statistical usage testing
  - is employed and adopted for
    - web-based certification environments
- all possible navigation paths
  - are formulated to
    - represent a usage model with
      - ✓ Markov chain property

web links and information structure

- 1. web links
  - structure link
  - associative link
  - referential link
- 2. link validity
- 3. link correctness
  - content correctness
  - linking correctness
- 4. information structure

software quality certification

- 1. statistical usage testing
- 2. software usage model
- 3. Markovian usage model
- 4. statistical usage testing procedure

statistical usage testing

- rationale lies in the fact that
  - failures which occurred most frequently in practical use
    - will be found early during the test cycle
- main benefit
  - make use of statistical inference techniques to
    - compute probabilistic properties of the testing process

software usage model

- characterize various operational uses of a software system
- If a usage sample of test cases is drawn statistically from usage population
  - performance on the sample may be used as a basis for
    - evaluation of software quality
- in this paper
  - Markov approach will be employed for
    - usage modeling

markovian usage model

- usage model consists of
  - all usage states that
    - are connected by links which
      - ✓ indicate all possible stimuli, responses
      - ✓ with a probability index
- generated test scenarios are formulated as
  - Markov chain by the following facts
    - occurrence of the current state depending on the previous state only
    - all usage states are incompatibility
    - all probabilities emerged from each state are summed to one

statistical usage testing procedure

- 1. building a usage model that defines
  - all possible events
  - their transition distribution
- 2. generating test cases statistically
  - by associated distribution
- 3. executing test cases
- 4. collecting
  - performance information
  - inter-failure data
- 5. certifying the software by
  - reliability evaluation model

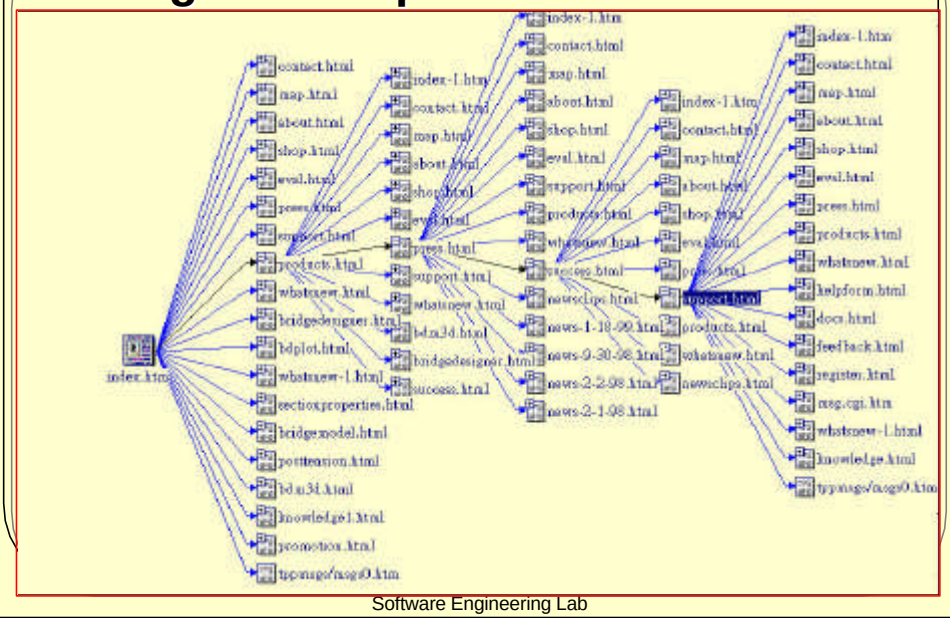
certification demonstration on link correctness

- 1. usage model
- 2. analysis report
- 3. failure analysis
- 4. certification result
- 4. discussion

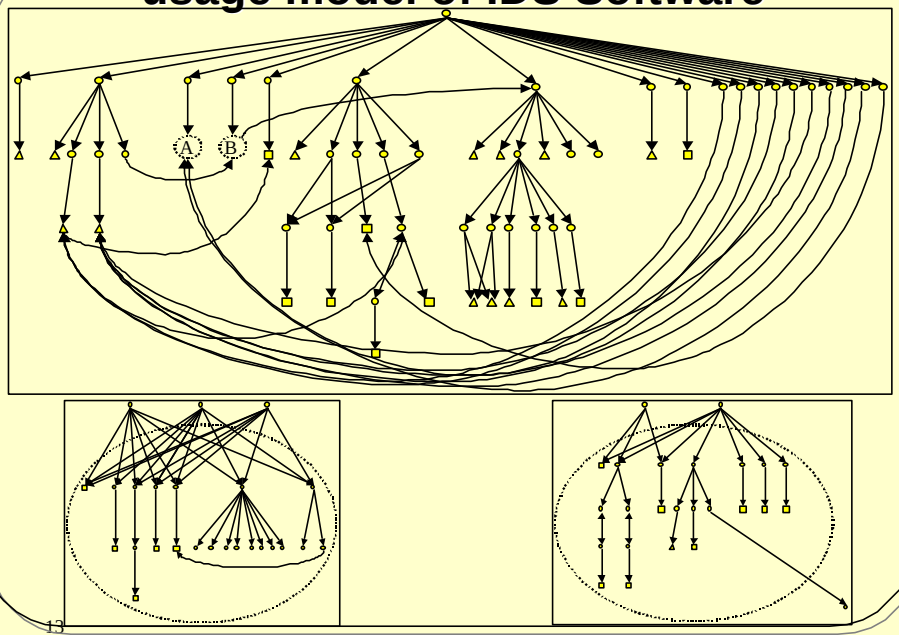
usage model

- **begin with statistical usage testing**
  - **build navigation structure to**
    - delegate all possible navigation paths for a web site
- **navigation map may be established by**
  - **FrontPage**
- **some hyperlink paths have no termination that**
  - **may be assumed to terminate at**
    - some level

navigation map for IDS SOFTWARE



usage model of IDS Software



analysis report

Number Of Nondeleted States	78
Number Of Active Arcs	154
Expected Script Length	4.586
Least Likely State Coverage Expected At	652.909
Least Likely Transition Coverage Expected At	652.909

failure analysis

Failure ID	Mean First Passage	Probability of Occurrence
1	0.000291	0.001307
2	0.000291	0.001307
3	0.000291	0.001307
4	0.000291	0.001307
5	0.000291	0.001307
6	0.000291	0.001307
7	0.000291	0.001307
8	0.000291	0.001307
9	0.000291	0.001307
10	0.000291	0.001307

certification result

Script #	Result	MTTF	Reliability	C=95%	C=99%	% States Certified	% Arcs Certified
30	Fail	29.99997	0.966667	0.011531	0.015154	0.8719	39.74359
31	Pass	30.99997	0.967742	0.011165	0.014674	0.862504	41.02564
33	Pass	33.00003	0.969697	0.010498	0.013797	0.849346	42.30769
34	Fail	17.00000	0.941176	0.013798	0.018135	0.849319	42.30769
232	Pass	116.0001	0.991379	0.002051	0.002696	84.61539	79.22078
233	Fail	77.66654	0.987124	0.002113	0.002778	84.61539	79.22078
234	Pass	77.99983	0.987179	0.002104	0.002766	84.61539	79.22078
576	Fail	64	0.984375	0.000968	0.001272	97.4359	96.75324
584	Fail	58.40002	0.982877	0.000965	0.001268	97.4359	97.4026
762	Pass	76.20015	0.986877	0.000739	0.000972	98.71795	98.7013
763	Pass	76.29996	0.986894	0.000739	0.000971	98.71795	98.7013
764	Pass	76.40002	0.986911	0.000738	0.000969	98.71795	98.7013
765	Pass	76.50001	0.986928	0.000737	0.000968	100	100

discussion

- 1. challenges
- 2. research in future

challenges

- if states in usage model are so many
  - the number of test scripts will
    - become quite huge
- in this research
  - manual execution of 765 test scripts took about 5 hours
- automated testing of web site will be
  - opportunity
  - challenge

research in future

- web testing must validate
  - not only hyperlinks
  - but also web technologies, such as
    - JAVA
    - Active X
- our purpose are
  - validating E-business website, such as
    - broken pages
    - faulty images
    - CGI-Bin error messages

conclusions

- approach based on statistical usage testing
  - provide
    - complete testing coverage and quantitative analysis
- it has many benefits such as
  - helping in testing plan
  - allocating testing resource
  - generating test scripts automatically
  - reaching the maximal testing coverage

A System Framework for Ensuring Link Validity under Web Browsing Environments[+]

Wen-Kui Chang[*] and Shing-Kai Hon

Dept. of Computer & Information Sciences, Tunghai University, Taiwan ROC

Abstract:

This paper investigates the issue of link validity under web browsing environments. The rationale of statistical usage testing is employed and adopted for the web-based certification environments. All possible navigation paths are first formulated to represent a usage model with the Markov chain property to denote all components for a Webster, which is then used to generate test scripts statistically. To perform link validation, tracing first on these test scripts, and then analyzing failure information based on the Markov chain theory. The recommended framework will provide certification result quantitatively. The proposed mechanism is not only feasible and systematic on certifying effectively hyperlinks, but also effective highly on identifying any invalidate node in a navigation structure. A real application of the proposed approach to a web site will be demonstrated quantitatively through a certification tool ToolCertify.

Keywords: statistical usage testing, usage model, Markov chain, software quality certification, software reliability

1. Introduction

Web applications are increasingly developed in many domains but so far most were implemented by the ad hoc approach. Furthermore, various quality metrics become progressively more difficult to measure and mange as the size and scope of the developing applications grows. In this research, factor of link validity, among these typical non-functional characteristics, will be deeply investigated. Essentially link validity includes various characteristics such as correctness, relevance, completeness and integrity implicitly, which will affect heavily effectiveness and efficiency on information retrieval and browsing process.

For most web applications, it is unlikely that application developers validate all possible navigation paths. In this paper, a framework of quantitative certification on link validity will be proposed to ensure that all linked paths provide the consistent and reasonable information streams and appropriate contexts.

In this research, the rational for statistical usage testing is investigated and employed to certify links for a website. In principle, a software usage model is first established, which characterizes various operational uses of a software system. Suppose that an operational use is

---

[+] This research is supported by the National Science Council of ROC (Grant NSC89-2213-E-029-00): A study on the establishment of software reliability certification process.

[*] Correspondence author: Add: Software Engineering Laboratory, P.O. Box 5-809 Tunghai University, Taichung Taiwan 407 tel.: +886-4-3598915; fax: +886-4-3591567; email: wkc@mail.thu.edu.tw

a skeleton for the intended use of the software in an intended environment. Thus, all possible operational uses of a software system will constitute a population with a huge size. If a usage sample of test cases is drawn statistically from the usage population, performance on this sample may then be used as a basis for the evaluation of software quality.

Under the web-browsing environments, all possible navigation paths are first formulated to represent a usage model with the Markov chain property that is then analyzed and used to generate test scripts statistically. The proposed mechanism is not only systematic in certifying effectively hyperlinks, but also efficient highly for those complex information structures. A real application of the proposed approach to a web site will also be demonstrated quantitatively through a certification tool ToolCertify.

2. Web links and information structure

2.1Web linking

In general, web links can be categorized into three types [1]:

1)Structure link

Structure links allow users to understand the scale of information space and help users navigate effectively all information structure, which may be in the forms of linear structure, hierarchical structure, network structure, or matrix structure [1]. The main benefit of such kind of links lies that they may help users to follow the guide and view each page quickly.

2)Associative link

An associative link represents semantic relationship between information elements and is based on the meaning of different information components. More specifically, the items linked by an associate link are conceptually related and they may exist independently. In fact, these relationships provide the essence of web browsing.

3)Referential link

A referential link provides a link between an item of information and an interpretation of that information. In other words, the existence of the item at one end of a referential link is due to the appearance of the other. Essentially, such kind of links is established with the purpose of guiding user to view the interpretation of the links.

Some web sites are found that they were run of a mechanism for differentiating between the above links types. Hence, users were not readily able to distinct the structure of the information space from the content of the information space, thus making web navigation somewhat difficult.

Among the three kinds of links as listed above, validity certification of associate links is a relatively complicate task due to the semantic association. In this investigation, both structure links and referential links will be highly investigated at the present stage.

2.2Link validity

Specifically, link validity includes validating various characteristics such as correctness, relevance, completeness and integrity implicitly [1], which will deeply affect the effectiveness and efficiency on information retrieval and browsing. However, it is not possible to assure complete link validity for practical applications, since information structure on web applications is much related to the following facts:

1)Link itself may be invalid.

2)Links between the adjacent concepts are missed.

3)Too many links cause information chunks to be meaningless.

4)Information structure is hard to maintain due to invalid design.

As to developing large-scale web applications, the above respects are more important. Application developer must ascertain link validity as much as possible to avoid the above situations. Accordingly, validity on information structure will be reached easily.

2.3Link correctness

In this paper, link validity will be limited to the most significant characteristics — correctness, which may further be considered in two aspects:

1)Content correctness

To acquire content correctness, it is highly dependent on the semantics of applications. Thus, this kind of correctness validation is better treated for special application domains.

2)Linking correctness

Functionally, link correctness affects not only completeness of the retrieved information but also comprehension for the whole information structure. Besides, link timeliness plays a vital role on linking correctness. Time-delay on a linking process may usually cause failure link. Moreover, other performance of network channels such as network loading, bandwidths, server's capability, etc. will influence validation performance for linking correctness.

Thus, it is a considerable issue to pinpoint linking correctness while developing large web applications, which is the principal interest of this research.

2.4Information structure

There is a world of factors impacting on the structure in which web application is represented, implemented and used. In general, employing appropriate information structure will make the most of the support for underlying concepts of accessibility and manipulation of information. However, different structures have different impacting on the development process.

In general, there are four types of information structure as following [1]:

1)Linear structures

Linear structure can be used to hold the sequential structure of legacy data and has been used in a number of ways within hypermedia application. Web application developer sometimes implements linking structure by binding a series of nodes together in a sequential way. These are commonly used as instructional aids. The implicit guide-tour can repeat the way an instructor organizes the presentation of information in a logical progression and are useful when such web application is being used to provide procedural instructions.

2)Hierarchical structures

Hierarchical structure can be used to retain the original structure of the information implemented in a web application, and replicated within a web database by browsing these hierarchical links. User can go to application index and select a link point within the information space to surf in the same way as the original web structure.

Conceptually, every node in hierarchical applications is part of single hierarchy. That is,

the root node is usually the starting point for a user, and from this entry point all nodes within application are reachable. Accordingly, any user can catch the whole structure of web site efficiently and access the wanted data quickly.

3)Network structures

Network structure is composed of associative links, which are of semantic or realistic nature and are non-sequential. They bind regular or related concepts together within the information space. It is noted that one of the main advantages of web application lies that using associative links will make effective browsing among the information space.

4)Matrix structures

Occasionally categorization of information is not singly dimensional. However, matrix structure can capture the multidimensionality of various sources of information. Consequently, it is appropriate to use matrix structure as top level of information structure represented for web application, providing an indexing mechanism to lower levels of comprehensive information.

3. Software quality certification

3.1Statistical usage testing

Recently statistical usage testing [2,3,4,5] has been justified and widely applied to software quality certification. Conceptually, the rationale of statistical usage testing lies in the fact that the failures occurred most frequently in practical use will be found early during the test cycle.

In essence, the main benefit of statistical usage testing is that it makes use of statistical inference techniques to compute probabilistic properties of the testing process, such as reliability, or mean time to failure (MTBF) for software quality certification.

3.2Software usage model

In essence, a software usage model characterizes various operational uses of a software system. Suppose that an operational use is a skeleton for the intended use of the software in an intended environment. Then, all possible operational uses of a software system will constitute a population with a huge size. If a usage sample of test cases is drawn statistically from this usage population, performance on the sample may then be used as a basis for the evaluation of software quality.

Whittaker suggests [6] that software testing is rather suitable to be treated as a stochastic process and, its usage be modeled by a finite state, discrete parameter, time homogeneous, irreducible Markov chain. In this paper, Markov approach will be employed for usage modeling [7].

3.3Markovian usage model

Represented by the notation of Markov chain, a usage model consists of all usage states that are connected by links. These links indicate all possible stimuli and responses with a probability index. The probability represents the likelihood of choosing one link from a usage state to the other. Furthermore, test scenarios, generated randomly as a sample of the population, shows some possible usage paths that will traverse the usage model from the start state to the termination state.

Furthermore, the generated test scenarios are formulated as a Markov chain by the

following facts [7]:

1)Occurrence of the current state depending on the previous state only.

2)All usage states are incompatibility.

3)All probabilities emerged from each state are summed to one.

3.4Statistical usage testing procedure

In principle, software quality certification by the usage testing approach may be performed in the following [3,8]:

1)Building a usage model that defines all possible events and their transition distribution.

2)Generating test cases statistically by the associated distribution.

3)Executing the test cases.

4)Collecting performance information and inter-failure data.

5)Certifying the software by the reliability evaluation model.

4. Certification on link correctness

4.1Demonstration description

In this paper, a Webster IDS Software [9] as shown in Fig.1 is illustrated to perform the proposed Webster test. As pointed in the previous section, four structures such as linear, hierarchical, network and matrix are usually used to represent structure of information space. Besides, more than one kind of information structure may be employed in a web application. In this IDS Software example, information structure is recognized as being hierarchical, which can be observed from the navigation structure as shown in Fig.2. Since hierarchical structure often makes use of structural links, there are many structural links implemented in the IDS Software Webster. As stated previously, the main advantage of hierarchical structure lies that it may retain original structure of information contained in an application system. Since it is a commercial Webster, in particular, links established by hierarchical structure may benefit users to browse the whole navigation paths to look up the desired products. In addition, user can go this web site's index and select point within information space to browse in the same way as it was built.

Fig. 1 IDS SOFTWARE example homepage

4.2Usage model

To begin with statistical usage testing, a navigation structure must first be built to delegate all possible navigation paths for a web site. However, it is somewhat difficult to collect all complete browsing paths as the potential paths may go divergently without termination. Thus, a macro structure to integrate some relevant nodes together is sometimes established first in order to simplify certain portion of the whole navigation structure. Nevertheless, the nonlinear information structure of web application usually makes the linking process severely complex. Fortunately some tools such as CAPBAK/WEB [10], Microsoft FrontPage, etc., may use as a "test wizards" to create all links, all buttons, and FORM-content tests passages.

In this research, a navigation map established by the FrontPage is shown in Fig.2. Suppose that every hyperlink as a browsing state and the homepage as the starting state. While using IDS Software, user clicks every possible hyperlink from the starting node until he finally reaches the exit node to terminate his browsing. As a result, these possible operational patterns formulate a Markov chain.

As for illustration, the navigation map can be restructured as a usage model in Fig.3, in which the dotted circle A and B in Fig.3 is expanded to Fig.4 and Fig.5, respectively.

Although the usage model of IDS Software is categorized as a hierarchical structure, some hyperlink paths have no termination, since the links expand endless. To resolve such kind of the problem, consider the implicit hierarchical structure constituting a tree structure with the root (site homepage) as the level 1. Accordingly, the whole tree structure of IDS Software consists of 6 levels, as demonstrated in Fig.2.

Furthermore, some hyperlinks, to avoid cognitive overload and spaghetti links, may be assumed to terminate at some level depending on the implied content. For instance, square nodes in Fig.3 signify the true termination nodes, while triangle nodes imply that they further have out-coming links, but are assumed to be terminated for the sake of simplification.

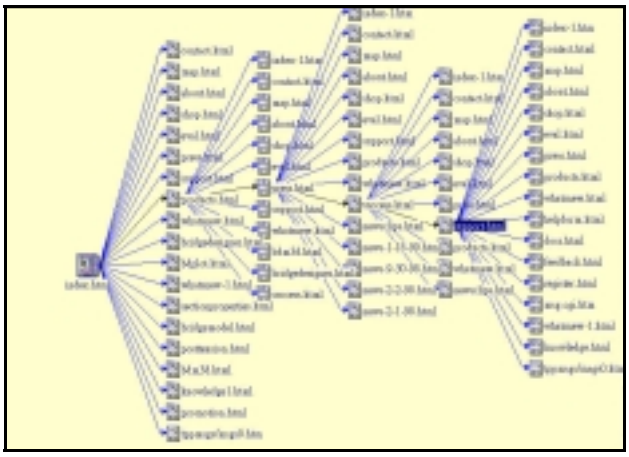


Fig. 2 the navigation map for IDS SOFTWARE example

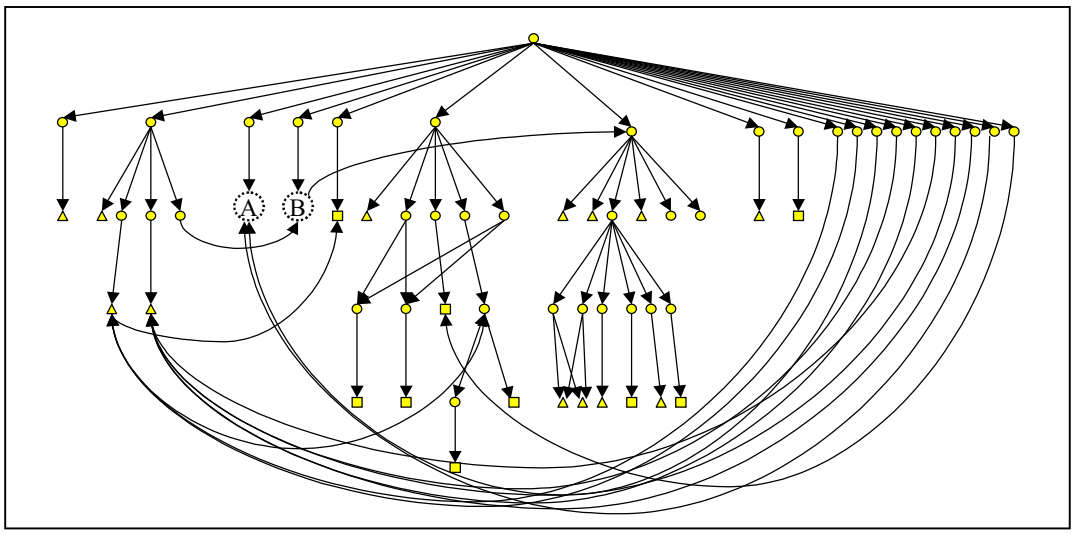


Fig. 3 the usage model for IDS SOFTWARE example

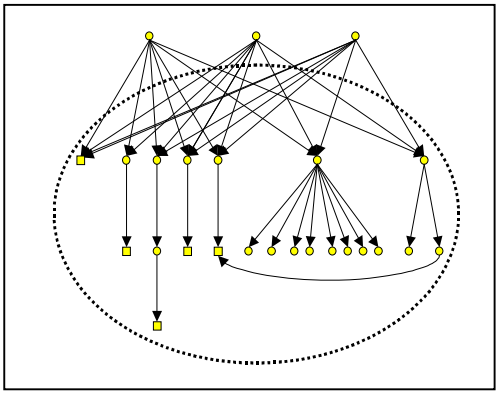


Fig. 4 The expanded usage model for the dotted circle A in Fig 3

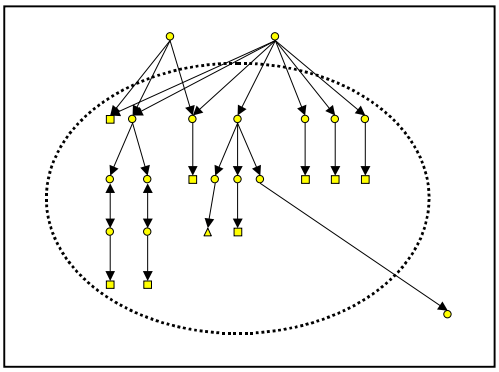


Fig. 5 The expanded usage model for the dotted circle B in Fig 3

4.3Model analysis on the navigation structure

As shown in the Table 1, the built usage model for the IDS Software website includes 78

states in all, that will play a fundamental framework for link certification on web applications. With the aid of ToolCertify certification tool [11], the analyzed report may be summarized as in the Table 1.

Table 1. Analysis report for the IDS SOFTWARE navigation structure

Number Of Nondeleted States	78
Number Of Active Arcs	154
Expected Script Length	4.586
Least Likely State Coverage Expected At	652.909
Least Likely Transition Coverage Expected At	652.909

4.4Certification on link validity for web applications

1)Test scripts

Theoretically, the number of test scripts required to browse all 78 navigation nodes and their associate links is estimated as 653 (652.909). However, after executing these generated 653 test scripts in practice, several link errors were found as listed in the Table 2. Thus, the number of test scenarios required to achieve the complete coverage in the demonstrated structure is increased to 765.

2)Failure analysis

On observing the failure information, one source of link invalidity may be deducted among those found several link errors with error message “HTTP 404 Not Found” appeared in a browser. Originally the failure comes from the nodes [IDS Tech Support Policy] that is further expanded from the node A. The failure path is on [Support Center] field that has subclasses [Help Form]. The failure hyperlinks [IDS Software Official Tech Support Policy] is in [Help Form].

In the Table 2, it is noted that the found invalidate node is the principal cause that made 10 distinct failures among the 765 testing browsing paths.

Table 2. Failure analysis for the IDS SOFTWARE example

Failure ID	Mean First Passage	Probability of Occurrence
1	0.000291	0.001307
2	0.000291	0.001307
3	0.000291	0.001307
4	0.000291	0.001307
5	0.000291	0.001307
6	0.000291	0.001307
7	0.000291	0.001307
8	0.000291	0.001307
9	0.000291	0.001307
10	0.000291	0.001307

3)Certification result

By the analysis result derived from the Markov usage model, partial certification may be summarized as in the Table 3, with 95% and 99% confidence intervals. Some corresponding test scripts are shown in Table 4.

Table 3. Certification result for the IDS SOFTWARE example

Script #	Result	MTTF	R	C=95%	C=99%	% States Certified	% Arcs Certified
30	Fail	29.99997	0.966667	0.011531	0.015154	0.8719	39.74359
31	Pass	30.99997	0.967742	0.011165	0.014674	0.862504	41.02564
33	Pass	33.00003	0.969697	0.010498	0.013797	0.849346	42.30769
34	Fail	17.00000	0.941176	0.013798	0.018135	0.849319	42.30769
232	Pass	116.0001	0.991379	0.002051	0.002696	84.61539	79.22078
233	Fail	77.66654	0.987124	0.002113	0.002778	84.61539	79.22078
234	Pass	77.99983	0.987179	0.002104	0.002766	84.61539	79.22078
576	Fail	64	0.984375	0.000968	0.001272	97.4359	96.75324
584	Fail	58.40002	0.982877	0.000965	0.001268	97.4359	97.4026
762	Pass	76.20015	0.986877	0.000739	0.000972	98.71795	98.7013
763	Pass	76.29996	0.986894	0.000739	0.000971	98.71795	98.7013
764	Pass	76.40002	0.986911	0.000738	0.000969	98.71795	98.7013
765	Pass	76.50001	0.986928	0.000737	0.000968	100	100

Table 4. Test Script for the IDS SOFTWARE example

Test Script: 30		Test Script: 233	
1	Software Not Invoked 1. to homepage homepage	1	Software Not Invoked 1. to homepage homepage
2	2. click to SUPPORTS WINDOWS95 SUPPORTS WINDOWS95	2	2. click to What's New2 What's New2
3	3. click to Help Form Help Form	3	3. click to Help Form Help Form
4	4. click to IDS Tech Support Policy IDS Tech Support Policy	4	4. click to IDS Tech Support Policy IDS Tech Support Policy
5	5. click to Software Terminated Software Terminated	5	5. click to Software Terminated Software Terminated
Test Script: 576		Test Script: 584	
1	Software Not Invoked 1. to homepage homepage	1	Software Not Invoked 1. to homepage homepage
2	2. click to SUPPORTS WINDOWS95 SUPPORTS WINDOWS95	2	2. click to SUPPORTS WINDOWS95 SUPPORTS WINDOWS95
3	3. click to Help Form Help Form	3	3. click to Help Form Help Form
4	4. click to IDS Tech Support Policy IDS Tech Support Policy	4	4. click to IDS Tech Support Policy IDS Tech Support Policy
5	5. click to Software Terminated Software Terminated	5	5. click to Software Terminated Software Terminated

4.5Discussion

1)Challenges

Without a doubt, if the states in a usage model are so many, the number of test scripts will become quite huge. Moreover, more test scripts will imply more execution time. Similarly, the longer the navigation path, the more testing execution time. In this research, manual execution of 765 test scripts took about 5 hours. In particular, every test script may take 5 (4.586) steps in average for browsing, according to Table 1. Thus, automated testing of web site will be an opportunity and a challenge to deal with such issue.

Another challenge for web testing comes from the browser itself. Though, at the current stage, there are only two existing players, i.e. Internet Explorer and Netscape Navigator and thus end users may operate easily, the testing process become somewhat complicated. In fact, different browsers are sometimes endowed with different features and support different techniques. Consequently, complexity on web testing is greatly increased. Furthermore, performance of a web browser is highly correlated with various factors such as: operating systems, platforms, machine configurations, etc. For instance, testing on the different configuration combinations can quickly become a huge and insurmountable task.

2)Research in future

In general, web testing must validate not only hyperlinks but also web technologies, such as JAVA and Active X. In particular, e-business applications propagate with extreme rapidity in recent years. It is more important to control quality of e-business websites. E-business webs site with poor quality such as: broken pages, faulty images, CGI-Bin error messages, etc. can always overload the user.

In brief, the combination of web site complexity and low quality is potentially fatal to company business. Those users will quickly depart for a different site; and they will probably leave with a bad impression. Hereafter, validating e-business website is the future direction of our research.

5. Conclusions

For software quality certification, a framework of statistical usage testing is investigated in this paper, and a mechanism for certifying all possible navigation links is developed.

The rationale of statistical usage testing is a complete and systemic method, which is rather than the general ad hoc approaches. The disadvantage of ad hoc approach lies that it cannot guarantee that all navigation paths and components have been tested. On the contrary, the suggested approach based on statistical usage testing may provide complete testing coverage and quantitative analysis as well.

The proposed mechanism has been shown to be effective for certifying quickly link validity in website applications. It has many benefits such as: helping in testing plan and allocating testing resource, generating test scripts automatically, and reaching the maximal testing coverage. Hence, the proposed testing framework will be more practical and efficient than traditional testing.

In future research, validating hyperlinks and web technologies as well in a single Webster will be highly emphasized. Meanwhile, an automated test environment for executing the generating test scripts in Webster applications will be investigated and developed. Besides, some common techniques in e-business Webster such as: form, buttons submit, CGI-Script, etc. need to be reviewed detailed from the viewpoint of quality assurance.

References

1. Lowe, David, Wendy Hall: Hypermedia and the Web—An Engineering Approach, John Wiley and Sons, ISBN: 0471983128, 1999, pp. 60-65.
2. Whittaker, J. A. and M. G. Thomason: "A Markov Chain Model for Statistical Software Testing," IEEE Transactions on Software Engineering, October 1994, 20 (10), pp. 812-824.
3. Walton, G. H., J. H. Poore and C. J. Trammell: "Statistical Testing of Software Based on a Usage Model," Software Practice and Experience, January 1995, 5(1), pp. 97-108.
4. Chang, Wen-Kui: " A Quadratic Programming Approach to Usage Modeling for Software Reliability Certification," Tunghai Journal, July 1997, Vol. 38, pp. 65-78.
5. Chang, Wen-Kui, Stephen Twu and William Teng: “Ensuring Functional Test Coverage For Avionics Control Applications Through Statistical Usage Testing,” FESMA'99 - 2nd European Software Measurement Conference 4-8 Amsterdam, The Netherlands, October 1999, pp. 261-268.
6. Whittaker, J. A. and J.H. Poore: "Markov Analysis of Software Specifications," ACM Transactions on Software Engineering and Methodology, Vol. 2(1), 1993, pp. 93-106.
7. Chang, Wen-Kui, Che-Po Wang, and Ching-Chun Fu: “An Investigation on the Markovian Property of Usage Model for Software Quality Certification,” Proc. of the Chinese Institute of Industrial Engineers National Conference, Taiwan, December 1999, pp. 129-135.
8. Chang, Wen-Kui, Che-Po Wang, and Ching-Chun Fu: “A Study on Usage Modeling for Software Reliability Certification via Prototyping Simulation,” The 3rd Symposium on Reliability and Maintainability, Taiwan, October 1999, pp.279-285.
9. IDS SOFTWARE website: http://ids-soft.com, 2000.
10. Software Research, Inc.: CAPBAK/Web [IE] Ver. 1.5 User Manual, San Francisco USA, 1999.
11. Q-Labs: ToolCertify User Guide, Version 4.0, 1999.

QW2000 Paper 9M1

Mrs. Hong Guo, Prof. Graham King, Ms. Margaret Ross & Mr. Geoffe Stable (Southampton Institute)

Using BOOTSTRAP to Improve the Management of Software Process in a Virtual Software Organization?

BACK TO QW2000 PROGRAM

Presentation Abstract

The aim of this paper is to introduce a new development trend in software industry and to present the initial findings in evaluating specific requirements for software process improvement in Virtual Software Environments (VSE). Surveys were designed with the purpose of identifying key issues of successful software development in Virtual Software Organisations (VSO).

The results of the survey indicate that a suitable process approach for managing and improving the software development process is also the most important issue in achieving product quality improvement in VSE. This issue has not been addressed by existing software process models. Research suggests that a software process model for Teleworking software development based on the modification of the BOOTSTRAP process assessment and improvement approach may be generated. To provide the necessary groundwork in developing such a model, the focus of this paper is the analysis of the BOOTSTRAP modelËs methodology, with the identification of the issues that are not covered in BOOTSTRAP. As a consequence of this study, the possible extensions are suggested.

About the Authors

Hong Guo is a Ph.D. research student at Research Centre of Computer Systems, Southampton Institute, in UK. Her research interests include software development, software process assessment and improvement, quality control. Hong Guo received a MSc Diploma from Tsing-hua University, Peking, in 1987 and a B.Sc. from Luoyang Institute of Technology in 1982, both in Solid Mechanics in P.R.China. She was a senior lecturer at university in P. R. China, teaching and research in programming languages, software design and development, solid mechanics analysis and finite element analysis.

Graham King is Professor of Computer Systems Engineering at Southampton Institute in U.K.. His expertise lies in Data Compression & Communication, mobile radio condition monitoring, VLSI systems and parallel computation. In the course of having succesfully completed numerous research programmes for commercial and non-commercial research sponsors, he has over 80 publications (conferences, journals, books) and he is named inventor in a project for which patent protection is

currently being sought.

BACK TO QW2000 PROGRAM

Process Improvement In Virtual Software Organisation with BOOTSTRAP

Hong Guo, Graham King
Margaret Ross, Geoff Staples

Southampton Institute, Southampton, UK

The world is changing

The world is in Information Technology Age

- Rapid communication
- Global information accessing
- New product delivery channels
- New kinds of business
- New kinds organization
- Flexible working - Teleworking

Software industry is the enabling force

New challenges in software industry

Demand for software product

- Continues increasing in quantity, complexity
- Deliver on time, Cheaper, Better quality

Supply and resource

- Demand is increasing much faster than supply
- Shortage of software professionals

Global competition

- Competitive pressures are changing the nature of software industry

Responses to the challenges

- Geographically distributed projects
- Distributed software development environments
- Virtual Software Organization
- Virtual Corporations

Teleworking

- Telecommuting, Home working ...
- Working from home, Telecentre, Telecottage
- The use of computers and telecommunications to change the accepted geography of work
- Based on or facilitated by network technology
- **Benefits**
  - Increasing flexibility & convenience
  - Increasing productivity & job satisfaction
  - Reducing costs with less stress
  - Solving transport problems
  - Improving quality of human life

Virtual Software Organization

- Organizations developing software involve some of their software engineers working from distance - Teleworking
- Software project is implemented in geographically distributed environments
- **Benefits**
  - Savings in staff training & office overhead
  - Improved product quality
  - Shorter time to market
  - Extensive opportunity to exploit competent human resource
  - Stronger competence

Software measurement and Quality

- Measurement of products is absolutely fundamental to the engineering process
- It is convinced that measurement as practised in other engineering disciplines is IMPOSSIBLE for Software Engineering
- Software process is a set of empirical practices in software development, organisation and management
- Software process assessment and improvement is an alternative to improve software quality

Improving Software Quality by Well-defined Process Management

- CMM
  - The first SPI model developed by SEI at Carnegie Mellon University in U.S.
- SPICE - ISO/IEC TR 15504
  - Emerging international standard for software process assessment, capability determination, and improvement developed by the SPICE project
- BOOTSTRAP methodology
  - The European process assessment and improvement approach developed by a European consortium, and BOOTSTRAP Institute

Process Management & Improvement in distributed environments

- A well-defined and managed software process is also most crucial as a guarantor of software quality
- Little attention has been paid to the support of large distributed projects and its development environments
- A major requirement for the creation and management of virtual software project and virtual organization is distributed project coordination and communication

Objective of the study

- To address the needs of virtual software organisation and its distributed project
- To adopt the existing process modeling to take account of the circumstances surrounding geographically distributed software environments

BOOTSTRAP methodology 3.0

- The purpose is to provide a European approach for process assessment based software process improvement
- The goal of the improvement is to apply good software engineering principles in the software development process
- Include
  - A process model
  - An assessment process
  - A mechanism to score the process and present the results
  - Guidelines on process improvement
  - Assessor training program & Licensing policy
  - A database of assessment results

Architecture of BOOTSTRAP

- Adopt three dimensions
  - Process dimension: *a set of good processes & activities*
  - Capability dimension: *a rating framework for each process*
  - Technology dimension: *Advantage over other models*
- The objectives of software process management are to produce products according to plan while simultaneously improving the organisation's capability to produce better products
- Achievement of process implementation and its capability are judged by objective evidence

The Process Model in BOOTSTRAP

- Used as basis to perform an assessment
- Fully compatible with the requirements of ISO TR 15504
- Structured as a tree that identifies
  - Process categories
    - *Organisation*
    - *Technology*
    - *Methodology*
  - Process areas
  - Processes and best practices

Issues Specific to distributed software project

Major requirements are:

- Distributed project coordination management:
  - Different people working on a common project agree to a common definition of what they are building, share information and mesh their activities
  - They must coordinate their work so that it gets done and fits together
  - How to handle the effects of changes to software artifacts
  - How to constrain the effect one change has to other parts of the software system
  - How to ensure proper flow of change information, e.g. to notify people involved in the process of the changes that concern them

Issues Specific to distributed software project

- Distributed Software Configuration Management
  - A communicator that establishes relationships with all the project activities - providing technical information about specification, document, manual, code, test results, and so forth, provided to the project team
  - To ensure that changes to the software are managed
    - Stronger change control management system
    - Stronger configuration management system:
    - Stronger documentation management system:
      Documentation becomes the most used means of communication
    - An effective version control system

Issues Specific to distributed software project

- Effective communication
  - A virtual software team must be able first to communicate in a distributed fashion, second to communicate effectively along the spectrum from real time to non-real time
  - Most popular communication techniques

Real time	Near Real time	Non Real time
Telephone	E-mail	Mail
video conference	Voice-mail	GroupWare
Teleconference	Pagers	Bulletin-Board
Networking	Fax	Database

- Most effective communication: Face-to-Face meeting

Requirement of Support System for distributed software projects

- Integrated automated support systems for
  - Co-ordination
  - Collaboration
  - Communication
  - Project monitoring
  - Configuration management
- Intranet - *the most enabling factor*
  - Quickly transferring large amounts of data from one site to another, with fairly secure
  - Faster communication services with integrated support capability
  - Easier channel to deliver computer-based training
  - Cheap and easy-to-construct

Extending BOOTSTRAP for VSE

- Establish management system for technique support system at organisational and project level
- Establish communication systems
- Set up an Intranet system to support information sharing, and project management
- Adopt automated configuration management tools
- Adopt automated version control management tool
- Adopt documentation tools to develop and maintain the project documents
- Establish and maintain a problem report repository to enable continually process improvement

Conclusion

- Formation of VSE & VSO is an inevitable trend caused by changes in the social structure induced by the global network and communication technology
- Present the initial findings of an evaluation of the specific problems and requirements for software process improvement in VSE
- Analysed the methodology of BOOTSTRAP to provide the background knowledge
- Indicated the suggested extension in BOOTSTRAP concerning its application into distributed software environment

Process Improvement In Virtual Software Organisation with the BOOTSTRAP Methodology

Hong Guo, Graham King, Margaret Ross, Geoff Staples
Research Centre for Systems Engineering
Southampton Institute, Southampton, UK

Abstract

The aim of this paper is to introduce latest development trends in the software industry and to present the initial findings in evaluating specific requirements for software process assessment and improvement in Virtual Software Environments. Surveys were designed with the purpose of identifying key issues of successful software development in Virtual Software Organisations.

The results of our survey indicate that a suitable process approach for managing and improving the software development process is also the most important issue in achieving product quality improvement in distributed development environments. This issue has not been addressed by existing software process assessment and improvement models. Research suggests that a software process approach for Teleworking software development environments may be based on the modification of the existing process models. To provide the necessary groundwork in developing such an approach, the focus of this paper is the analysis of the BOOTSTRAP model's methodology, with the identification of the issues that are not covered in BOOTSTRAP. As a consequence of this study, the possible extensions are suggested.

Key Words: BOOTSTRAP model, Teleworking, Virtual Software Environments (VSEs), Virtual Software Organisation (VSO), Software Process Improvement (SPI), Software Quality.

1. Introduction: *Teleworking, Virtual Software Environments, Virtual Software Organisation*

It is becoming increasingly harder to justify completing a software development project inside company walls [1]. As the software community begins to appreciate the economy of merging diverse development skills and domain expertise, and as communication media become more sophisticated, the cost, time to delivery and competitive pressures are pushing more companies toward geographically distributed projects: It is becoming less and less cost-effective or competitive to develop a software product within the same building, company, or even country. Meanwhile, the structure of society has steadily changed into a new era of Information and Communication Technology, which means we are being able to work and live wherever we like. The distance boundary between home, organisation, and country is becoming less and less important for everyone in the world. Now work may traverse time and space instead of people having to travel, because the power of information technology (IT) existing today has made it possible to remotely access information across organisational and national boundaries. All of which is therefore allowing groups from different locations and cultures and with different expectations and goals to come together as a global software development team.

The trend towards distributed software development is induced by the changes in the social and economical environment of software engineering. The major driver of this trend is the changes in the nature of the software industry:

- *Supply and Demand:* The demand for software services has historically outpaced the supply of people who perform them [1]. There is a common recognition that software development is a very competitive and profitable business. More importantly, software is the enabling force of the changes happening today, and is becoming the key element that governs competition at the threshold to the 21st century. As society improves, the demand for software with better quality at lower cost will continues to increase as a faster pace than the supply, whilst the software development projects get continually larger and more complicated. Software is now a huge global industry faced with a great challenge in attracting and retaining highly skilled software professionals.

- *Cost and Resource:* Labour costs have escalated as companies compete for resources. As equivalent resources became available at a lower cost – especially overseas – work started to migrate outside the company. In essence, the supply and demand of software professionals was driving costs, which in turn was driving the migration of software development outside company walls. Today, many companies are finding it economically attractive to have some of their employees working remotely in full or part time. These staff would otherwise unavailable to them in bringing together the scarce competencies and resources to meet development requirements. This is therefore necessitating the emergence of global distributed software development projects.
- *Global Business and Market:* Competition in business today has significantly changed. Competitors are now all over the world. Customers may be global as well. The rapid development in global networking, with increasing power at low-cost nature has not only caused an explosion in e-business and e-commerce activity, which creating a paradigm shift in the business world, but also provided organisations with extensive opportunities to develop an additional channel for doing business. Putting business "on-line" is opening up a whole new world of possibilities such as enhanced service levels, increased efficiency, reduced costs, faster delivery of products like software, shorter time-to-market and wider market reach. Meanwhile, computers, especially lower end types like PCs, are increasingly less expensive, contain more computing power, and include more features and functions than they did even one year ago. All of this is particularly beneficial for organizations developing software and promotes the growing formation of Virtual Software Environments and Virtual Software Organizations. On the other hand, this means new challenges. Organisations will have to keep pace with the rate of technology change to remain globally competitive. For those who produce software, this means they can no longer pass off a product with poor quality or average performance. It is no longer a seller's market. Customers will start demanding software that meets more of their particular need.

The pace of change is now extraordinary. Not only new kinds of organisation and new ways of organising are being fostered, the new way of working - Teleworking, is also developing very rapidly as an increasingly normal way of working [2] that can greatly expand the pool of available workforce and improve the quality of human life. The product life cycle increasingly takes place within so-called virtual team [3] and virtual organisation [4,5] that requires close co-operation across a variety of distributed locations or national boundaries. The number of new concepts: Teleworking software engineer; Virtual Software Environments (VSEs); and Virtual Software Organisation (VSO) are also being generated as a representation of new way of working and a novel form of organisation [6].

A characteristic of VSEs and VSO is the intensive use of new technologies to provide support for interactions between its members who are operating in geographically distributed locations, whilst Teleworking is the use of computers and telecommunications to change the accepted geography of work that is based on or facilitated by the world-wide interconnected network technology. The creation of the VSE and VSO is a modern approach to economies of scale. It provides the software industry with a great opportunity to exploit highly competent human resources all around the world to dramatically increase its global competitive power.

Competition in business today is fierce. Firms have to change just about everything they do simply to survive. The trend towards global or virtual software development is inevitable as practical response to the changes, and increases the magnitude of problems to be addressed. For organisations developing software, the most important goals were still getting control of software cost, schedules, and quality. However, the problems associated with the poor quality software products has not only given a bad name to software organizations, but also increasingly became a rigorous obstruction in the way software engineering developing. As such, the problem must be resolved, as a major challenge now facing the software industry. Unfortunately, there is no measure that comprehensively represents programming quality [7], neither is there a sign of a magic new technology to solve these problems [8]. Probably it is because of the nature of software - both finite and incomplete. Software is a finite and incomplete model of an unbounded application in an unbounded operational domain [9]. In order to escape from a continuous software crisis, fundamental changes must occur both in how we manage software development processes and in the way software engineers do their work. A considerable effort in current software engineering has consequently being concerned with Software Process Improvement (SPI) as an alternative to improve software quality.

Over the last decade, software process assessment and improvement approaches have played a momentous role in achieving, evaluating, maintaining and improving software quality in the software industry. A number of process models have been developed and contributed efficiently to a fundamental awareness of the concepts and the importance of adopting a defined and disciplined software process and management system, both at organisational

and individual level in software engineering. A number of methods and tools which support software process modelling, evaluation and improvement are currently available [10].

However, the existing process models are only concerned with the traditional centralised software development environments, little attention has been paid to the support of large distributed projects and its development environments. Given the number of SPI models available, the questions arises as to whether these models can be used or which one of them is the best suited for software development in the geographically distributed environments. A significant difference is that this kind of software development project and team members spans extensive distances and a numbers of countries of varying culture while still operating as if they are at company headquarters. The resulting need for the VSEs and VSO is now a great challenge for project management and requires new techniques for project co-ordination, communication and change control.

The authors explicitly address the problem with the two important issues: software process improvement and managerial activities relevant to the VSE of VSO. The attempt is to adopt existing process modelling to take account of the circumstances surrounding geographically distributed software environments. In doing this, two surveys have been conducted to gain the practical insights into the software industry. The results of the survey revealed that a suitable process approach for managing and improving the software development process is also the most important issue for VSEs and VSO. The issues of supporting large distributed software projects and their special needs with respect to managing and improving the software process have however not been addressed by the existing SPI models. More discussion about the surveys can be found in [11, 12].

To provide the necessary background knowledge in developing such approaches for VSE, a comparative analysis of the existing SPI model has been made in terms of its possible application into VSE [13, 14]. One of the most important existing models, the European software process assessment methodology – BOOTSTRAP, is further identified as a potential candidate in the attempt of the development. The primary focus of this paper is to analyze the methodology of BOOTSTRAP, with the initial identification of the issues specific to VSEs that are not covered in BOOTSTRAP.

2. The BOOTSTRAP Methodology

2.1 Background of BOOTSTRAP

Bootstrap methodology is the most well known European process assessment and improvement approach developed by a European consortium partially funded by the European Commission within the ESPRIT program (ESPRIT Project 5441) [15]. This project ran from 1989 to February 1993, with mission to study investments in technology upgrades and lay the groundwork for European technology transfer standards and common practices. The main goal was to speed up the application of software engineering technology in the European software industry.

The purpose of the Bootstrap methodology is to provide a European approach for assessment based software process improvement. The goal of the improvements is to apply good software engineering principles in the software process. In executing this goal, BOOTSTRAP includes both CMM and the ISO9000 standards, ISO 9000-3 in particular, as its starting points. Since the end of the project, the methodology has been further developed and marketed by the BOOTSTRAP Institute. The latest version of BOOTSTRAP 3.0 has been developed by BOOTSTRAP Institute in September 1997 [16]. The result is to ensure full conformance to ISO 15504, and to align to ISO 12207.

The methodology includes a process model, an assessment process, a mechanism to score the process and present the results, guidelines on process improvement, assessor training program, licensing policy. The Bootstrap Institute maintains a database of assessment results mainly from European companies but also from others in the rest of the world.

2.2 Architecture of BOOTSTRAP

The BOOTSTRAP methodology supports both software process assessment and improvement. It makes an assumption that assessment is a part of the improvement process and provides assessment results as main input for the improvement action planning and as feedback from the improvement activities implemented in the process [15]. Three dimensions are adopted in version 3.0 to create a single set of best-in-class practices for software engineering utilising existing internationally-recognised process assessment methodologies. They are:

- The process dimension provides a set of fundamental processes and activities associated with good software engineering.
- The capability dimension provides a rating framework for each process assessed. The capability of each process assessed is measured within six maturity levels on the basis of a quartile scale dividing each maturity level into quartiles. Thus more detailed and precise results can be obtained, and the quartiles make it possible to compare different assessment results inside the same capability level.
- The technology dimension aims to evaluate the extent to which the process capability is strengthened with adoption of suitable tools for each process.

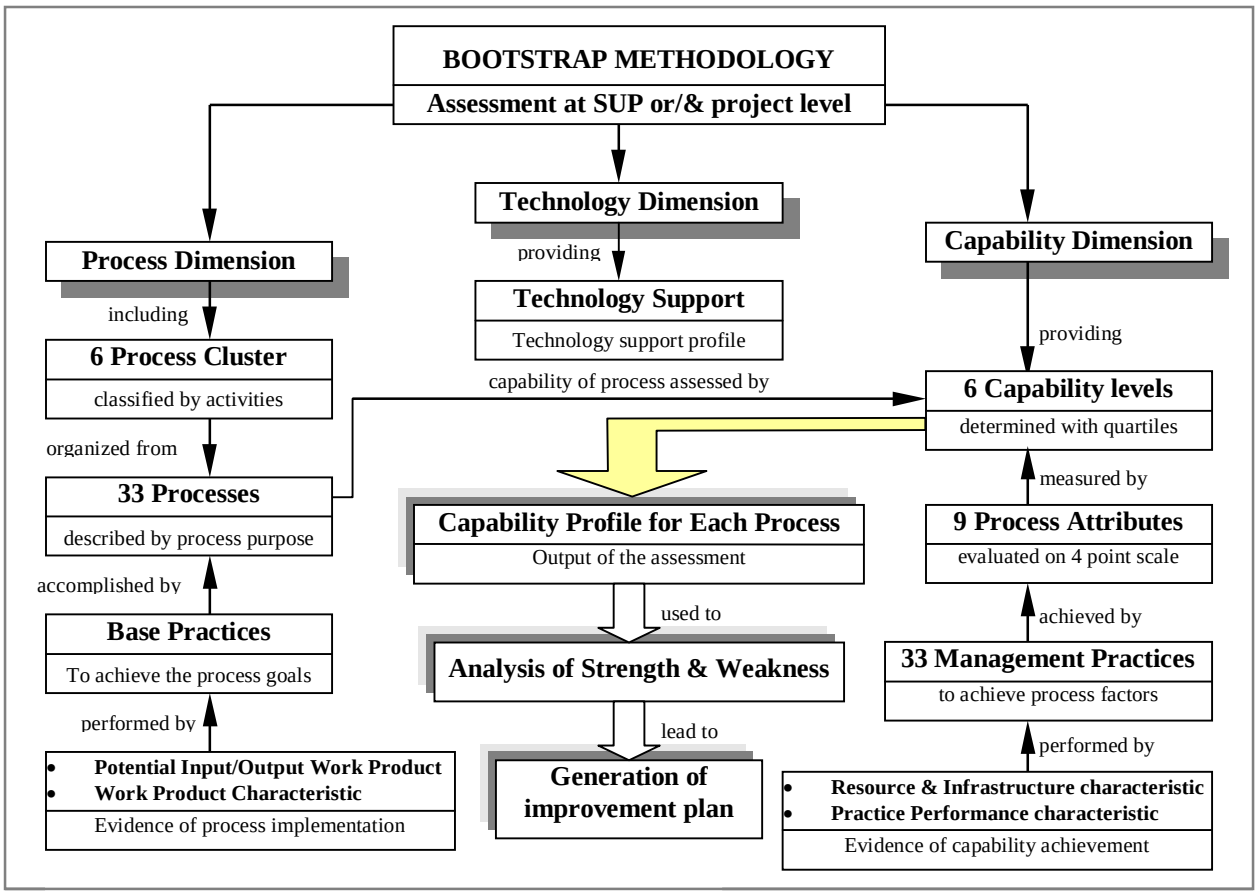


Figure 1 BOOTSTRAP Principle and Architecture

The fundamental elements throughout the BOOTSTRAP architecture are the process purpose in process dimension and process attribute in capability dimension. They are the criteria against which an assessment is performed. The achievement of the performance and the capability of implemented processes are judged and measured by objective evidence of process implementation and capability achievement (see Figure 1). The methodology not only provides an assessment of the current practices, but also provides guidelines for transforming the assessment results into an action plan and gives guidance on prioritising the actions. Version 3.0 also supports software process improvement driven by the organisation's needs and business goals.

One of the unique features to BOOTSTRAP is the assessment is performed at two levels: the Software Producing Unit (SPU) level and its projects level. One of the advantages is to show the capability of single processes as capability profiles and with quartile precision within each level. The output profiles are specific to the BOOTSTRAP methodology, it may be expressed by the absolute maturity level of all the key attributes forming the strengths and weaknesses profile of the SPU or project.

2.3 Process model of BOOTSTRAP

The process model is used as basis to perform an assessment. As shown in figure 2, the processes in the process model are structured as a tree that identifies the following objects: process categories, process areas, processes and best practices.

Like other SPI models (e.g. CMM, SPICE), a software process is a set of empirical practices in software development, organisation and management. The process is defined as a set of activities, methods, practices, and transformations that people use to develop, maintain software and the associated products, whilst the objectives of software process management are to produce products according to plan while simultaneously improving the organisation's capability to produce better products.

However, all these processes and practices defined are originally concentrated on the issues within a single-site development environments. Few existing process assessment and improvement models address the needs of VSEs and VSO. As yet little research has specifically addressed modelling the software process relevant to VSEs, notwithstanding the fact that their intellectual characteristic application areas are frequently the most challenging in terms of integrity and reliability, and those in which a well-defined and managed software process is therefore most crucial as a guarantor of software quality.

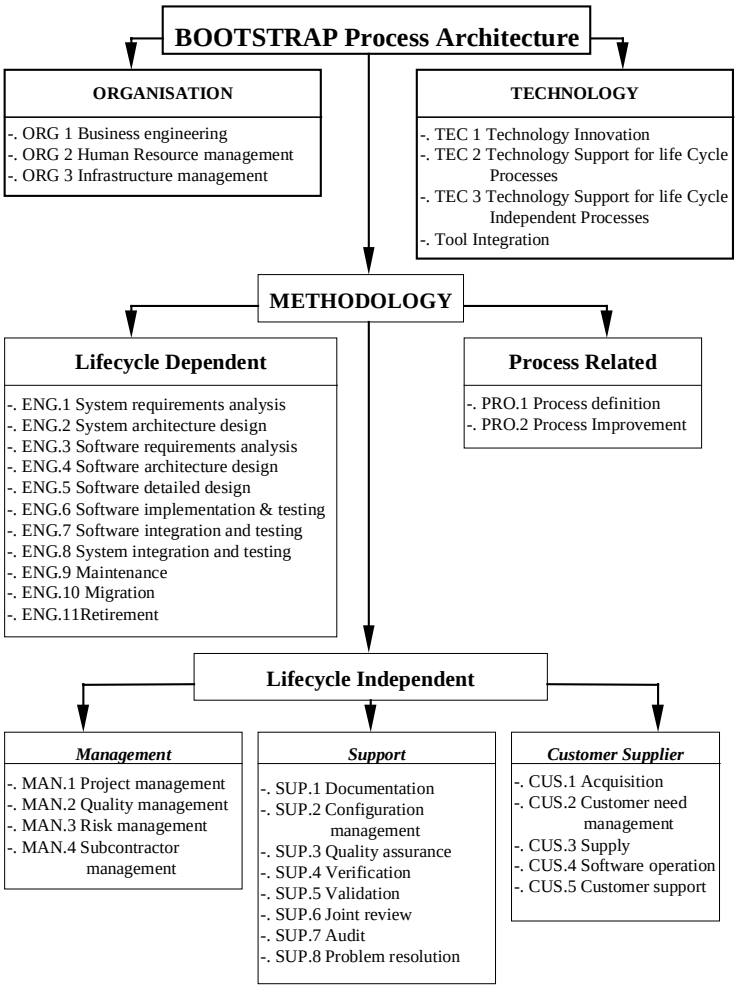


Figure 2 BOOTSTRAP process tree

Based on the surveys and the research we carried out so far, a number of issues specific to the virtual software environments that are not addressed in the existing SPI models were identified [11,12, 6]. One of the most important aspects that must be considered by the distributed software development process is that it requires facilities to support collaboration, communication and coordination among software developers. A significant advantage of BOOTSTRAP over other models is that it contains the Technology dimension in evaluating the process capability, which is the major reason for its selection. Unfortunately, the unique feature of BOOTSTRAP also forms its weakness, since the technology rating in BOOTSTRAP based assessment is indicated as having no influence on the overall maturity or capability in the BOOTSTRAP version 3.0. In the following sections we further describe the issues specific to VSEs in addition to the issues identified and discussed in [6], with emphasis on investigation of the issues relevant to technical support systems.

3. Issues specific to the distributed software projects and its development environments

A major requirement for the creation and management of virtual software projects and virtual organizations is distributed project coordination and communication. This includes the planning and scheduling of projects, execution of projects, coordination of tasks, resolution of competing objectives, achievement of global coherence, change propagation, communication across heterogeneous individuals or groups, and maintenance of access to valid information. Without these and other similar functions, the development process becomes disorganized and the goals of improving competitiveness and productivity cannot be achieved.

3.1 Technology support in distributed project coordination management

Coordination has been defined as the direction of "individuals' efforts toward achieving common and explicitly recognised goals" [17] and "the integration or linking together of different parts of an organisation to accomplish a collective set of tasks"[18]. In software development, it means that different people working on a common project agree to a common definition of what they are building, share information, and mesh their activities. They must have a common view of what the software they are constructing should do, how it should be organised, and how it should fit with other software systems already in place or undergoing parallel development. To build the software efficiently, they must share detailed design specifications and information about the progress of software modules. Overall, they must coordinate their work so that it gets done and fits together, so that it isn't done redundantly, and so that components of the work are handed off expeditiously.

It is clear that achieving a successful software system requires tight coordination among the various efforts involved in the software development cycle. Yet this coordination is difficult to achieve [19] even for the software projects developed in a single-site. Coordinating multiple developers working on a distributed software development project is much more difficult, and gives rise to the following management problems:

- Developers need specific tasks assigned, which must be coordinated to ensure a working system results.
- Developer need to, at times, communicate and collaborate closely, while at other times can independently work on parts of the project.
- Software artefacts (code, designs, documentation etc.) need to be shared and kept consistent.
- Multiple tools must be used to modify artefacts, with some tools supporting close collaborative editing (e.g. via synchronous editing), while others supporting looser collaboration (e.g. via alternate version editing and subsequent merging).
- Progress towards specified goals needs to be tracked, developers need to remain aware of others' work, and complex software systems need to be configured from the constituent, distributively developed parts.
- Developers need to flexibly configure their environments' support for artefact management, communication, and work coordination.

In effect, the efficient management of changes in requirement, design, and code documents appears to be one of the main problems of software development. Questions that arise are:

- How can one handle the effects of changes to software artifacts, e.g. to update its internal state to reflect the current state of the software development process.
- How can we constrain the effect one change has to other parts of the software system.
- How can we ensure proper flow of change information, e.g. to notify people involved in the process of the changes that concern them.

All of these aspects have to be taken into account in the software process management activities. Such global distribution of software projects drastically increases the need for coordination and communication support. Additionally, it adds another dimension to the problem of coordinating such a project: Not only does the project's complexity make it hard for people to figure out who to contact when questions come up. In a distributed project they cannot just walk over to the person in question. Even picking up the phone and calling the other person might not be feasible if the project is distributed over different time zones. Discussions in the style that people are used to, i.e. official group meetings, or informal discussions over some idea on a piece of paper, become impossible. Instead, the necessity arises to support asynchronous as well as synchronous collaboration with computer tools.

Over the last few years, many systems, techniques and tools have been developed which attempt to address these issues. Computer Supported Collaborative Work (CSCW) systems have been used to aid distributed software development [20]. Groupware tools provide support for collaborative work by providing shared workspaces that can be accessed from geographically distributed locations; workflow management approaches support the coordination of software engineering processes, change impact analysis and change management techniques can be used to send notifications to an appropriate set of users in order to ensure the timely conclusion of a project with high-quality products, and so on. However, none of these technologies by themselves are enough to solve the problems posed by distributed software engineering [21]. In order to provide useful support for that complicated process, methods and tools must integrate coordination, collaboration and communication support, i.e. it must integrate, to a degree, all of the above technologies, as well as provide support for project monitoring, and configuration management. Since

VSEs are becoming more common in software development projects, there is an urgent need in these domains to improve project coordination by using automated computer support systems.

3.2 Distributed Software Configuration Management

Configuration management is a key contributor to successful software development [22]. The configuration management (CM) is an important function on any software project since it is a communicator that establishes relationships with all the project activities. CM communicates by providing technical information about each specification, document, manual, code, test results, and so forth, provided to the project team. Basically, Software Configuration Management ensures that changes to the software are managed. While Configuration Management as such is necessary as an underpinning of a continuous process improvement programme, distributed Configuration Management is essential for a VSEs.

In a single-site development environment, it is considered good practice to apply established SCM tools and techniques. A problem arise is the fact that multi-site projects are much more complex that give rise to unique challenges:

- *How to manage different changes at different sites as one product*. It can be difficult to control the changes within each location and coordinate each location's change process with total product development across multiple sites. The changes at each site must be coordinated according to the schedule for that site, but they must also be viewed in the context of their effect on the total project schedule.
- *How to apply standards consistently*. Maintaining the consistency of standards application – such as enforcing when to release software, what documentation format to use, or how to approve changes – is more difficult in a virtual organisation. Different sites tend to interpret the standards according to their culture and local interests. Written expectations should make it clear how standards should be applied so that everyone is working from the same interpretation. Wrong assumptions can lead to a misapplication of the standard, which can show up as problems during integration and test. This in turn can delay releasing the product, which hurts all parties. The consistent application of standards assures the customer that the software has gone through an acceptable process, no matter where it was developed.
- *How to incorporate changes in a timely manner*. Challenges here include how to coordinate changes into different software builds, make different versions work together, and ensure that everything follows the process in an acceptable timeframe. With changes going through a local software configuration control board, and then through additional reviews at the location where the software is integrated. This is much more difficult than in a nonvirtual organisation, where changes are managed in one location.

In order to address these issues, a stronger configuration management process and change control management has to be proposed. All these may be addressed through a virtual software configuration management board that reports to a centralised software configuration control manager. The virtual SCM board must represent all affected and interested functional and organizational parties; to be a single body accepts responsibility for review, approve, or reject proposed changes to the software; and ensure that the proper process and documentation exists. Consequently, a centralized configuration library needs to be set up to help in configuration control; a version control system is need to help track changes and a centralized version control system should automatically handle changes from each remote site.

It is believed that a strong SCM process will help overcome differences in management styles and cultural interpretations by enforcing a common discipline and set of expectations for documentation and change handling. Documentation control is another important SCM task. In a distributed software project, documentation becomes the most used means of communication, like glue its holds the project together. An SCM plan that documents the expectations of all team members is an extremely useful tool. As first step, it is important to lay a solid foundation for global development and unite everyone's expectations about lines of responsibility, schedules, cost, and other business and technical concerns. To do this, a legal document, a statement of work identifies what must be performed or delivered and over what timeframe, should be as unambiguous as possible. Documenting the virtual organisation structure also gives management, each project team member, and other project stakeholders a sense of organisation and an understanding of individual roles and responsibilities. Documentation requirements specifications and plans such as quality assurance identify expectations for all team members before issues come up and cause confusion. The reference to maintenance, and ownership of these documents and the benefits they provide are critical in large teams with geographically distributed members.

Also, in distributed projects it is vitally important to make sure the requirements are thoroughly understood by everybody involved, and to arrive at high quality definition of the interfaces between components that are to be developed at different sites. This means that more time and effort needs to be spent in the early phases of system requirements, system design, and interface definition, rather than implementing first and finding problems only during system integration and testing. In other words, the focus of the project has to be moved from implementation to earlier phases of the software engineering process, thereby putting into action an important software engineering paradigm.

Clearly, stronger configuration and change management are needed to achieve process adaptability. A central feature is the need to automatically inform the appropriate users about a change and its consequences for their work. In doing this, Software Configuration Management (SCM) is key in holding a virtual project together. Since a SCM system tracks both changes and their impact on the whole system, its implementation involves tools and methods unique to controlling software process. It is thus necessary to create a virtual configuration management system to support the VSEs, and the approach to such a system must exploit the latest technologies for distributed systems and distributed problem solving.

3.3 Effective Communication

A virtual software team or virtual software organisation must be able first to communicate in a distributed fashion and second to communicate effectively along the spectrum from real time to non-real time. When managing software development projects at distributed locations, communications methods and tools offer one of the most powerful and effective ways to gather and disseminate information and control the project. In effective communication, people not only send and receive verbal or text messages, but nonverbal indicators make it easier for the receiver to interpret the message's intent and for the sender to ascertain that the message was received and understood. Figure 3 shows the most popular communication techniques for supporting virtual software development with classification from real-time, to non-real time delivery.

Real time	Near-real time	Non-real time
▪ Telephone ▪ Video Conference ▪ Teleconference ▪ Networking	▪ E-mail ▪ Voice-mail ▪ Pagers ▪ Fax	▪ Mail ▪ Groupware Software ▪ Bulletin Boards ▪ Database

Figure 3: Classification of communication according to real - time, near-real time, and non-real time delivery

All the communication techniques today have increased our ability to communicate meaningfully. Cellular telephones, portable telephones, pagers, telephone conferencing have allowed access to instant information, thus increasing information frequency. Videoconferencing and teleconferencing have enabled groups of individuals communicate instantly in meetings without being in the same place. Access to stored information through voice mail, e-mail, electronic networking such as Internet or Intranet, bulletin boards and databases that contain common project information, such as schedules, problems report, and the shared software and tools in Groupware packages has increased information bandwidth. Some or all of these technologies must be in place for a virtual software development project to be successful.

However, there is no substitute for regular face-to-face meetings. Whatever advance in communication technology is, the face-to-face meeting is always the most effective method for human interaction at real-time. Since only in this type of communication the receiver can most effectively understands both the message and its intent, including nonverbal messages such as body language, gestures, eye movements, intonation, and facial expression. During a conversation, messages are formed, communicated, interpreted, responded to, and clarified. With this rich interaction, communicators can grasp the message with little misunderstanding.

In a VSE, face-to-face communication is unfortunately far less frequent than in a nonvirtual environment because it is costly. How often depends on the company, the project's budget, and the project management structure. It should be part of the project's kickoff, if at all possible, and used in meetings where it is important that a message must be clearly sent and understood.

The next best communication is likely to be video conference which promises the same type of interface quality as face-to-face meetings without the cost. The only difference in between is the quality of the communication. The inherent transmission delays between speech and video make video conference communications that involve movement less effective than a face-to-face meeting. As transmission delays continue to diminish, this artificial environment will hopefully improve. The availability of this technology is also an issue. It is still expensive to purchase transmission bandwidth and the proper equipment. As the information superhighway infrastructure

continues to build worldwide, this type of technology should become more widely available via the Internet, and personal computer vendors will intensify efforts to add video conference peripherals to PCs. Until these problems are addressed on a large scale, however, face-to-face and telephone communications will continue to be more effective media for the virtual development team.

3.4 Development support through Intranet

So far, we have been discussing specialised process support systems that need to be used for distributed software development. Practically, distributed software development would be much harder, maybe even impossible, without the Internet's capability of quickly transferring large amounts of data from one development site to another**.** The Internet, more specifically, Intranet, is thus the most important enabling factor for distributed software development.

In fact, the Intranet also facilitates the asynchronous exchange of messages, allows the deployment of documentation and even software, on central servers for everybody to access, thereby avoiding the necessity to hold redundant data at each development site. In addition to data transfer and communication services, distributed configuration management and other data management can be provided via the Intranet. Also, project management and coordination services can be implemented using the Internet [23] with fairly secure since Intranets usually employ firewalls to keep out intruders from the outside world. Building on artifact handling services the Internet already provides, dependencies between those artifacts, and between the activities that create the artifacts, will be identified and managed. Intranet will create easier channel to deliver computer-based training just in time, capture and share their knowledge, learn and innovate.

Though Groupware has become very popular and has proven its worth in companies worldwide, the problem with Groupware is that it's been very expensive and difficult to implement. With the introduction of cheap and easy-to-construct intranets [24], that's no longer the case. Firms can now get the benefits of groupware on their intranet.

The uses for intranets are almost unlimited. Intranets are just as easy to build as they are to use, so this may be just the perfect solution to support distributed software development and share information throughout intranets allow the employees to do a better job. These things give organisations a competitive edge. Though the Internet today is restricted to visual means only, restricting human communication to a serious extent, this problem might be solved with growing bandwidth, allowing for audio and video conferences. More introduction of Intranet can be found in [24].

4. Extending BOOTSTRAP for VSE of VSO

VSEs and VSO have evolved largely because of technology improvements over the last decade. These include improved communications structure, increased communication bandwidth, decreased communication cost, better price-performance ratio for PCs and controllers, and better software. Technology has made it easier and more cost-effective to manage organisations that are geographically distributed.

The success of software developing organisations heavily depends on the quality of their software products which is intrinsically related to the quality of the software development processes. Since VSEs and VSO are becoming more common in software industry, process assessment and improvement models should not only be definable for single-site development, but also for a distributed project. In contrast to the single-site, there are significant differences and problems in the distributed development environment, as discussed above, non of these issues are included in the BOOTSTRAP methodology. Therefore a number of extensions should be made in the BOOTSTRAP methodology to achieve process management success for the VSE and VSO.

In addition to the issues discussed in [6], the technology dimension in BOOTSTRAP are concerned mostly affected, and the processes in this dimension has to be extended. At a high level, the extensions are:

- Technology support system management needs to be established at organisational and project level.
- The documentation management process may need to be added into project management category.
- Adopt documentation tools to develop and maintain the project documents.
- Establish communication systems.
- An Intranet system should be set up to support information sharing, and project management.
- Adopt automated configuration management tools.
- Adopt automated version control management tool.

- Establish and maintain a problem report repository to enable continually process improvement.

5. Conclusion

The formation of Virtual Software Environments and Virtual Software Organisation is an inevitable trend caused by the changes in the social structure induced by the global network and communication technology. The development of software products has become a highly cooperative and distributed activity involving working groups at geographically distributed locations [25]. These groups show an increasing mobility and a very flexible organisational structure. Process methodology and technology have to take such evolutions into account.

This project is addressing practically the needs of virtual software organisation and its distributed development environments. This is the first attempt to apply software process assessment and improvement models into the virtual software environment. In addition to the need for a well defined software process and disciplined process management and improvement approach, distributed software development strongly requires facilities to support coordination, and communication among software developers.

This paper presents the initial research findings of an evaluation of the specific problems and requirements for software process improvement in VSEs. The methodology of the BOOTSTRAP approach has been analysed to provide the background knowledge, and the suggested extension has been indicated concerning its application into distributed software environment.

6. Reference

[1] Dale Walter Karolak, *Global Software Development*, IEEE Computer Society, 1998.

[2] *Status Report on European Telework 1999 – New Methods of Work*, Annual Report from the European Commission. < http://www.eto.org.uk/twork/tw99/index.htm>, 1999.

[3] J. Lipnack and J. Stamps, *Virtual Teams: Reaching Across Space, Time and Organisation With Technology*, Wiley & Sons, New York, 1997.

[4] Katrina D. Talbot, The virtual company, http://forum.iee.org.uk/forum/library/items/1999_05/talbot/frame2.htm

[5] R. Posch, *Maintaining Public Trust in the Virtual Organisation World*, Direct Marketing, Vol.57, 76-79, 1994.

[6] H. Guo, G. King, M. Ross, G. Staples, *Extending ISO/IEC (TR) 15504 for Virtual Software Environments*, accepted by the 1st International SPICE Conference, June, 2000, Limerick, Ireland

[7] Humphrey, W. S., *Managing the Software Process*, Addison Wesley, Reading, MA, 1989.

[8] Humphrey, W.S., *A discipline for Software Engineering*, Addison-Wesley, 1995, pp. 97-106.

[9] M.M.Lehman, *Software's Future: Managing Evolution*, IEEE software, Vol.15, No. 1, 1998, pp.40-44.

[10] A. M. Christie, *Software Process Automation - The Technology and Its Adoption*, Springer-Verlag, 1995

[11] H. Guo, M. Ross, G. King, G. Staples, *A Study of Software Quality Assurance and Teleworking in The Software Industry*, Proceeding of INSPIRE III (Training and Teaching for the understanding of Software Quality), pp.251-261, 1998, London.

[12] H. Guo, M. Ross, G. King, G. Staples, *An investigation of Software Quality Implementation in Teleworking Environments*, Proceeding of INSPIRE IV (Training and Teaching for the understanding of Software Quality), pp.176-192, 1999, Crete

[13] H. Guo, G. King, M. Ross, G. Staples, *SPICE: A Suitable Model for managing the Software Process in Virtual Software Environments ?* Proceeding of CONQUEST'99, pp.44 – 52, 1999, Germany.

[14] H. Guo, M. Ross, G. King, G. Staples, *BOOTSTRAP 3.0 – An European Software Process Assessment Methodology based on CMM and SPICE*, accepted by the international conference of World Computer Congress 2000, Beijing, China.

[15] Kuvaja, P., Simila, J., Krzanik, L., Bicego, A., Koch, G., and Saukkonen, S., *Software Process Assessement and Improvement - The BOOTSTRAP Approach*, Blackwell Business, Oxford, UK, and Cambridge, MA 1994

[16] A Bicego, M Khurana, P Kuvaja, *Bootstrap 3.0 - Software process assessment methodology*, Proceedings of Software Quality Management (SQM 98), 1998, pp. 26 -37.

[17] Blan. P. and Scott. W. R., *Formal Organizations*, Scott, Foresman, San Francisco, 1962.

[18] Van de Ven, A. H., Delbecq, A. L., and Koenig, R. Jr., *Determinants of coordination modes within orgnnisations.* Amer. Soc. Rev. 41 (1976), 322-338.

[19] Roberl E. Kraut and Lynn A. Streeter, *Coordination in Software Development*, Communications of the ACM, Vol. 38, 3, pp.69-81, 1995.

[20] John McCarthy, *The state-of-the-art of CSCW: CSCW systems, cooperative work and organisation*, Journal of Information Technology (1994) 9, pp. 73-83

[21] Harald Holz, Sigrid Goldmann, Frank Maurer, *Working Group Report on Coordinating Distributed Software Development Projects*, Proceedings of the IEEE 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98), 1998, pp.69-72, Stanford University Californian, USA

[22] H. Ronald Berlack, *Software Configuration Management*, John Wiley & Sons, Inc. 1992

[23] Kigrid Goldmann, Boris Kotting, *Working Group Report of the 2nd Workshop on Coordinating Distributed Software Development Projects*, Proceedings of the IEEE 8th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '99), 1999, Stanford University Californian, USA

[24] Mellanie Hells, *Intranet as Groupware*, Wiley & Sons, New York, 1997. ISBN 0-471-16373-2.

[25] Norbert Glaser, Jean-Claude Derniame, *Software Agents: Process Models and User Profiles in Distributed Software Development*, Proceedings of the IEEE 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98), 1998, pp. 45-56, Stanford University Californian, USA

QW2000 QuickStart 9Q

Mr. Otto Vinter
(DELTA Danish Electronics, Light & Acoustics)

Experience-Based Approaches to Process Improvement

BACK TO QW2000 PROGRAM

Key Points

Problem diagnosis

- Is a simple and effective way to find problems in software development process
- Is a good starting point for process improvement programmes in companies
- Step-wise improvements with quick wins
- Changes assessment recommandations

About the Author

Otto Vinter is a project manager specializing in software process improvements with the Danish software engineering consultancy company DELTA. Until recently, he was responsible for software process improvements at Br'el & Kj†r.

He has been active in defining software engineering standards, procedures, and methods to be employed at Br'el & Kj†r. He has been the driving force in the company's improvement activities in testing, requirements engineering, development models, and introduction of object-oriented development methods.

He has managed software development projects for 30 years; with Br'el & Kj†r from 1986, before that with the Danish branch of Control Data Corporation, and with Regnecentralen.

He received his Masters Degree in Computer Science from the Danish Technical University in 1968. He is an associate teacher for BSc. level education in Computer Science, is an active participant in Danish software knowledge exchange groups, is on the programme committee of several international conferences, performs mentoring activities for clients, and is an expert evaluator on the framework programmes of the CEC.

BACK TO QW2000 PROGRAM

Quality Week 2000

Experience-Based Approaches to Process Improvement

-

Otto Vinter
Project Manager Software Process Improvements,
DELTA Software Engineering,
Tel: +45 4586 7722, Fax: +45 4586 5898
otv@delta.dk
&
Software Engineering Mentor
Tel/Fax: +45 4399 2662, Mobile: +45 4045 0771
vinter@inet.uni2.dk http://inet.uni2.dk/~vinter

Normative Models for SPI

CMM
BOOTSTRAP
SPICE (ISO15504)

Defined comprehensive process
Levels of maturity (capability)
Key process areas (KPAs) for each level
Assessed by certified assessors

Fundamentals in Normative Models

Level	Characteristics	Principal Characteristics	Result
5 OPTIMISING	feedback — the process includes feeback	Continuous evolution & continuous process improvement	Productivity & Quality
4 MANAGED	quantitative — the process is measured	Technological replacement Problem prevention	
3 DEFINED	qualitative — the process is institutionalised	Process measurement Process analysis Quantitative quality plans	
2 REPEATABLE	intuitive — the process is person-dependent but it is managed	Training Systematic review & testing Standards	
1 INITIAL	ad hoc — chaotic	Project management Planning Configuration control Quality assurance	

Software Process Assessment & Improvement (The BOOTSTRAP Approach) ISBN 0-631-19663-3

Problems with Normative Models

Abstract model
Assessment by external body
Costs money
Organisational focus
Points out KPAs to be improved
Little help on precisely what to do
Raises a lot of expectations

Alternative Approaches to SPI

Experience-based improvement actions

- Analyse problems from previous projects to extract knowledge on frequently occurring problems
- Change the development process through the use of an optimum set of methods and tools available to prevent these problems from reappearing
- Measure the impact of the changes in a real-life development project
- Diffuse the results to the rest of the organisation

Characteristics of Alternative Approaches

Experience-based
No specific model
Hot-spot driven
Focus on prevention
One issue at a time (incremental)
Piloting at project level
Evolve rather than define (feed-back)
Fits CMM level 1-2 cultures
(where most of us are)

Examples of Alternative Approaches

Analysis of
- defects
- progress reports, etc.

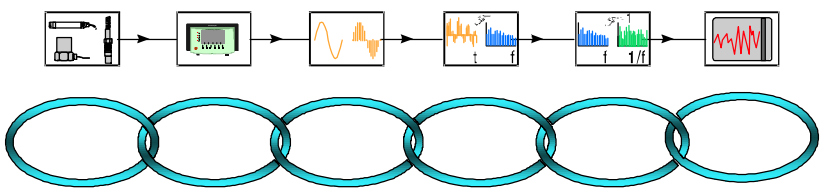
Structured/Selective interviews
- project managers
- project members
- customers, etc.

Goal-Question-Metric paradigm (GQM)

Some frameworks for alternative approaches:
- Experience Factory (V. Basili e.a.)
- Product Process Dependency Models (PROFES: www.ele.vtt.fi/profes)
- but primarily: you must find your own way

Brüel & Kjær - Sound & Vibration Measurement

Transducer | Signal conditioning | Digital conversion | Signal processing | Post processing | Output



ISO 9001 CERTIFIED

Defect Analysis from Error Logs

Definitions

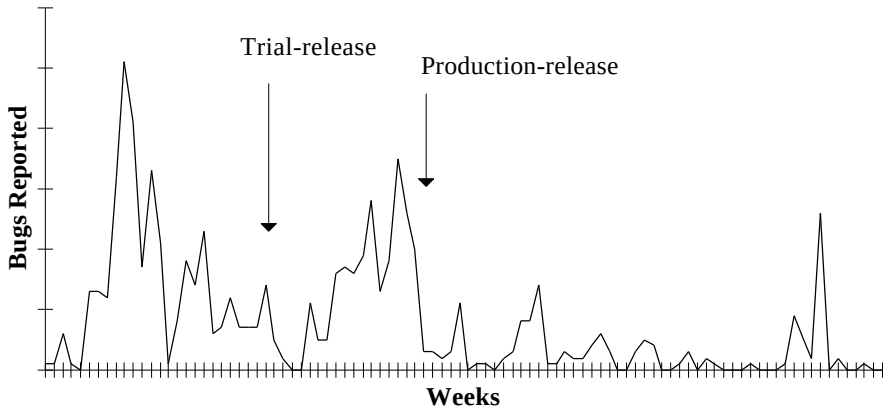
- Bugs are anything between serious defects and suggestions for improvements
- Problem reporting starts in the integration phase

Error Logs Analysed

- Embedded and PC Windows development projects
- Project sizes app. 5-7 person years
- In total app. 1000 bugs analyzed in detail
- Problem reports covered a period until 18 months after first release

Problem Report Distribution over Time

Example of an Error-prone Product



Defect Analysis Technique

Interview sessions

- 1-2 developers and 1-2 process consultants
- app. 5 minutes / bug

Bug Categorisation

- based on a bug taxonomy by Boris Beizer:
  - Boris Beizer: Software Testing Techniques, Van Nostrand Reinhold, 1990
- comprehensive set of bug categories and statistics

Capture Subjective Information on the Bugs

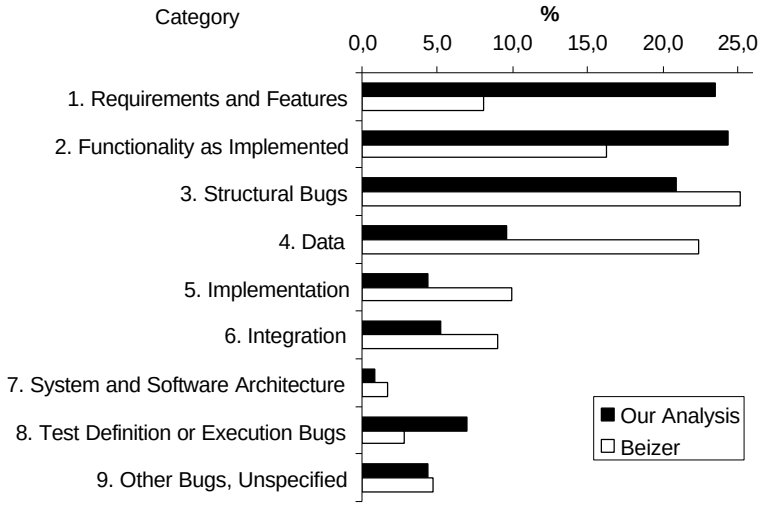
- where and how the bug occurred
- quality factor (reliability, usability, functionality ...)
- complexity of bug (correction cost)
- what could prevent the bug

The Beizer Bug Taxonomy

1. Requirements and Features
2. Functionality as Implemented
3. Structural Bugs
4. Data
5. Implementation (standards violation, and documentation)
6. Integration
7. System and Software Architecture
8. Test Definition or Execution Bugs
9. Other Bugs, Unspecified

Each category detailed to a depth of up to 4 levels

Problem Report Categorization



1st Action: The Prevention of Errors through Experience-driven Test Efforts (PET)

Results of the analysis of error logs

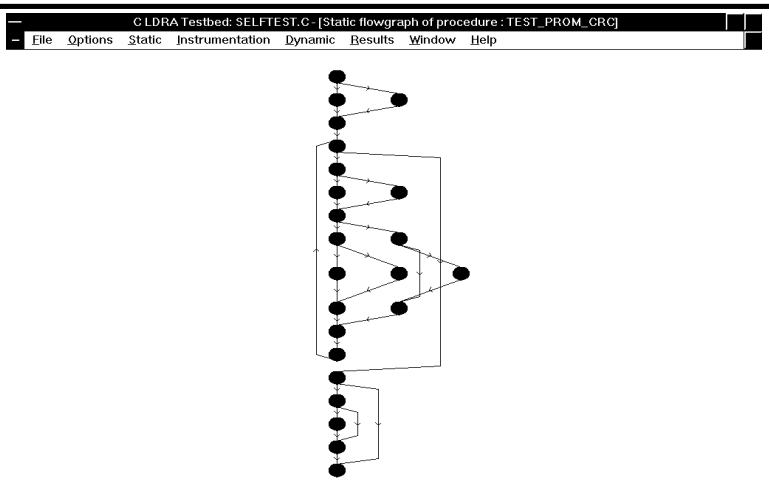
- **no special bug class dominates embedded software development**
- **requirements problems, and requirements related problems, are the prime bug cause (>36%)**
- **problems due to lack of systematic unit testing is the second largest bug cause (22%)**

Action: Improve Testing Processes

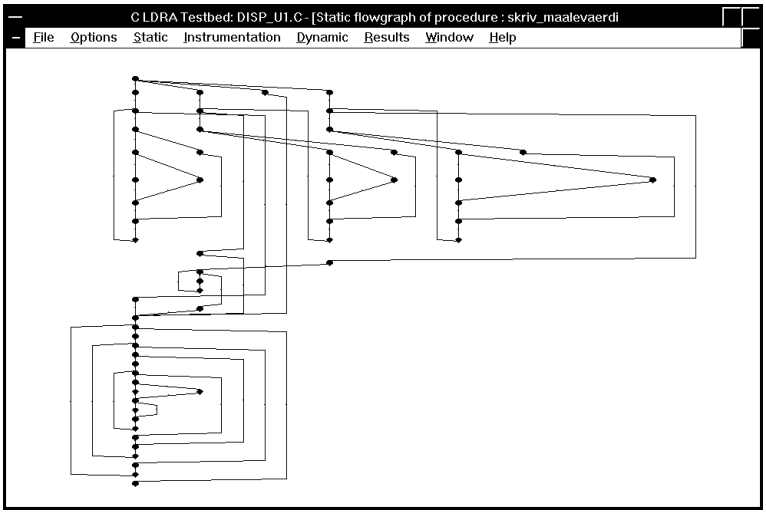
- **static analysis**
  - **source code complexity, data flow anomalies, coding standards**
- **dynamic analysis**
  - **code coverage by test cases, cross references code to test cases**

Funded by CEC. Final Report: http://www.esi.es/ESSI/Reports/All/10438

Static Flowgraph (McCabe = 10)



Static Flowgraph (McCabe = 20)



Test Coverage Strategy

Test coverage must take into consideration

- perceived complexity
  - based on a visual inspection
- criticality
  - product and project issues
- developer experience
  - domain, product, and unit
- bug and change request history
- degree and type of reuse

Test cases for dynamic analysis are selected by normal testing techniques

- equivalence partitioning, boundary value analysis etc.

Results of the Improved Testing Process

46% Improvement in Testing Efficiency

- Removing static bugs
- Increasing unit branch coverage to > 85%

75% Reduction in Production-Release Bugs

- Compared to Trial-Release

70% Requirements Bugs in Production-Release

Next Action: Improve Requirements Engineering

2nd Action: A Methodology for Preventing Requirements Issues from Becoming Defects (PRIDE)

Results of the analysis of error logs

- Requirements related bugs 51%
- Usability issues dominate 64%
- External software (3rd party & MS products) 28%

Action: Introduce requirements techniques

- Use Situations (Scenarios)
  - Relate demands to use situations. Describe tasks for each scenario.
- Usability Test, Daily Tasks, Navigational Prototype
  - Check that the users are able to use the system for daily tasks, based on a navigational prototype of the user interface.

Funded by CEC. Final Report: http://www. esi.es/ESSI/Reports/All/21167

Scenarios

"The defining property of a scenario is that it projects a concrete description of activity that the user engages in when performing a specific task, a description sufficiently detailed so that design implications can be inferred and reasoned about"

John M. Carroll,

Scenario-Based Design:
Envisioning Work and Technology in System Development,
Wiley 1995.

Many names for the same concept:

- scenarios
- use situations
- work situations
- work setting

Road Test Scenario

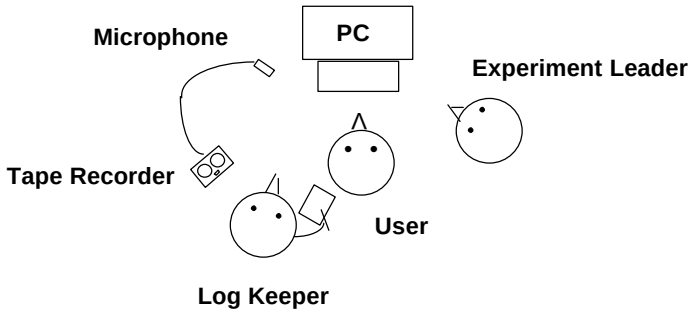
Road tests are done in the car when it is driving on special test roads. The purpose of the recordings is to identify noise sources, comparing them to earlier measurements, and eventually removing the noise through changes to the car design.

The engineer will record noises from various parts of the car when it is driving at various speeds, when it is turning, when it is breaking, etc. The microphone will have to be mounted at various places not accessible from the drivers seat.

Usually, the engineer has a plan for what to measure, but circumstances may change so that he has to do something different and later find out what he actually did and which sounds relate to what.

Back at the lab the sounds will be analyzed by the engineer himself, or - in many cases - someone else.

Usability Test Environment



Results of the 2nd Improvement Action

Product is selling steadily more than twice as many copies

- compared to a similar product developed for a similar domain by the same team

Usability is exceptional in the market

- Users'interaction with the product was totally changed as a result of the early usability tests

Almost 3 times increase in productivity for the development of the user interface

27% reduction in problem reports

Improvement Approach Based on Project Manager Perceived Problems

Interview project managers

- 1 project manager and 3 process consultants
- app. 1 hour per interview
- process-oriented interview guide

Analyze problems raised

- classify the problems according to software process
- present the issues at a project manager workshop
- establish consensus on a few major issues
- let each manager select an issue to improve

Monitor development projects

- introduce and train the team on the selected issue
- coach, and support regularly and on request
- collect experience and results

Improvement Approach Based on Project Manager Perceived Problems

Major Issues

- iterative software development model
- requirements
- project monitoring (estimation, progress evaluation, etc.)
- project conclusion (test, configuration management, etc.)

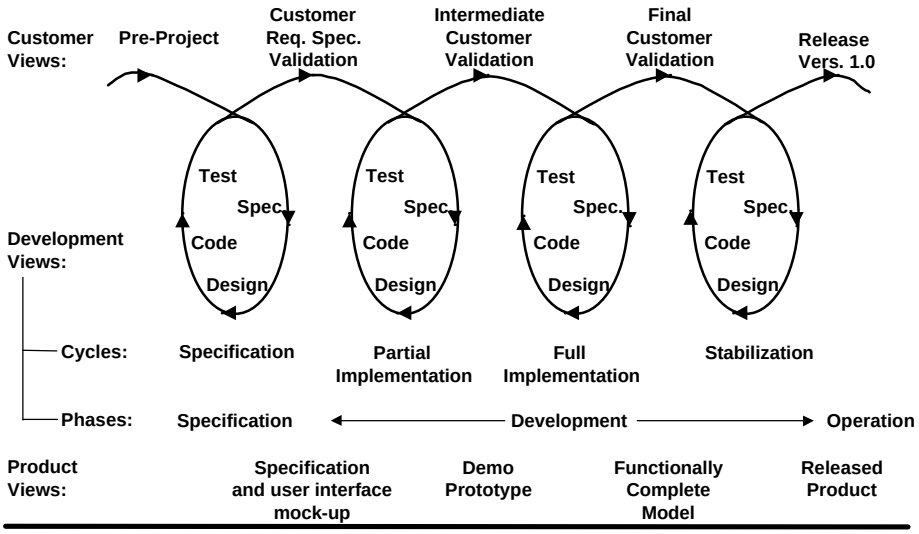
Improvement Actions:

- diffuse and adopt the test and requirements techniques
- define new iterative development models
- improve project monitoring

Results:

- improvement actions were performed with enthusiasm
- project related actions completed successfully

3rd Action: Iterative Software Development Model



Iterative Software Development Model

Types of Iteration

- user validation (usability test)
- exploration (e.g. performance)
- proof-of-concept (e.g. architecture)
- well-defined functionality (increment)

Duration

- time-boxed
  - less than 6 weeks, with 2 weeks spacing
- full increments
  - app. 3 months, with 2 weeks of stabilization

Essential for Success

- user validation
- stabilization

Results of the 3rd Improvement Action

- drive and motivation of team increased dramatically
- turbulence within and around team decreased
- control over project progress increased
- requirements creep was kept under control
- resources used more efficiently
- serious problems were uncovered early
- quality improved step-by-step

Comparison of Recommendations

Recommendation	Defect Analysis	1st Bootstrap Assessment	Project Mgr. Interviews	2nd Bootstrap Assessment
Development Model				
- iterations	x	x	x	Ö
- risk management			x	
Requirements	x	x	x	Ö
Project Monitoring				
- estimation		x	x	x
- time & resource usage		x		x
- monitor progress		x	x	x
Project Conclusion				
- configuration mgmt.		x	x	x
- testing	x	x	x	(Ö)
- release criteria	x		x	
Reuse			x	
Process Descriptions		x		Ö

Problem Diagnosis Results

Defect analysis from error logs

- has established a basic process for testing and release
- has improved our requirements process and products

Improvements from interviews

- successful diffusion and adoption of previous improvement actions
- established a new software development model

Impacts on maturity

- development model, test, and requirement issues are no longer on the Bootstrap recommendation list

In Conclusion

Problem diagnosis approach

- is a simple and effective way to find problems in the software development process
- good starting point for process improvement programmes in companies
- step-wise improvements with quick wins
- changes assessment recommendations

However, normative models are needed

- comprehensive framework
  - KPAs are important, levels are not
- established assessments
  - effect of improvements

Experience-Based Approaches to Process Improvement

Otto Vinter[1]
DELTA Software Engineering, DK-2970 Hørsholm, Denmark
Email: otv@delta.dk

Abstract

Software process improvement (SPI) is usually based on well-known models of software process maturity such as the Software Engineering Institute's Capability Maturity Model (CMM) and the European counterpart Bootstrap. This paper reports on alternative approaches to SPI based on knowledge and experience that is already available in the organisation. Rather than a formal comprehensive assessment of the all software development processes in the organisation a "problem diagnosis" is performed. The problem diagnosis approach to SPI aims at identifying the most important (process) issues as perceived by the organisation. Improvement actions are then planned and implemented in close collaboration with the powerful actors in the organisation. The paper focuses on the results of using such problem diagnosis techniques at Brüel & Kjær as an alternative SPI strategy. The paper will report on problems and successes, and relate these results to formal assessments performed in parallel by an external body.

1 Introduction

Software process improvement (SPI) is usually based on well-known models of software process maturity such as the Software Engineering Institute's Capability Maturity Model (CMM) [6] and the European counterpart Bootstrap [2]. The assessment of current practices through the use of such normative models is generally considered as the proper way of identifying and prioritising improvement initiatives.

However, it is also claimed [3] that these models represent a too rigid and limited view of the software development processes and that they do not consider the variety and complexities of software producing organisations.

[1] The work reported in this paper was performed while the author was employed as manager for software process improvement (SPI) projects at Brüel & Kjær Sound & Vibration Measurement A/S, DK-2850 Nærum, Denmark. For more information on the process improvement activities at Brüel & Kjær visit: http://inet.uni2.dk/~vinter/engindex.htm and look for projects.

Furthermore, many organisations have found it very difficult to translate the assessment results into concrete improvement actions. And almost all process improvement persons or groups in organisations at the lower levels of maturity (1-2) have encountered severe resistance to their improvement initiatives from within the organisation. Alternative or complementary approaches to assessments should therefore be considered.

This paper reports on alternative approaches to SPI based on knowledge and experience that is already available in the organisation. Rather than a formal comprehensive assessment of the all software development processes in the organisation a "problem diagnosis" is performed. The problem diagnosis approach to SPI aims at identifying the most important (process) issues as perceived by the organisation. Improvement actions are then planned and implemented in close collaboration with the powerful actors in the organisation.

The paper focuses on the results of using such problem diagnosis techniques at Brüel & Kjær as an alternative SPI strategy. The paper will report on problems and successes, and relate these results to formal assessments performed in parallel by an external body.

Brüel & Kjær is a leading manufacturer of high-precision measurement instruments for sound and vibration measurement applications. The company is headquartered in Denmark, but the majority of the products are sold through subsidiaries around the world. In the past most of the products were based on embedded real-time software, but now PC-Windows applications prevail.

The next section (section 2) describes one problem diagnosis approach based on the analysis of defects (error reports) from earlier projects. Two improvement actions were initiated and completed successfully as a result of this approach. In section 3 is reported the findings of the first Bootstrap assessment, which was performed during the implementation of the above-mentioned improvement actions.

Section 4 describes another problem diagnosis approach based on interviews of the leading project managers in the organisation – the powerful actors of a level 2 organisation. Based on the findings from these interviews a number of improvement actions were identified and implemented by the project managers. Improvement actions that directly related to development projects succeeded; those of a more general nature (e.g. organisation-wide) did not.

Section 5 describes the findings of the second Bootstrap assessment, which was performed after the above improvement actions. The findings and recommendations clearly indicate that the improvement actions performed as a result of the problem diagnosis approaches actually helped our organisation to improve on the key practices of a normative model.

Section 6 compares the findings of the different approaches. Section 7 draws the conclusion, that problem diagnosis approaches to SPI seem to be an effective improvement strategy for an organisation at the lower maturity levels (1-2), and a valid alternative to formal assessments according to normative models like CMM and Bootstrap, when an organisation wants to initiate a SPI programme.

2 Defect Analysis

When we first considered improving our processes at Brüel & Kjær, our management did not want to invest in a comprehensive improvement programme. The company had just been through a major reorganisation and downsizing. So rather than starting our process improvement programme with an assessment according to one of the common maturity models, the improvement strategy had to be an experience-driven, incremental process based on the available information in the organisation, and focusing first on the major process issues.

We therefore started by performing root cause analyses of error logs generated during previous development projects. Based on the findings of the analyses we would then introduce improvements in our development process to prevent frequently occurring types of errors. We conducted thorough analyses of bugs reported during development and after release of products. In these analyses we classified bugs according to a taxonomy described by Boris Beizer[1].

The analyses [7][8] showed the need to perform a more systematic unit test of our products. The first improvement action therefore focused on improving our testing process. However, the analyses also showed that the major cause of bugs stemmed from requirements related issues. So when the test improvement action had completed with success, a second improvement action was then undertaken to improve our requirements engineering process.

These improvement actions have been funded by the Commission of the European Communities (CEC) under the ESSI programme: European System and Software Initiative. The title of the test improvement project is: PET - The Prevention of Errors through Experience-Driven Test Efforts (ESSI project no. 10438) [7][8]. The title of the requirements engineering project is: PRIDE - A Methodology for Preventing Requirements Issues from Becoming Defects (ESSI project no. 21167) [9].

2.1 The Test Improvement Project

The software quality of our company was felt to be unsatisfactory. Too many products were shipped with bugs. It was the general opinion that this was caused by a lack of testing by the developers before release.

It was therefore decided to conduct a process improvement experiment to find ways to improve the testing process. The project was titled: The Prevention of Errors through Experience-Driven Test Efforts (PET) [7][8].

The problem reports were analysed and bugs in them categorised using Boris Beizer's taxonomy [1]. We found that bugs in embedded real-time software follow the same pattern as other types of software. We found that the major cause of bugs reported (>36%) are directly related to requirements, or can be derived from problems with requirements. The second largest cause of bugs (22%) stems from lack of systematic unit testing.

The techniques selected to improve unit testing were: Static and dynamic analysis. Tools were installed to support these techniques. We experienced a 46% improvement in testing efficiency (bugs found per person hour) and we raised the branch coverage of all units to above 85%.

An improved (production) version of the baseline product was then released and tracked for the same number of weeks we had measured on the existing (trial) version after its release, so that we were able to evaluate the effect of the experiment on problem reports.

The team received 75% fewer error reports than for the trial-release version of the product. Of those error reports 70% were found to be related to requirements e.g. to bugs that could not have been found through static and dynamic analysis. This once more confirmed the need for us to improve the requirements process.

In spite of these remarkable results the use of static and dynamic analysis never spread throughout the organisation. Some project managers ignored the results. Others started to work with the techniques, but stopped when time pressure built up. Those who continued, released products with remarkably fewer bugs.

2.2 The Requirements Engineering Improvement Project

In the second improvement action we performed a closer analysis of requirements related bugs in order to find and introduce effective prevention techniques in our requirements engineering process. The project was titled: A Methodology for Preventing Requirement Issues from Becoming Defects (PRIDE) [9].

From the analysis of requirements related bugs we found that requirements issues are not what is expected from the literature. Usability issues dominate (64%). Problems with understanding and co-operating with 3rd party software packages and circumventing their errors are also very frequent (28%). Functionality issues that we (and others) originally thought were the major requirements problems only represent a smaller part (22%). Other issues account for 13%. The sum of these figures adds up to more than 100% because one bug may involve more than one issue.

This result had an impact on our methodology. We focused on usability problems, and early verification and validation techniques, rather than correctness, and completeness of requirements documents.

We therefore introduced the following techniques on a real-life project:

- *Scenarios*
  Relate demands to use situations. Write down short descriptions for each known use situation. Explain the work environment, purpose of the work, and the people working. Describe the essential tasks in each scenario. Tasks should have a goal, a beginning, and a termination.

- *Navigational Prototype Usability Test, Daily Tasks*
  Check that the users are able to use the system for daily tasks based on a navigational prototype of the user interface. Navigational means: screens have static content; menu points and buttons automatically change to another screen. The technique tests the prototype with users simulating daily tasks, revises the design, tests it again, and so on until the result is acceptable.

The quantitative results of using these techniques on the project were an overall reduction in error reports of 27%, a 72% reduction in usability issues per new screen, and a 3 times increase in productivity in the design and development of the user interface.

What was also surprising was that not only did we experience a reduction in bugs related to requirements issues, we also found a reduction in other bug categories. The derived effect on other types of bugs than the requirements related can be explained by the fact that most of the developers achieved a deep understanding of the domain in which the product was going to be used from describing use situations (scenarios) and taking part in the usability tests. This invariably leads to a reduced uncertainty and indecision among the developers on what features to include and how they should be implemented and work. In the previous project the new screens were constantly subject to change all through to the end of the project.

However, the impact of these techniques on the perceived quality of the released product is even greater than the prevention of bugs. Describing use situations (scenarios) enabled the team at a very early stage in the requirements engineering process to capture the most important demands seen from a user/customer perspective. The developers therefore got a very clear vision of the product before the requirements were fixed. The subsequent usability tests on very early prototypes verified that the concepts derived from the descriptions of use situations (scenarios) still matched the users' needs and could be readily understood by them in their daily use situations.

The product has now been on the market for more than 18 months and it steadily sells more than twice the number of copies than the product we have compared it to. This is in spite of the fact that it is aimed at a much smaller market niche, and that the price of the new product is much higher.

In contrast to the test techniques, the interest among project managers to adopt the scenario and usability techniques has been much higher. This may be because developers much rather will work with requirements than with test.

3 Bootstrap Assessment

When the first results of the improvement actions based on defect analysis had materialised, the management of our company could be convinced that a more formal assessment of our software development processes should be performed in order to further the improvement programme.

A Bootstrap assessment was performed using the Danish company DELTA as assessors. Four projects and the software development management were interviewed by the external assessors. The overall result of the assessment was that we were at level 2.25.

The recommendations from the assessors pointed out weaknesses in the following areas:

- Development Model
  The shift in focus from a primarily hardware driven development to software had to be more focused. The assessors recommended to introduce a specific life-cycle for software development.

- Process Descriptions
  Introduce the formal as well as informal improvement actions of the software processes in the quality management system.

- Unit and Integration Testing
  The assessors commented that the improvements we had achieved under the test improvement experiment (PET) needed to be applied on a wider scale in the company.

- Configuration Management
  Again the assessors commented on the need for a more uniform way of performing configuration management, change control, and planning.

- Requirements
  The assessors were aware of the ongoing requirements engineering experiment (PRIDE) and commented on the need for a description of the process.

- Project Management
  Improve planning, estimating procedures, introduction of time and resource usage, closer monitoring of project progress.

The recommendations above clearly show that a Bootstrap assessment has a much wider perspective of the software development process than defect analysis. Defect analysis primarily highlights "hot-spots" in the development process. However, the testing and requirements "hot-spots" that we found through our defect analysis were also found through the assessment.

Due to the strong project manager culture of our company, which is typical of a level 2 maturity, the recommendations from the assessment were never turned into improvement actions. Top management had stated their commitment to follow up on the assessment recommendations, but in the end they left it to the project managers themselves to find and introduce improvement actions on their individual projects. And the process improvement group did neither have the resources nor the "power" to be able to introduce new activities on a wider scale.

With no "pressure" from the top, and projects running late, it is no wonder that the project managers chose to concentrate on their day-to-day problems of getting products out of the door. Consequently, no improvement actions were started as a result of the assessment. This is characteristic of an organisation of level 1-2 maturity.

4 Project Manager Involvement

At this point it was evident to us that we were effective in defining and introducing new and improved processes on individual development projects. However, the diffusion and adoption of the techniques on an organisation wide scale did not happen by itself. The failure of the Bootstrap assessment to spur new improvement initiatives among the project managers themselves made it evident to us that we had to involve these powerful organisational actors directly in order to improve on a broader scale.

For this to happen we needed to increase our resources for process improvements, and we succeeded in convincing our management to let Brüel & Kjær participate in the Center for Software Process Improvement [5], which is partly funded by the Danish government. The participation gave us access to a number of researchers and consultants and the resulting joint effort became the basis for spreading improvements on a wider scale in the organisation.

We involved the project managers directly by performing a new type of problem diagnosis. We wanted to analyse which (process) issues were perceived as important by them. The most common problems would then form the basis for our next improvement activities, and this - we hoped - would enable an easier diffusion and adoption across the projects.

The problem diagnosis technique [4] we used was to conduct a series of structured interviews with each of the leading project managers where they were asked about which type of problems in their development projects they felt were the most serious. Seven project managers were interviewed and detailed minutes were recorded.

From these minutes we could compose a list of problem areas that the project managers found to influence their projects most. The problems perceived by the project managers correlated very well, so a consensus could be reached quite easily:

- Software Development Model
  The present ISO9001 registered waterfall model was deemed unsatisfactory for efficient software development. A new model based on iterations, e.g. through experimental prototypes, was asked for. Risk management was also mentioned as an important element.

- Requirements
  The project managers were aware of the ongoing requirements engineering improvement action (PRIDE) and requested improvements in this area.

- Project Monitoring
  Better estimating procedures, follow up, and progress evaluations.

- Project Conclusion
  Configuration management, release criteria, and testing.

- Reuse
  Actually this item did not appear in the interviews, but the organisational changes that took place at the time of the evaluation centred on establishing a group responsible for reuse. So the process improvement group added reuse to the list.

The problems perceived by the project managers clearly resemble the recommendations from the Bootstrap assessment. Once again we see a much wider perspective of the software development process than defect analysis. However, the testing and requirements "hot-spots" that we found through our defect analysis were also found through this evaluation.

The project managers were presented with the findings at a workshop where top management also was present. They were each asked to select a topic from the list that they felt most natural to work with on their present (or up-coming project). The process improvement group and the researchers from the Center for Software Process Improvement then established support groups that would train, mentor, and follow these projects.

Three project managers chose to work with implementing the requirements engineering techniques from PRIDE (section 2.2). Two chose to work with new development models. One chose to work with reuse and one with project conclusion. Finally the R&D manager wanted to contribute by improving project monitoring.

These new improvement actions were performed headed by the project managers with mentoring and support from the process improvement group.

4.1 Diffusion and Adoption of the Test and Requirements Techniques

The project manager, who agreed to work on the project conclusion issue, concentrated on testing. The action concurred with the efforts to concentrate our products on fewer hardware and software platforms, and the project manager was responsible for one of these. Unfortunately, the tool that was introduced with the test improvement project (section 2.1) did not integrate well with the MFC environment needed for the project.

The philosophy from PET, however, that a project should achieve a certain level of testing before release was permanently in the minds of the team, and it determined the way that the testing process was organised. A specific group of people was assigned to testing the products on the platform, and they managed to achieve the necessary credibility from the developers and project managers, so that the products were not released until accepted by them. They automated the test suites and are currently investigating tools to reintroduce static and dynamic analysis in their process as advocated by PET.

Three projects decided to work with the requirements techniques (section 2.2), and later another project that worked on the iterative software development model adopted the techniques too. The teams were trained in the techniques, and they immediately accepted them with great enthusiasm. Later as the requirements phase progressed, the project managers and the teams continued to find the techniques effective. The mentoring and support from the support group ensured that the techniques were used properly by the projects.

The projects have released successful products to the market. It is too early to report sales figures, however. The techniques are now accepted by all project managers as the standard way of conducting the requirements phase at Brüel & Kjær.

4.2 The Software Development Model Improvement Actions

The most important problem perceived by the project managers was the existing waterfall software development model. They felt that it was inconsistent with the way they actually wanted to develop their products. They needed a new development model based on iterations, e.g. some form of prototyping.

One of the project managers, who agreed to work on the development model, wanted to extend the prototyping principle used in the requirements phase to the implementation phase also, e.g. frequent functional prototypes for user validation. When this model was defined, two more projects wanted to adopt the model too. The other project manager, who agreed to work on the development model, wanted to drive the iteration further and base it on the principle of time-boxing from RAD.

The process improvement group worked closely with the projects to define the new iterative software development models. The models were initially presented in draft form for the teams. As the development progressed and experience was gained, the models were revised and amended. The mentoring and support from the support group turned out to be essential to the successful use of the models. There seemed to be very little resistance to the improvements recommended by the support group. In fact there was great enthusiasm in the teams for the development model improvement actions.

We found that introducing an iterative software development model clearly had an effect on the involved projects. We have not been able to find a way to quantify the effects. However, through interviews and observations we have been able to establish the following statements:

- drive and motivation of team increased dramatically
- turbulence within and around team decreased
- control over project progress increased
- requirements creep was kept under control
- resources were used more efficiently
- serious problems were uncovered early
- quality improved step-by-step

The project managers involved in the improvement action have clearly expressed that they want to continue to use these iterative software development models. However, more experience with the models is needed before they can be deployed as a new standard software development model at Brüel & Kjær.

4.3 Unsuccessful Actions

The reuse improvement project was initiated, but the group responsible for this never received the necessary resources and support to get the project going. This is in fact quite natural since the item was added by the improvement group and not perceived as a problem by the project managers. We are a level 2 company, and reuse is an organisation wide activity (level 3).

Similarly the project monitoring improvement project did not succeed. The R&D manager acquired and implemented a tool to support estimation and monitoring activities. But since he did not press the project managers to use the tool, which by the way also had quality problems, the project managers to a very large extent simply ignored the tool. The improvement group tried to establish a support group, which should assist in getting the project managers to accept the tool, but the R&D manager felt that he was too busy to take part in such an activity. The failure of this improvement project is also quite natural. A tool by itself never solves a process problem, you need methodology and organisational support, and this was not achievable at our level of maturity.

The conclusion that can be drawn from these improvement actions is that only those improvements will be successful that are perceived by the project managers as solving their problems. Neither the introduction of tools nor organisation wide actions will be effective in an organisation at maturity level 1-2 unless they align with the "power structure".

5 Did We Improve Our Maturity?

After the improvement actions mentioned in the previous sections had been completed a second Bootstrap assessment was performed in order to check on the results and further the improvement programme.

The Bootstrap assessment was performed by the same assessors from DELTA. Four projects and the software development management were interviewed by the external assessors. The overall result of the assessment was still level 2.25.

This immediately seems like no improvement. However, when you study the detailed results that lead to this number, you can see that the first assessment result was only a little more than halfway between 2.0 and 2.25 (and the final result therefore achieved through rounding), the second is straight at the level.

More important, however, than this number was the changes in recommendations made by the assessors. This time they pointed out weaknesses in the following areas:

- Configuration Management
  The assessors commented on the need for a more uniform way of performing configuration management, change control, and planning.

- Project Management
  Improve planning, estimating procedures, introduction of time and resource usage, closer monitoring of project progress.

- Management Responsibility
  Improve status reporting, roles and responsibilities, common policies and strategies.

- Architectural Design
  Introduce interface design methods and standards.

- Process Measurements
  Introduce measurement programme and procedures.

- (Unit and Integration Testing)
  The assessors acknowledge that the present level of testing seems adequate for the quality of the products in the market.

These recommendations clearly show that those improvement actions that the project managers have undertaken w.r.t. development model and requirements are no longer on the list. The testing recommendation has been moved to the bottom position of the list, and the assessors acknowledge that the testing performed seems to be adequate for products like ours, even though it is not performed fully according to the maturity model.

The recommendations from the previous assessment on configuration management and project management that we had either ignored in our improvement actions (configuration management) or not succeeded in getting to work (project management) are still on the list and now at the top of the list.

Shortly after the second assessment, we performed another round of interviews with the project managers, e.g. repeated the problem diagnosis. We wanted to find out whether the improvement actions had had an effect on the (process) issues that they perceived as important to them. Four project managers were interviewed in this round.

The conclusions from the interviews are in remarkable concordance with the conclusions from the Bootstrap assessment above: development model and requirements issues are no longer on their list. Testing issues have been reduced in importance. The top issues on the list are configuration management and project management. There is also an agreement on two of the new issues: management responsibility and design.

Based on these new recommendations, two further improvement actions have been initiated. One improvement action addressing configuration management, and one action attempting to revitalise the acceptance and use of the estimation and monitoring tool, this time focusing on project feed-back and management reporting, e.g. addressing the organisation around the tool.

Due to a change in R&D management the intention is to perform these improvement actions with much more higher-level management attention and involvement, e.g. an improvement style suiting a level 3 maturity organisation. It remains to be seen whether this approach will be successful.

6 Comparison of Approaches

The recommendations from the different approaches to SPI that we have performed show a great deal of overlap though they sometimes use different words.

Recommendation	Defect Analysis (section 2)	Bootstrap Assessment (section 3)	Project Mgr. Interviews (section 4)
Development Model			
- iterations	x	x	x
- risk management			x
Requirements	x	x	x
Project Monitoring			
- estimation		x	x
- time & resource usage		x	
- monitor progress		x	x
Project Conclusion			
- configuration mgmt.		x	x
- testing	x	x	x
- release criteria	x		x
Reuse			x
Process Descriptions		x	

What is important, however, is the fact that the findings from the defect analysis approach are recommended by the more general evaluations. This means that a problem diagnosis approach to SPI based on the available error logs in the organisation is a valid approach to initiate a process improvement programme. And we have seen that improvement actions based on the defect analysis approach actually lead to improvements that changed the recommendations in a formal Bootstrap assessment.

The majority of the assessment recommendations are also found by the approach based on project manager interviews. This means that a problem diagnosis approach to SPI based on the (process) issues perceived by the project managers is also a valid approach to initiate or sustain a process improvement programme. And we have seen that improvement actions, which are adopted by the project managers, actually lead to improvements that changed the recommendations in a formal Bootstrap assessment.

7 Conclusions

From the above comparison of approaches and the results of our improvement actions, we conclude that problem diagnosis approaches to SPI based on already available experience in the organisation are effective and valid alternatives to formal assessments for organisations at maturity level 1-2.

Normative models for SPI like CMM and Bootstrap should not be discarded because of this. They represent a comprehensive framework for improvement actions and have established a way of measuring (assessing) whether improvement actions have been successful within an organisation.

However, their basic principle of establishing levels of maturity, and consequently the recommendation for organisations to improve from one level to the next until the top level is reached, seems to be very difficult for organisations to accomplish. This may discourage many organisations from improving their processes at all.

Alternative approaches to SPI like the ones we have employed tend to be hot-spot solutions to very specific problems. However, they have a high motivational effect because of the immediate quick wins they bring. This will encourage the organisation to pursue a continuous line of improvement actions, and as we have seen, completely in sync with the key process improvement areas of the normative models.

Irrespective of the SPI approach, it is our experience that in order to achieve a widespread adoption of process improvements in an organisation, it is necessary that the improvement actions are defined by or co-ordinated with the powerful actors in the organisation, which at a level 2 maturity are the project managers. Recommendations from a Bootstrap assessment or dedicated improvement actions will only be effective if they are aligned with the "power structure" of the organisation.

8 References

[1] Beizer B., *Software Testing Techniques. Second Edition*, Van Nostrand Reinhold New York, 1990.

[2] Bicego A., Khurana M., Kuvaja P., *BOOTSTRAP 3.0: Software Process Assessment Methodology*, Proceedings of the SQM'98, 1998.

[3] Bollinger, T. B. and McGowan, C., *A Critical Look at Software Capability Evaluations*, IEEE Software, Vol. 8, No. 4, pp. 25-41, 1991.

[4] Iversen, J., Nielsen, P. A., and Nørbjerg, J., *Problem Diagnosis in Software Process Improvement*, Proceedings of the IFIP WG8.2 & WG8.6 Working Conference, Helsinki, Finland, 1998. (http://www.bi.no/dep2/infomgt/wg82-86/proceedings/iversen.pdf)

[5] Mathiassen L., *Beskrivelse af forskningsprojekt om: Softwareprocesforbedring*, Aalborg Universitet, DK-9220 Aalborg Øst, Denmark, 1996.

[6] Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V., *Capability Maturity Model for Software, Version 1.1*, 93-TR-024, Software Engineering Institute, Pittsburgh, Pennsylvania, 1993.

[7] Vinter O., Poulsen P.-M., Nissen K., Thomsen J.M., *The Prevention of Errors through Experience-Driven Test Efforts. ESSI Project 10438. Final Report*, Brüel & Kjær A/S, DK-2850 Nærum, Denmark, 1996. (http://www.esi.es/ESSI/Reports/All/10438).

[8] Vinter O., Poulsen P.-M., Nissen K., Thomsen J.M., Andersen O., *The Prevention of Errors through Experience-Driven Test Efforts*, DLT Report D-259, DELTA, DK-2970 Hørsholm, Denmark, 1996.

[9] Vinter O., Lauesen S., Pries-Heje J., *A Methodology for Preventing Requirements Issues from Becoming Defects. ESSI Project 21167. Final Report*, Brüel & Kjær Sound & Vibration Measurement A/S, DK-2850 Nærum, Denmark, 1999. (http://www.esi.es/ESSI/Reports/All/21167)

Appendix: CV of Author

Otto Vinter is a project manager specialising in software process improvements with the Danish software engineering consulting company DELTA. Until recently he was responsible for software process improvements at Brüel & Kjær. He has been active in defining software engineering standards, procedures, and methods to be employed at Brüel & Kjær. He has been the driving force in the company's improvement activities in testing, requirements engineering, development models, and introduction of object-oriented development methods.

He has managed software development projects for 30 years; with Brüel & Kjær from 1986, before that with the Danish branch of Control Data Corporation, and with Regnecentralen.

The author received his Masters Degree in Computer Science from the Danish Technical University in 1968. He is an associate teacher for BSc. level education in Computer Science, is an active participant in Danish software knowledge exchange groups, is on the programme committee of several international conferences, performs mentoring activities for clients, and is an expert evaluator on the framework programmes of the CEC.

QW2000 Paper 9T2

Mr. Tim Szymanski
(Advanced Software Technologies, Inc.)

Quality Starts with Requirments: How the UML Can Help

BACK TO QW2000 PROGRAM

Key Points

- Setting requirements should be the first significant engineering work product of any software development lifecycle.
- A quality process for gathering and documenting an undestanding of the problem to be solved is essential to attaining quality products throughout the development lifecycle.
- Using the Unified Modeling Language (UML) and a light weight process, the development team can maximize quality during a very early phase of the lifecycle and leverage those results throughout the project's lifetime.

About the Author

Mr. Tim Szymanski is an expert in object oriented technology. As a known speaker and publisher in the field, Mr. Szymanski brings a fresh and down-to-earth approach to implementing OO technology.

Mr. Szymanski currently supports technical sales and OO consulting with Advanced Software Technologies, a leader in UML modeling technology and OO training.

Mr. Szymanski began his career with 8 years as an officer in the United States Air Force, involved in a variety of sophisticated military and communications technology. Mr. Szymanski's work experience also includes: Reltec Corporation and Gateway Computers where his area of focus involved process improvement, implementation of UML technology and commercial application development.

Mr. Szymanski graduated from the United States Air Force Academy in 1990 with a Bachelor of Science degree in Electrical Engineering. In 1994 he received a Master of Science degree in Computer Science from California State University Sacramento. He may be contacted at: tszymanski@advancedsw.com.

BACK TO QW2000 PROGRAM

Quality Starts with Requirements How the UML Can Help

Quality Week 2000

June 2, 2000

Tim Szymanski
Senior Consultant
Advanced Software Technologies, Inc.

Introduction

- SRA – The Weak Link
- The Goals of SRA
- Problem Space vs. Solution Space
- A Recipe for Success
- Results

© 2000 Advanced Software Technologies, Inc.

SRA – The Weak Link

- A Plethora of Incredible Tools Exist in Today's Software World
- Many More Incredible Tools are Right Around the Corner
- Many Projects are Still Late and/or Over Budget

© 2000 Advanced Software Technologies, Inc.

Who Wants to be a Millionaire?

What Factor Most Often Causes a Project to Encounter Hurdles?

- A Technical Barrier
- Long Lunches
- Insufficient Understanding of the Problem to be Solved

© 2000 Advanced Software Technologies, Inc.

The Solution

- Better Software Requirements Analysis
- However…
  - "We Don't Have Time"
  - "We Don't Have the Resources"
  - "The Requirements are not Yet Known"

© 2000 Advanced Software Technologies, Inc.

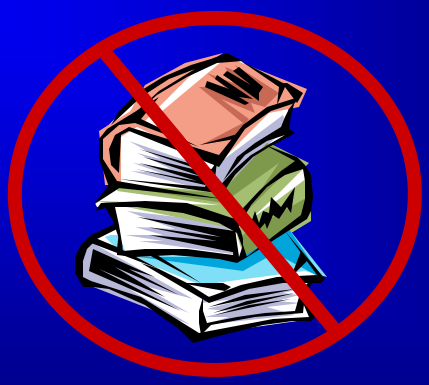
Once Again…

What Factor Most Often Causes a Project to Encounter Hurdles?

- A Technical Barrier
- Warm Springtime Weather
- Insufficient Understanding of the Problem to be Solved

© 2000 Advanced Software Technologies, Inc.

The Goal of SRA



- The goal is not to produce as much documentation as possible.

© 2000 Advanced Software Technologies, Inc.

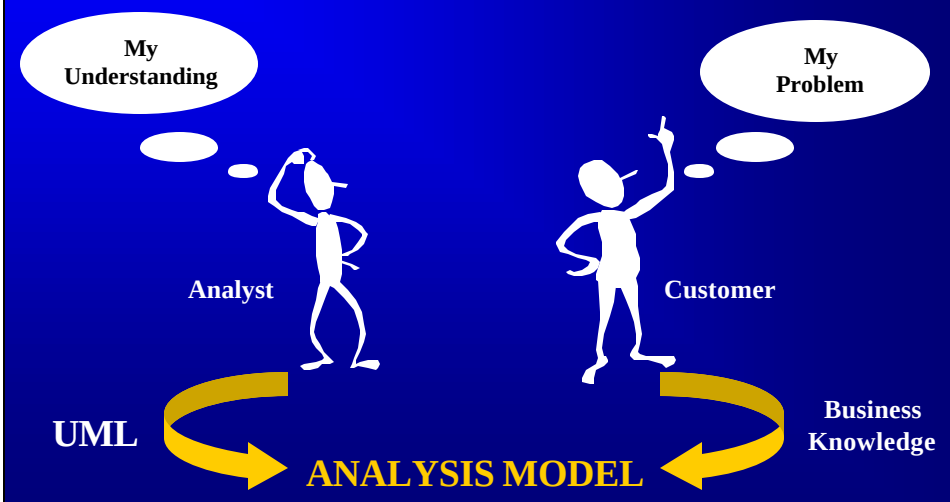
The Goal of SRA

- The real goal is to capture an understanding.

- The UML is the perfect tool to capture that understanding!

© 2000 Advanced Software Technologies, Inc.

SRA - The First Step



© 2000 Advanced Software Technologies, Inc.

SRA – The First Step

- A Picture is Worth 1000 Words
- Build an Analysis Model First
- Analysis Model Should Drive a Requirements Document
- If Need Be, the Analysis Model May Be Your Requirements Document

© 2000 Advanced Software Technologies, Inc.

What is an Analysis Model

- Class Diagrams
  - Business Entities and Relationships
- Use Case Diagrams
  - Business Functionality
- Activity/State Diagrams
  - Business Processes and Logic

© 2000 Advanced Software Technologies, Inc.

What is an Analysis Model

- An Analysis Model Captures Required Business Information and Behavior

© 2000 Advanced Software Technologies, Inc.

Tie It All Together

- A Proper Analysis Model Can…
  - Minimize Project Risk
  - Maximize Project Quality
  - Build a Foundation for a Complete SRA
  - Substitute for Detailed Requirements when the Schedule is Tight

© 2000 Advanced Software Technologies, Inc.

Building an Analysis Model

Problem Space

Vs.

Solution Space

© 2000 Advanced Software Technologies, Inc.

Problem Space

- Business Rules
- Business Entities
- Business Relationships
- Business Processes and Logic
- Any information or behavior that exists in the customer's world.
- 100% Implementation Independent

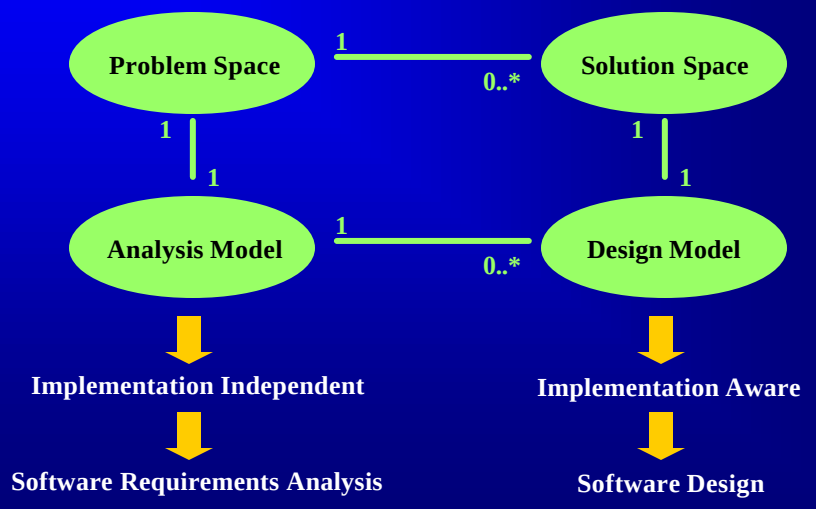
© 2000 Advanced Software Technologies, Inc.

Solution Space

- Components Working Together to Implement a Solution
- Many Solutions for any One Problem
- Most Frequent Application of the UML
- Must be Implementation Aware (I.e. C++ vs. Java, Unix vs. Windows)

© 2000 Advanced Software Technologies, Inc.

Problem Space / Solution Space



© 2000 Advanced Software Technologies, Inc.

Remember…

- Software Requirements Analysis often overlaps with business process design.
- Designing business processes is still very different than designing software.

© 2000 Advanced Software Technologies, Inc.

A Repeatable Process

- Build a Domain Model
- Identify Use Cases for Functional Requirements
- Identify Use Cases for Interface Requirements

© 2000 Advanced Software Technologies, Inc.

A Repeatable Process

- Model Complex Interactions Between the User and the System
- Add Detail to the Domain Model

- Identify Requirements Statements (rules) that Govern Each Use Case

© 2000 Advanced Software Technologies, Inc.

Domain Model

- Defines a Mutual Understanding of the Customer's World
- Identifies Business Entities and the Relationships Among those Entities
- 100% Independent of Design and Implementation Details

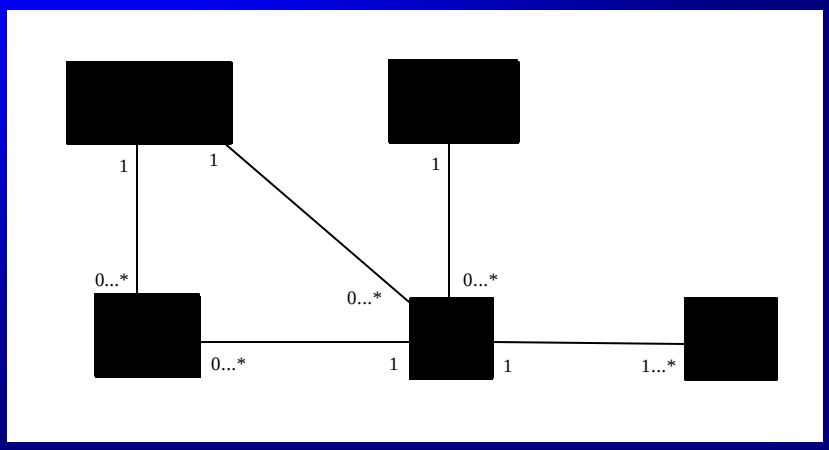
© 2000 Advanced Software Technologies, Inc.

Example:

"I need a system that allows me to track issues as identified via e-mail by our customers. For each issue I need to track activities performed by support reps while trying to solve the various issues."

© 2000 Advanced Software Technologies, Inc.

Domain Model



---Problem Space---

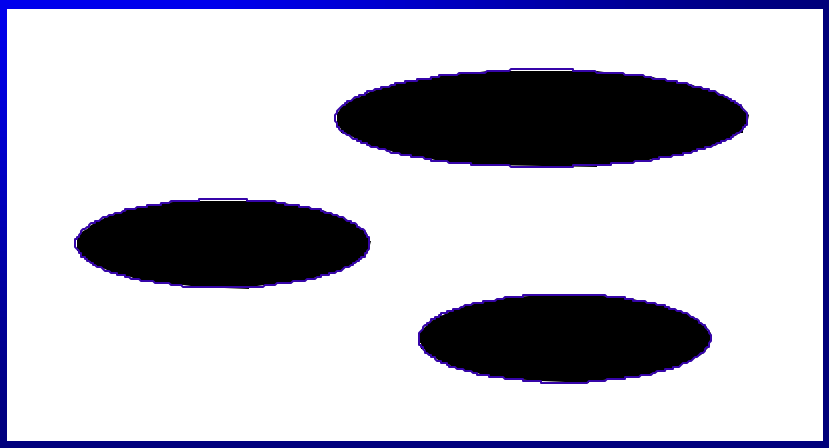
© 2000 Advanced Software Technologies, Inc.

Functional Capabilities

- For each entity on the domain model:
  - Create - Does the system need to allow the user to create new instances?
  - Review - Review Existing Instances?
  - Update - Update Existing Instances?
  - Delete - Delete Existing Instances?

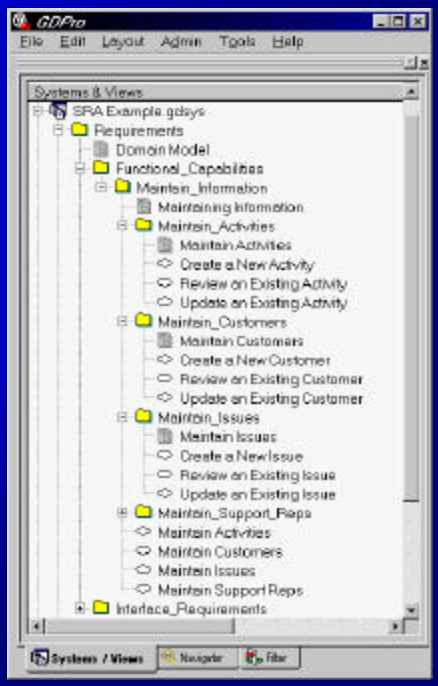
© 2000 Advanced Software Technologies, Inc.

Functional Capabilities



© 2000 Advanced Software Technologies, Inc.

Project Organization



Interface Requirements

- Does the user have any specific requirements regarding the system's interface?
- If so, develop use cases that describe these interface requirements.

© 2000 Advanced Software Technologies, Inc.

Example

"When a new e-mail is received, I want the system to lead me through a process. First I search to see if the customer already exists. Only if the customer exists do I proceed. I then search for an existing issue that matches the issue brought up in the new e-mail. If the issue exists, I ...

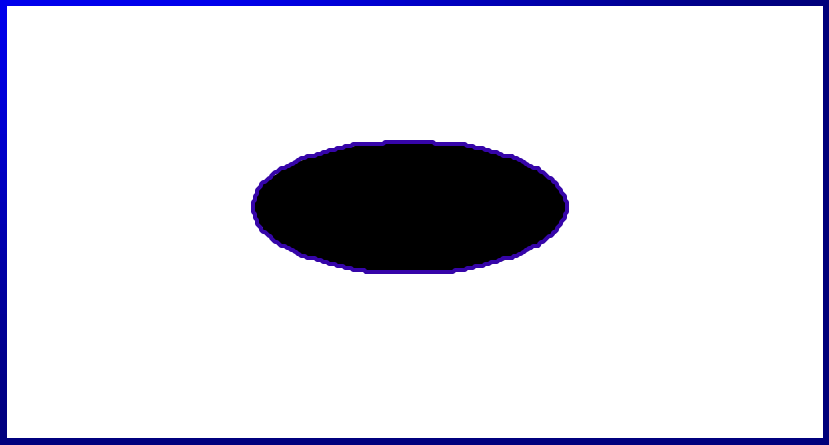
© 2000 Advanced Software Technologies, Inc.

Example

…attach the e-mail to the existing issue. Otherwise, I create a new issue and attach the e-mail. If I create a new issue I must also assign a support rep to handle that issue.

© 2000 Advanced Software Technologies, Inc.

Interface Requirements



© 2000 Advanced Software Technologies, Inc.

Model Complex Interactions

- Interface requirement use cases often have many paths that use many of the primitive use cases identified in the functional requirements.
- Current industry direction suggests writing scenarios.

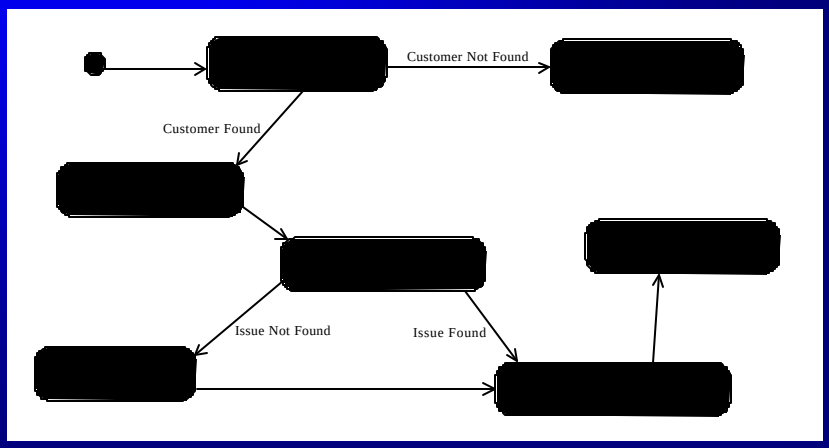
© 2000 Advanced Software Technologies, Inc.

Model Complex Interactions

- Activity diagrams and state diagrams are specifically designed to show multiple behavior paths that depend on various conditions.
- Draw one model that represents the required interaction as it pertains to the use case in question.

© 2000 Advanced Software Technologies, Inc.

Model Complex Interactions



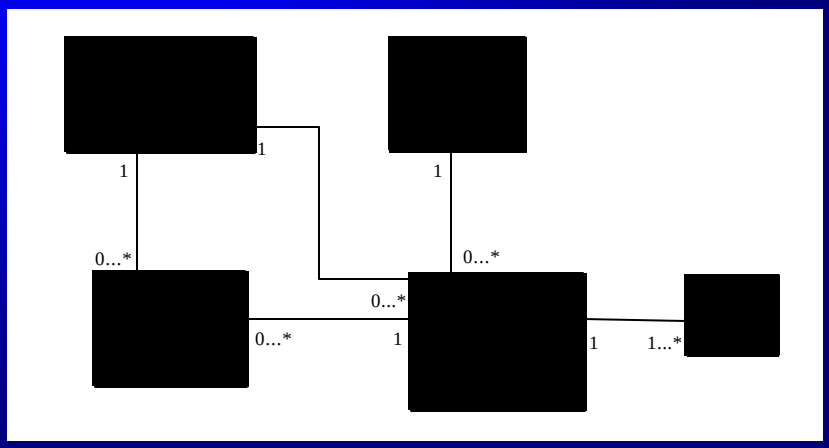
© 2000 Advanced Software Technologies, Inc.

Add Detail to Domain Model

- The original domain model only contained classes and associations.
- Fill in attributes for each class.

© 2000 Advanced Software Technologies, Inc.

Add Detail to Domain Model



© 2000 Advanced Software Technologies, Inc.

Identify Requirements Stmts.

- Most Difficult Step
- Requires Precision to be Valuable
- Questions MUST be Answered Sooner or Later
- Too Many Projects Leave this Step to the Programmers

© 2000 Advanced Software Technologies, Inc.

Identify Requirements Stmts.

- The system shall allow the user to create a new issue for any existing customer.
- When creating a new issue, the system shall require the user to supply an issue description.
  - Issue description shall be a minimum of 5 characters.
  - Issue description shall be a maximum of 500 characters.
- When creating a new issue, the system shall assign the current date to the the issue's date opened attribute.

© 2000 Advanced Software Technologies, Inc.

Process Review

- Domain Model
- Functional Requirements
- Interface Requirements
- Complex Interactions
- Domain Model Detail
- Requirements Statements

© 2000 Advanced Software Technologies, Inc.

Summary

- The Goal of SRA is to understand the customer's problem.
- Current industry direction suggests a subjective approach using use cases to model understanding.

© 2000 Advanced Software Technologies, Inc.

Summary

- Applying engineering discipline and process to use cases can produce a repeatable use case technique for understanding and modeling problems.
- 10 analysts working independently will produce similar results.

© 2000 Advanced Software Technologies, Inc.

Final Result

```
If Process.IsGood() = True Then
    Repeatable Process = Repeatable Success
End If
```

© 2000 Advanced Software Technologies, Inc.

QW2000 Paper 9A2

Mr. Jerrold Landau
(IBM Canada)

An Overview of Testing Methodology and Experience at IBM Corepoint Banking Solutions

BACK TO QW2000 PROGRAM

Key Points

- Web based application
- Real-life experience
- Java

Presentation Abstract

Corepoint solutions are a set of web-based components and reference applications designed primarily for the banking industry. Currently, there are four applications available: Internet Banking, Contact Center Management, Teller, and Branch Sales. These products, produced by the IBM Toronto Lab, are written in Java, and run on an AIX or NT server and an NT client. This presentation will provide an overview of the testing strategies and methodologies used by IBM to test these products, discuss our experience with automation, and outline the problems and challenges that are encountered in our testing effort.

At this point, the components are bound together in the four different reference applications, and are intended to be customized by an engagement team prior to being installed at a real customer site. We anticipate moving toward a paradigm of more generalized components in the future, which will defocus further from the end-user banking application. Of course, this will present a new set of challenges for the testing group.

Topics covered in this presentation include:

- The makeup of our testing department
- The system test process that we use, with examples of testplans and testcases
- A discussion of incident reporting and tracking
- A brief overview of Performance testing
- A brief overview of Component testing
- Our experience with Testcase automation
- A discussion of the problems, pitfalls and challenges that are encountered during our testing effort

It is not the aim of this presentation to present any new theories and methodologies to approach the testing effort. Rather, it will show how existing test methodologies are

adapted to fit a real life, dynamic, leading edge development environment, replete with the time constraints, plan changes, moving target, and other such challenges that all professional testers are accustomed to. Anyone who is interested in learning from our experience is welcome to attend this presentation.

About the Author

Jerrold Landau has been an employee of IBM Canada for over 9 years. He spent his first 8 years with the AS/400 Compiler group, being primarily involved in testing and customer support. For the last year and a half, he has been working as part of the testing group with the Corepoint division of the IBM lab.. Prior to joining IBM, Jerrold worked at two smaller companies, Cherniak Gottlieb and Netron, as a software developer-analyst.

Jerrold has a M.A.Sc. in Industrial Engineering from the University of Toronto, 1986, and a B.Sc. in Computer Science from the University of Toronto, 1984.

Jerrold has been involved with various aspects of the software development process during his career, and has always maintained a primary interest in testing and quality assurance. He is interested in enhancing the stature of the testing discipline within the software development environment.

Recently, Jerrold was the chairman of a workshop entitled ôPractical Experience with Quality Assurance at the IBM Toronto Labö, presented at the CASCON 1999 conference in Toronto.

BACK TO QW2000 PROGRAM

Software Testing at the IBM Toronto Lab
(Corepoint Banking Solutions)

Jerrold Landau, M.A.Sc.
landau@ca.ibm.com

copyright: Jerrold Landau, IBM Canada, 2000

Topics of Discussion

- Organization of Test Department
- Corepoint Product Suite
- System Test Process
- Reporting of Results
- Unit, Component, and Performance Test
- Automation
- Testing Challenges

Testing Organization

- **Separate Testing department (8 people)**
  - Manual System and Regression Testing testing
  - GUI based applications
  - various backgrounds
    - business
    - technical
    - co-ops
    - Borrow people from other teams during peaks
- **Unit Testing done by developers**
- **Performance and Scalability Testing**
- **Component Testing done by separate group**

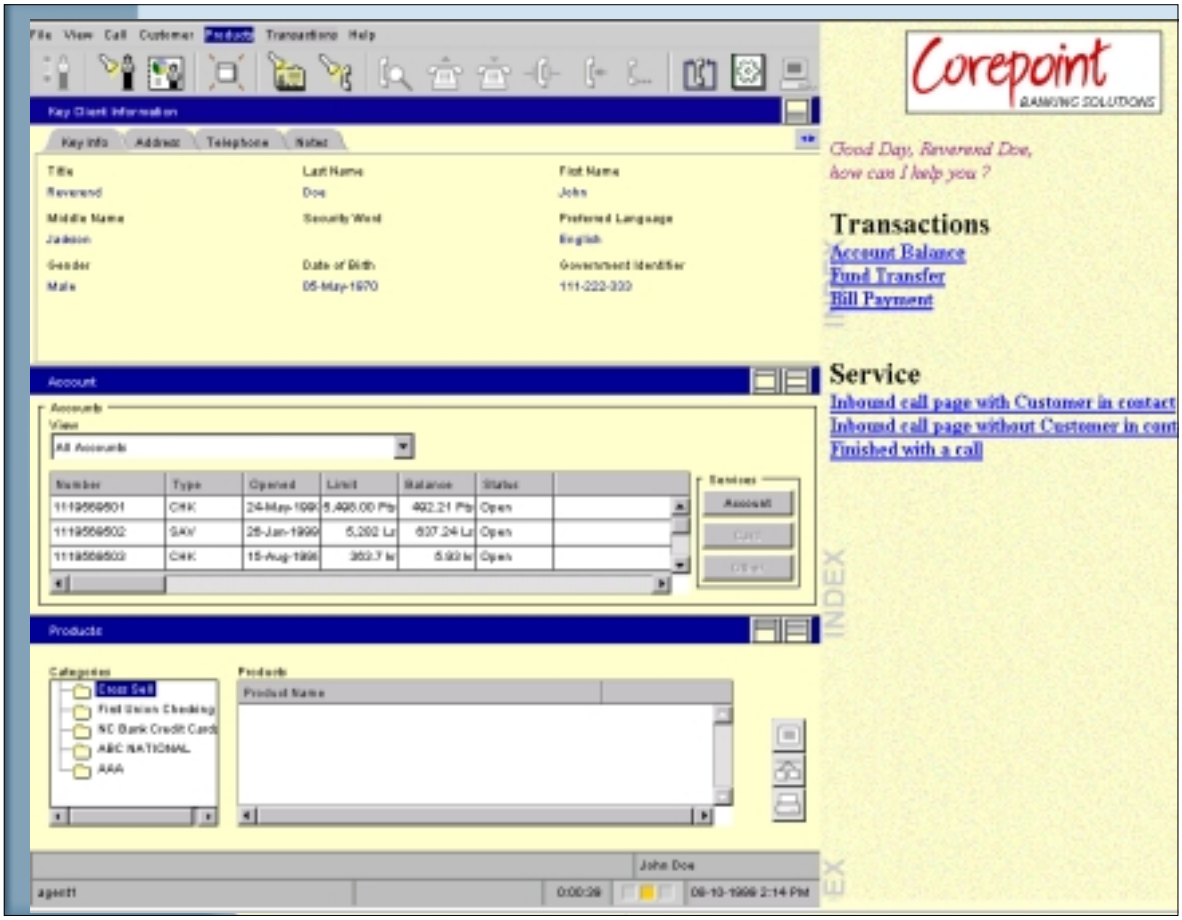
Corepoint Product Suite

- **Contact Center**
- **Internet Banking**
- **Teller**
- **Branch Sales**

- **These are Reference Applications, built on top of a set of frameworks**

Corepoint Product Suite

CONTACT CENTER V 1.4



Contact Center Prerequisites

Testing staff need to have a basic understanding of:

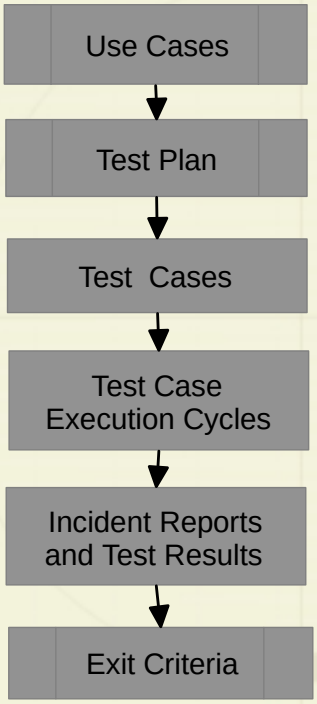
- Java environment (JDK, JVM)
- AIX operating system
- Websphere web server and Servlet Engine
- IBM TX Series (CICS, Encina, DCE)
- DB2 / UDB
- MQ Series
- Callflow
- Callpath
- COBOL
- Jetform Printing
- Domino Webserver
- Netscape Browser

Types of Testing

- Unit Test
- Component Test
- System Test
- User Acceptance Test
- Performance / Scalability Test
- Fixpack Testing

All of the above can be done in a manual or automated fashion

System Test Process

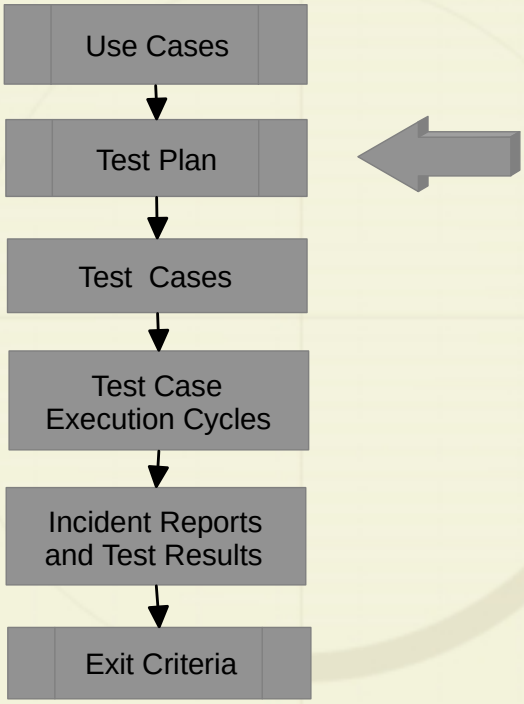


- ✓Tracebility throughout documents
- ✓Master Test Plan
- ✓Component Test Plans
- ✓Test cases tuned during test execution
- ✓Formal Sign Offs

Use Case - Funds Transfer

# (Main course) UCFND	Action By Actor	System Response
M1	requests to transfer funds	displays a view that allows the input of a funds transfer
M2	enters the appropriate funds transfer details, including - from account (mandatory) - to account (mandatory) - amount (mandatory) - date, fee, description, etc.	accepts the entries
M3	submits the funds transfer request	- accepts the submission - displays confirmation of the request - updates the appropriate accounts

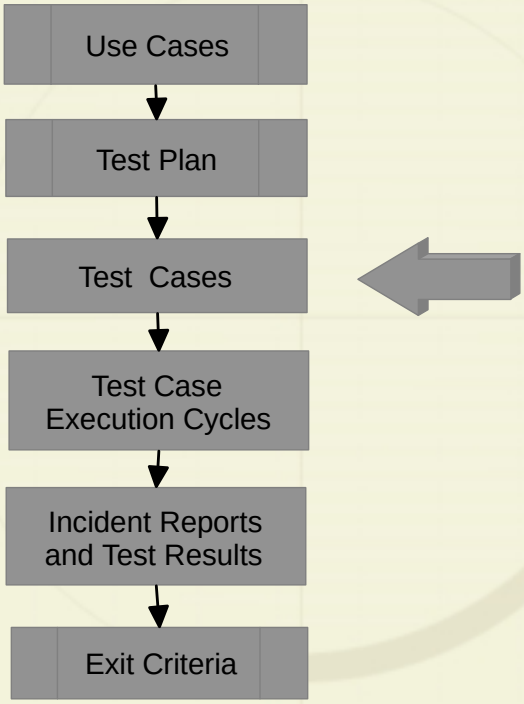
System Test Process



Test Plan - Funds Transfer

Test Requirement Number	Source Document	Use Case Reference	Main Alt Derived	Description	Specific Details
ai-ucfndtrn -01	draicc		M	Requests to transfer funds from one account to another	Transfers the money from the accounts using the Sample Banking Transactions Component and new transactions are added to the list of the transactions
ai- ucfndtrn -02	draicc		A	Sample Banking Transactions generates error for transfer	Handles the transaction error
ai- ucfndtrn -03	draicc		D	Requests to Cancel a transfer of funds from one account to another	Cancel handled successfully
ai-ucfndtrn -04	draicc		D	Funds Transfer Field Formats	Field Formats must meet requirements

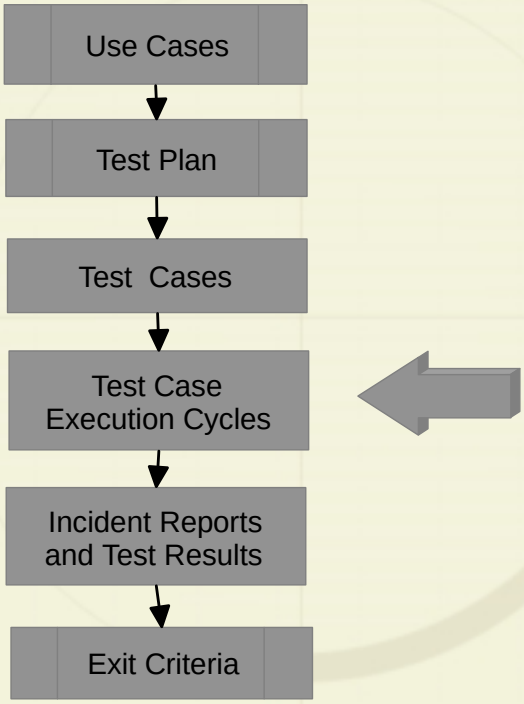
System Test Process



Test Case- Funds Transfer

Test Req #	Step	Test Procedure	Expected Result	P/ F I/U
ai-ucfndtm-01	1.	Select one of the bank accounts	The Account button is active. The *Card* and *Other* buttons are disabled.	
	2.	Select the Account button	Account Services Catalog is displayed with a list of available financial transactions	
ai-ucfndtm-01	3.	Select Funds Transfer from the list of available financial transactions	Funds Transfer option gets highlighted	
	4.	Select OK	Funds Transfer window displays account number selected in step 1 as a default in the Funds Transfer window.	
ai-ucfndtm-01	5.	Enter the following data: • From Account:: select the 1st account on list • To Account:: select the 2nd account on list • Amount:: enter valid amount (amount entered + fee <= balance) • Fee:: $25.00 • Description:: Enter a reason	When accounts are selected in the FROM and TO account field, a balance for each of the accounts is displayed in non-editable fields. All the data entered is accepted	
ai-ucfndtm-01	6.	Select Transfer	Transfer successful and confirmation window displays indicating the transaction details along with a confirmation number	
ai-ucfndtm-01	7.	Select OK	Confirmation window closes	
ai-ucfndtm-01	8.	Verify Account info is correct	All information is correct and complete	

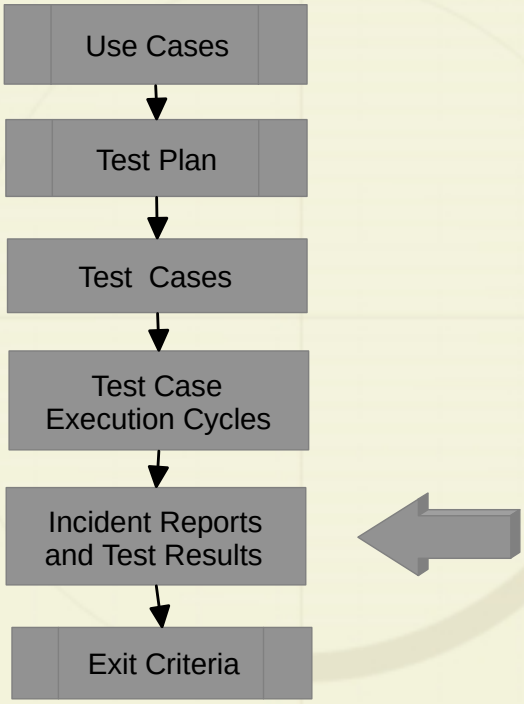
System Test Process



System Test Cycle

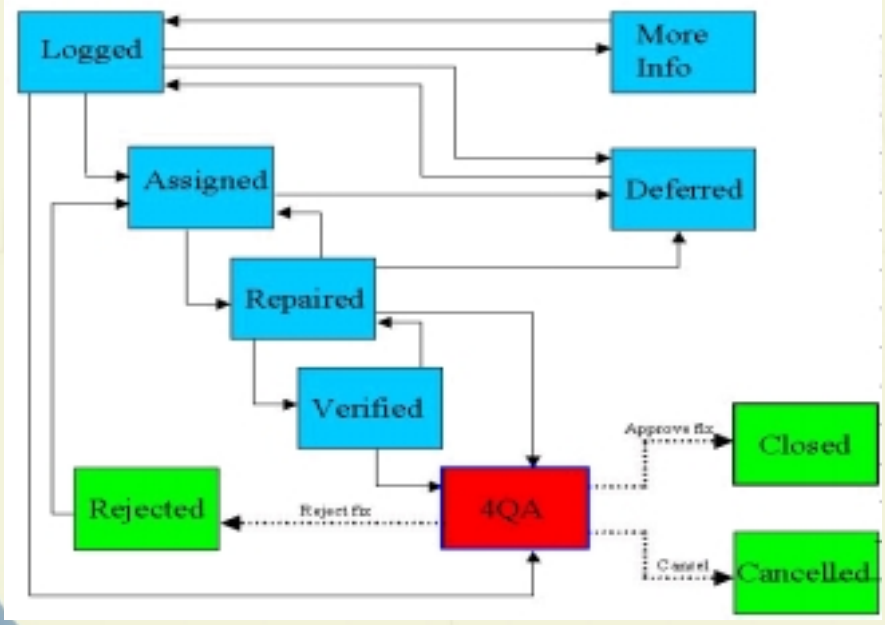
- Typically 6-8 builds per release at weekly intervals
- Schedule always subject to change
- Installation Test
- Focus on new functionality
- Verification of fixes
- Regression testing, tuned to the specific functionality provided in the build (typically full for 2 builds, partial for rest)

System Test Process



Incident Reporting System

- Lotus Notes - home grown application



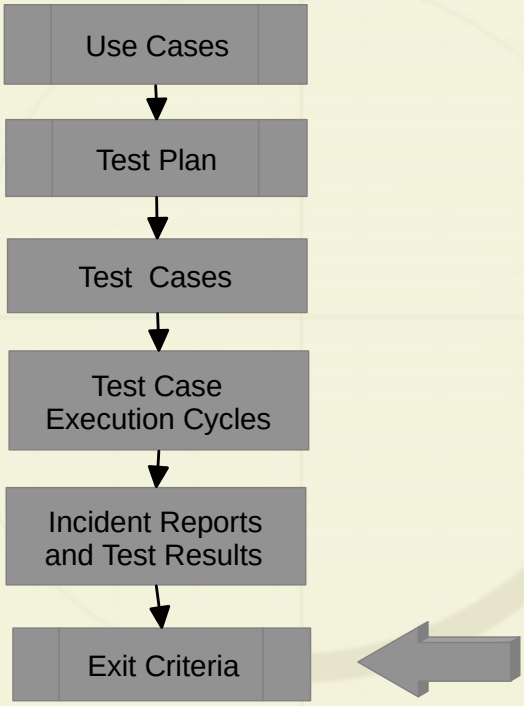
Incident Severities

- **Incident Reports Classified by Four Severity Levels**
  - Sev 1 -- Totally Inoperable. A critical component with a major business impact cannot be used because the required functionality does not exist.
  - Sev 2 -- Severely Inoperable. Functionality exists but is impaired or incomplete.
  - Sev 3 -- Partially Inoperable
  - Sev 4 -- Annoyance

Recording of Results (Metrics)

- **Manual data entry :o(**
  - Excel spreadsheets with graphs
    - test case execution - pass/fail
    - total and by component
- **Automated :o)**
  - Lotus 123 spreadsheets with graphs
    - defect reporting by priority

System Test Process



Exit Criteria

- **checklist of activities to be completed and criteria to be met prior to shipping**
- **criteria identified by stakeholders**

- **example of exit criteria**
  - 6 test cycles
  - 0 open sev 1, 10 sev 2, 3 sev 3
  - performance meets agreed to criteria
  - release notes describe all outstanding problems
  - all documentation items completed

Sign Off

- Release notes describing existing problems prepared for users
- Sign Off Document outlining status of product prepared for stakeholders
- Meeting of stakeholders called to discuss exit criteria
- Signoff solicited from all stakeholders

Component Testing

- Test cases written in Java to test functionality of component
- Test plan necessary for each component
- JUnit (freeware) used to provide testing framework, automation and result logging
- Assert statements to check results
- Important to test correct inputs as well as incorrect inputs
- Tests can be run singlely or as a batch
- Problems logged just as in system test

Performance and ScalabilityTest

- Insert code hooks to provide time stamp
- Run typical business scenario
- Obtain timings for key checkpoints
  for single user, 50 users, and 100 users
- Compare timings with benchmarks provided by client expectations
- Two week session at Dallas lab for 500+ users

Performance Test Example

Checkpoint	Run 1	Run 2
Login	3016	2758
Customer Search	531	510
Customer In Context	4265	3687
Account Inquiry	1256	1073
Funds Transfer	2448	3762
Wrap Up Customer	3926	3548
Total	15442	15338

- times in milliseconds
- times are average of several runs
- slight overall improvement but severe degradation in one area

Automation

- **Goals**
  - to be able to run test suites more frequently, and thereby to catch errors sooner
  - eliminate ambiguity in interpertation
  - free up testing staff for more in-depth testing

Automation Tool

- **Segue Silk Test (GUI based tool)**
- **Manages test plans and test suites**
- **Provides charts and graphical reports**
- **You must capture declarations prior to writing test cases**
- **Test cases written in 4Test language**
- **You can write or record the test cases**
- **You must insert error checking into test cases to determine pass/fail**

Automation - Our Experience

- Pilot project completed successfully
- Test suite with 100 test cases produced
- Very expensive and time consuming
- Long training period, much education needed
- Chasing current technology -- Netscape support as well as recognition of Java objects
- Product suite evolution makes benefits questionable

Not advisable to completely automate

Testing Challenges

- Automation
- Constricting of timeframes
- Moving Target
  - feature creep
- Home grown reporting tool
  - manual tallying
- Fixpack procedures not in place yet
- Fluid exit criteria
- Balance between quality and market pressure

Summary

- Team of people with diverse backgrounds
- Test GUI applications (CBS)
- Follow proven testing methodologies
- Unit, Component and Performance Testing
- Automation vs Manual Testing

Lots of challenges

Software Testing at the IBM Toronto Lab

(Corepoint Banking Solutions)

Jerrold Landau, M.A.Sc.
landau@ca.ibm.com

QW2000 Paper 9W2

Mr. Stephen Kingston
(Watchfire)

The Web Application Process: Development & Testing

BACK TO QW2000 PROGRAM

Key Points

This presentation will review every stage of application testing, from user requirements to deployment and the various forms of testing involved:

- Maximizing your web environment
- Importance of SQA
- Importance of Qaulity Analysis

About the Author

Stephen Kingston's responsibilities include establishing the direction for Watchfire's solution line up and facilitating the working relationships among cross-functional groups involved in product development. Stephen joined Watchfire in 1999. His prior experience includes nine years at JetForm and then at Corel in various customer-oriented roles. At Watchfire, Stephen has been instrumental in building the Product Management Team and implementing the process infrastructure of the Product Development Lifecycle.

BACK TO QW2000 PROGRAM

Functional and Regression Testing of Web Applications

Stephen Kingston
Director, Product Management
Watchfire

Presentation for: Quality Week 2000

Overview

- Web application testing is evolving as sites become more complex.
- Automation of some functions is essential.
- Cross-functional teams and related requirements have a significant impact on testing websites.

“Brochure ware” Web Architecture

The “olde days”:

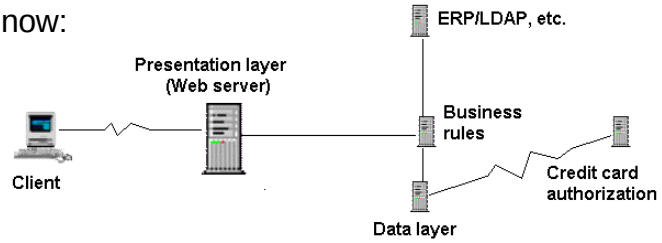
The website was the web-based application.

Testing requirements:

Simple, page by page link checking - not many pages

Applications-based Web Architecture

Websites now:



The website can be huge and act mainly as a front-end to a complexity of various applications.

Overall testing goal:
Need to ensure that the whole thing works together from the customer’s perspective.

Main Testing Requirements

- Can the customer navigate your site?
  - **Link testing/Integration testing**
- Is site performance adequate? Do pages load fast enough?
  - **Stress testing/Performance monitoring**
- Is the site and its application infrastructure accessible?
  - **Infrastructure monitoring**

Main Testing Requirements cont'd.

- Do the applications work on your site from the customer's perspective?
  - **Functional testing**
- Does the site still work properly after changes?
  - **Regression testing**

Functional Testing

During development, during QA and UA, and once a site is operational, web-based applications must be tested and monitored for their operational functionality.

- Do key page elements have the correct properties?
  - **Are forms, graphics and links structured properly for customer interaction?**
- Are forms interacting properly with the application?
  - **Are values or error messages generated?**

Functional Testing contd.

- Do different values, including invalid values, input into forms produce expected results from the back-end?
  - **Are the applications business rules properly reflected by the web front-end?**
- What are the page load times for key web/back-end interactions?
  - **Is the application performing adequately on average and on a regular basis?**

Regression Testing

As new builds are produced, and as the production site is updated:

- Do key page elements still have the correct properties?
- Are forms still interacting properly with the application?
- Do different values, including invalid values, input into forms still produce expected results from the back-end?
- Have the application's business rules changed and are these changes properly reflected by the web front-end?

Regression Testing contd.

- What are the page load times for key web/back-end interactions?
- Are links to other parts of the site and to external sites still functioning?
- Were any old files left over from the last build?
- Have old problems that were previously fixed been re-introduced?

Automated Testing: Why automate?

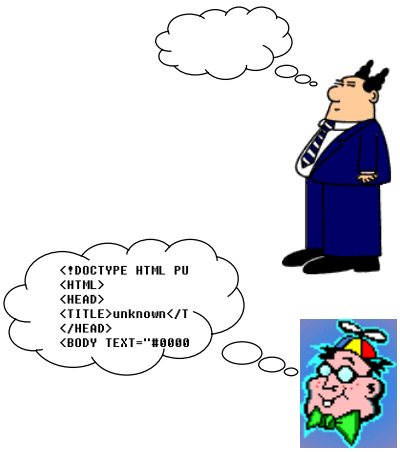
Time - Resources - Scope

- Time to market for Web applications -- manual testing takes too long
- HR crunch -- not enough people to test everything
- Not enough time to run through all permutations and combinations of possible values
- Repetitive testing of the same functions after any changes to the front-end or back-end
- Repetitive testing of the same functions with different input values

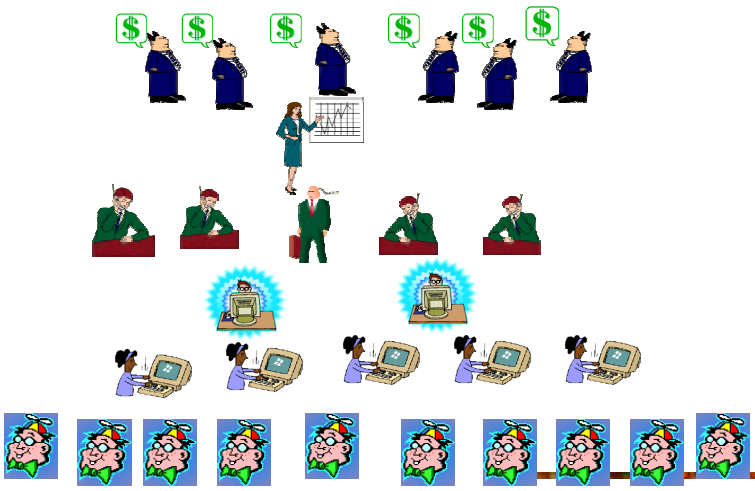
Automated Testing: What to automate?

- Page elements verification
- Content verification
- Performance - page loads, response times
- Forms input - multiple values, data input capabilities
- Browser interactions - testing from the customer's perspective

Old Corporate Web Organizations



New Corporate Web Organizations



Cross-Functional Web Teams – Why have them?

- **SDLC**
  - Many players with key responsibilities for the application throughout the product development, testing and release cycle
- **RAD**
  - Short development cycles require that joint development teams test throughout development process
- **User Acceptance Testing**
  - End-to-end testing of final application to ensure all applications work as expected from the customer's perspective (i.e. web front-end)

Cross-Functional Web Teams – Who are the Members?

- **Management**
  - Bottom line - revenues or cost savings
- **Product/Project Management**
  - User acceptance of the complete application
- **Operations**
  - 24x7 - site accessibility; site functionality

Cross-Functional Web Teams – Who are the Members?

- **Quality Assurance**
  - Testing of all aspects of application
- **Development**
  - Unit testing during development and end-to-end testing of pre-Alpha

Cross-Functional Testing Requirements

- Ease of use
- Web-centric
- Sharing of scripts and reports across teams
- Geographic diversity
- Historical reporting - trends over time
- Scheduling/monitoring
- Performance bench marks

Wrap-Up

- Complexity of web sites means testing is complicated and involved
- Testing from the user's perspective is paramount
- Time/resource constraints make automated testing essential
- Complex organizations and rapid development make testing a cross-functional responsibility
- Cross-functional web teams have varying requirements for testing and related solutions

QW2000 Paper 9M2

Mr. Richard Kasperowski
(Altisimo Computing)

Opportunistic Software Quality (9M2)

BACK TO QW2000 PROGRAM

Key Points

- Opportunistic software testing: a strategy for improving quality with limited resources
- Examples from a recent project
- Data and results

Presentation Abstract

I recently worked on a project for which I was the sole QA person and the ship date was only three months away. The project was to build and deliver a web-based billing system for the local telephone customers of a large telephone company. The team used HTML and JavaScript to present bills, a relational database at the back end, and Java and Enterprise Java Beans in the middle.

The development team consisted mainly of highly skilled designers and programmers who knew little about quality assurance. They were motivated to deliver a great product on time. My job was to help them do that, by inventing and executing a QA program for the project.

I was the only team member concerned primarily with product quality, and the delivery date was firm. I didn't have time to develop a well thought-out plan to assure the goodness of the product. Instead, I kept my eyes open for opportunities to improve product quality, and took advantage of these opportunities. I later realized that I have done this many times over the years, and began to think of my capitalizing on quality improvement opportunities as a general strategy--"opportunistic software quality."

Here are examples of the opportunities we discovered during this project and how they helped us deliver a high quality product on time:

- Configuration management: Prior to my joining, the team was already using a configuration management system. They used the system to track source code changes and to label the configurations that were delivered to their customer. I increased the rigorous use of this system, making it easier to identify the configuration that corresponded to the one the customer was using, and thus making it easier to fix the defects the customer identified.
- Bug tracking system: Again before I joined, the team put a bug tracking system in place. They used it sporadically and didn't record all defects in it. I became the manager of the bug tracking system, making sure all defects were recorded and addressed. We didn't forget to fix any of the

defects we needed to fix, so the product we delivered was better than otherwise.

- Automated nightly builds: Infrequent source code builds are a quality problem. "Code rot" can occur: individual programmers' code changes can break the compilation of another programmers' code. Broken builds can be difficult to repair if the defect was introduced too long ago for the programmer to remember why he changed the code. On previous projects, I found that it is possible to find defects simply by compiling the source code and building the product at regular intervals. On this project, I established a system for automatically building the product every night, after the programmers had gone home. The programmers agreed that they would only check-in source code changes that would at least compile correctly. We found a number of defects with this method. Because we found the defects the within a day of the code change, it was easier for the programmers to fix the defects than it would have been if we hadn't built the code regularly. In addition, builds usually succeeded when they were needed most, such as for an emergency patch release.
- Automated nightly testing: The programmers had built a number of semi-automated tests for particular sections of code. I fully automated the existing tests. Over time, I added tests of other important sections of code. I built a system for automatically executing the tests and analyzing the results. I augmented the nightly builds with the nightly automated tests. This also helped us find new defects within a day of a broken code change, making it easier for the programmers to fix the defects than it would have been if we hadn't built the code regularly.
- QA web site: The team's philosophy was that delivering a high quality product was a group effort, and not merely my responsibility. To give the other team members a view of the state of the product's quality, I built a QA web site for the team. The web site consisted of the most recent automated test results, a way to compare any set of test results with any other set of test results, hyperlinks to web pages that could be used for manual testing, and hyperlinks to written procedures. The test results section of the site made it easy for me to analyze nightly results and update baseline results. The hyperlinks to web pages that could be used for manual testing turned out to be extremely useful for the other team members; they vociferously complained whenever the site was down.
- Manual testing: It wasn't practical to automate all testing, especially GUI testing. I adapted an old test script so we could use it to test the current version of the product. I made it a policy that the script had to be executed at least once per week, and I rigorously followed the policy. With this regular planned testing, augmented by ad hoc testing from other team members, we identified many defects.
- Source code compiler: One of the team members recommended a better Java source code compiler. I modified the nightly build-and-test scripts to use the new compiler. The new compiler used a stricter definition of the Java language; we identified a few defects simply by using the new compiler.
- Improved delivery: One of my responsibilities was to deliver patch releases and beta releases to the customer. Before I joined the team, these deliveries were time consuming and plagued with mistakes. I developed a set of scripts to automatically build and install the product, and a procedure for making the delivery, drastically reducing the number of delivery mistakes.
- HTML validation: On a previous project, I used a tool to automatically check the validity of the HTML code the product delivered to users' web browsers. I used the tool on this project and successfully found a few defects.

There were a few techniques we considered using, but did not use. These were: automated GUI testing, code coverage analysis, exhaustively testing all source code, retrofitting pre-existing tests to a better test framework, and reorganizing the source code to improve building and maintenance.

Our results were good. The customer discovered only 7% of the total number of known defects. The two best techniques for identifying defects were manual testing and automated nightly testing. Manual testing was responsible for finding 46% of known defects, and automated nightly testing helped us find 29% of known defects. (These figures are slightly out of date; I will present current figures in the final version of the paper and at the conference.)

We delivered a relatively high quality product on time. The customer accepted the delivery and found very few defects. The opportunistic strategy worked.

Author Bio...

Richard Kasperowski is president of Altisimo Computing, a software development consulting firm based in Cambridge, Massachusetts. Richard has worked as tester, developer, manager, and consultant since 1988. He has a degree from Harvard University, is a member of the ACM, and usually cycles to his clients' offices.

BACK TO QW2000 PROGRAM

Opportunistic Software Quality

Richard Kasperowski
mailto:kasper@altisimo.com

Altisimo Computing
http://www.altisimo.com

© 2000 Altisimo Computing Corporation

Altisimo

Outline

- Introduction
- Opportunistic software quality
  - Definition
  - Examples
- Results

Altisimo

Introduction

- The project
  - Web- based billing system
  - HTML, JavaScript, RDBMS, Java, EJB
  - Deadline 3 months away
- My role
  - Sole QA person
  - Develop & enact QA program

Altisimo

Opportunistic Software Quality

- Strategy:
  - Identify quality improvement opportunities
  - Capitalize on them
- Identify: anything goes, as long as there's pay- back
- Capitalize: effective, persistent, reusable

Altisimo

Configuration management

- *Problem:* Need configuration management system
- *Opportunistic solution:*
  - CVS was in use during previous phase; continued use for current phase.
  - More rigorous use of labels
- *Results:*
  - CVS is good enough
  - Easier maintenance

Altisimo

Defect tracking system

- *Problem:* Need defect tracking system
- *Opportunistic solution:* JitterBug
- *Results:*
  - JitterBug is good enough
  - Not as good as low- end commercial systems

Altisimo

Automated nightly builds

- *Problem:* Source code rots if you build it infrequently.
- *Opportunistic solution:*
  - Scripts to build system unassisted
  - cron to automate and build nightly
- *Results:* Found new defects within 1 day of injection

Altisimo

Automated nightly testing

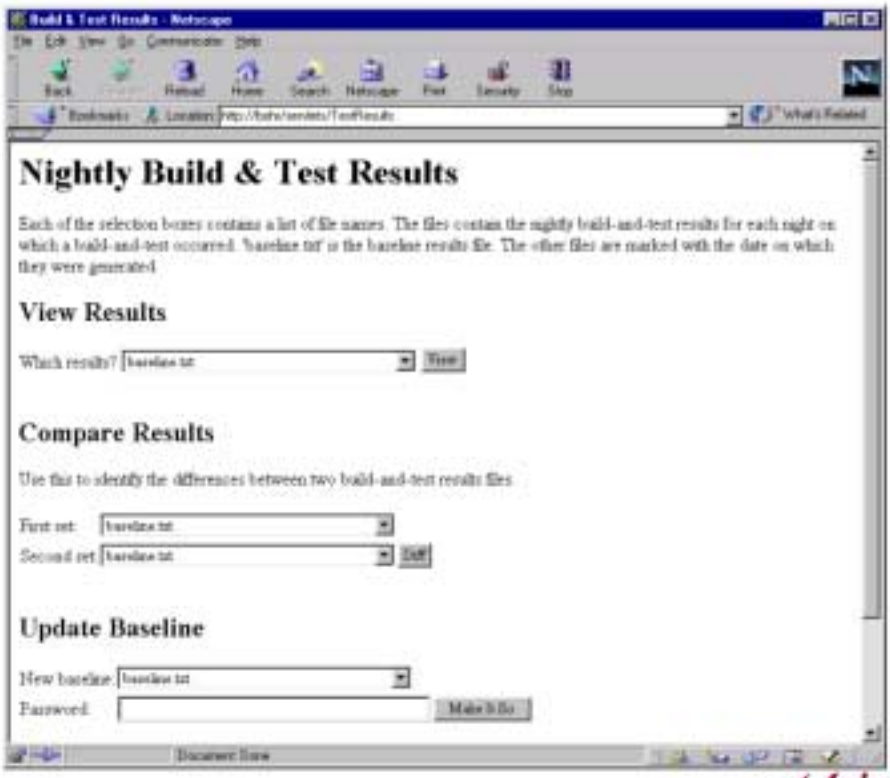
- *Problem:* Too many semi- automated tests to be able to execute them all regularly
- *Opportunistic solution:*
  - Wrapper script executes tests, savings results in log
  - diff & sed to compare results logs with baseline
  - Nightly build → nightly build- and- test
- *Results:* Found new defects within 1 day of injection

Altisimo

QA web site

- *Problem:*
  - Nightly logs too large to receive as email
  - Other team members need view of state of quality
- *Opportunistic solution:* QA web site
- *Results:*
  - Easy to compare results with baseline
  - Mixed results on use by other team members

Altisimo



Altisimo

Manual testing

- *Problem:* Impractical to automate GUI testing
- *Opportunistic solution:*
  - Reuse leftover test script
  - Execute weekly
- *Results:*
  - Found most defects
  - Successful because all team members were able to participate

Altisimo

Source code compiler

- *Problem:*
  - Conformance to Java Language Specification
  - Compile time
  - Cross- platform portability of build scripts
- *Opportunistic solution:* jikes
- *Results:*
  - Identified few new defects by itself
  - But decreased maintenance

Altisimo

Improved delivery

- *Problem:* Ad hoc delivery system introduced defects
- *Opportunistic solution:*
  - Carefully written procedures
  - Automation
- *Results:* No defects injected by delivery

Altisimo

HTML validation

- *Problem:* Can't test generated HTML on every combination of browser, OS, and bill data
- *Opportunistic solution:*
  - Custom web spider
  - weblint
- *Results:* Discovered surprisingly few HTML defects

Altisimo

Opportunities not taken

- Automated GUI testing
- Code coverage analysis
- Develop new tests
- Adapt old tests to test framework
- Reorganize Java source code tree

Altisimo

Results

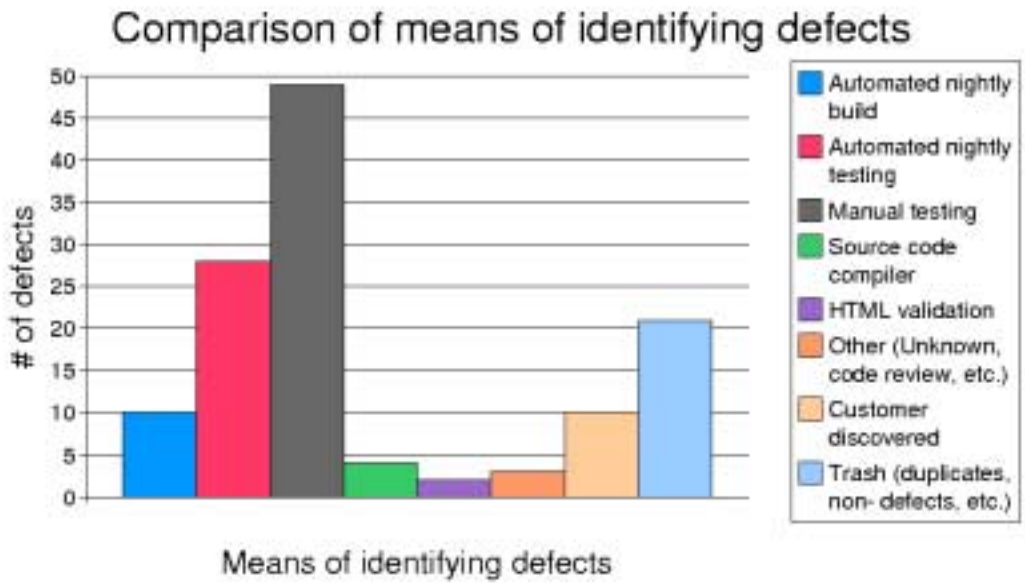
- Identified 106 unique new defects
  - 127 potential new defects identified
  - 21 duplicates, non- defect issues
- 10 post- delivery defects
- Manual testing most successful: 49 defects
- Automated nightly build & test next most successful: 38 defects

Altisimo

Results

Means of identifying defects	Number of defects	% of total	% of non-trash
Automated nightly build	10	7.9%	9.4%
Automated nightly testing	28	22.0%	26.4%
Manual testing	49	38.6%	46.2%
Source code compiler	4	3.1%	3.8%
HTML validation	2	1.6%	1.9%
Other (Unknown, code review, etc.)	3	2.4%	2.8%
Customer discovered	10	7.9%	9.4%
Trash (duplicates, non-defects, etc.)	21	16.5%	
Totals	**127**	**100.0%**	
Totals, non-trash	**106**	**83.5%**	**100.0%**

Altisimo



Altisimo

Conclusion

- Team delivered high quality product on time.
- Customer was satisfied.
- Opportunistic strategy worked.

Altisimo

Opportunistic Software Quality

Richard Kasperowski

Altisimo Computing
28 Regent Street
Cambridge, Massachusetts 02140
http://www.altisimo.com/

Email: kasper@altisimo.com

Abstract: *I recently acted as the sole quality advocate on a software project that had a rapid development schedule. As the project progressed, I realized that I was addressing the need for quick and productive quality assurance activities in a way that was similar to what I had done on other projects—I was identifying and acting on the best opportunities for quality improvement, iteratively improving the product's quality as I went. I call this strategy "opportunistic software quality."*

In this paper I present this strategy, using examples from the project described above. For each of the opportunities I identified, I present the number of software defects the opportunity helped me find. For this project and others on which I have worked, this strategy for improving software quality was successful.

Keywords: *Rapid software testing, techniques for quality assurance and quality improvement, short product development schedule, opportunistically improving software quality*

1. Introduction

I recently worked on a project for which I was the sole QA person and the ship date was only three months away. The goal of the project was to build and deliver a web−based billing system for the local telephone customers of a large telephone company. The software development team used HTML, JavaScript, and Java servlets to present bills, a relational database at the back end to hold customer and bill information, and Java and Enterprise Java Beans (EJB) in the middle for business logic.

The development team consisted mainly of highly skilled designers and programmers who had little experience with quality assurance. They were motivated to deliver a great product on time. My job was to help them do that, by inventing and executing a QA program for the project.

I was the only team member concerned primarily with product quality, and the delivery date was firm. I didn't have time to develop a well thought−out plan to assure the goodness of the product. Instead, I continuously looked for opportunities to improve product quality and took

---

Richard Kasperowski is president of Altisimo Computing, a software development consulting firm based in Cambridge, Massachusetts.

© 2000 Altisimo Computing Corporation

advantage of these opportunities. I later realized that I have done this on many projects over the years, and began to think of my capitalizing on quality improvement opportunities as a general strategy—"opportunistic software quality."

2. Opportunistic software quality

The strategy is to identify opportunities to improve software quality and to take advantage of these opportunities. To take advantage of the opportunities is to turn them into effective, persistent, reusable processes and procedures. Over the life of a project, one−time hacks are not as useful as repeatable, sustainable quality improvement activities. The value of this strategy lies in the improvement to the product gained by the repeated application of the opportunities.

In the following sections, I present examples of the opportunities we identified during this project and how they helped us deliver a high quality product on time.

2.1 Configuration management

Problem: Configuration management (CM) is an important part of any large project. Most CM systems share a few characteristics. They make it possible to track revisions of individual files, including who made the change, why the programmer made the change, and the content of the change. By branching source code into multiple trees, CM systems make it possible to maintain a production version of a software system on one branch while enhancing the system on an independent branch. CM systems also let you label a particular set of files and their revision numbers, making it easy to identify exactly which set of files corresponds to a particular release or bug fix.

Opportunistic solution: Prior to my joining, the team was already using CVS [SourceGear] as the project's configuration management system. They used CVS to track revisions to individual files and to label major releases using CVS's tag feature. I increased the use of CVS by creating branches for maintenance and for further development. I also began labeling each patch release, making it possible to identify the configuration that corresponded to the one the customer was using in production, thus making it easier to fix the defects the customer identified.

Results: Our overall experience with CVS was that it is a reasonable CM tool. It does a fine job tracking revisions to individual files and enabling the identification of sets of files and their revisions. Unlike some more advanced CM tools, CVS doesn't directly facilitate associating code changes with defect IDs; we worked around this deficiency by keeping a separate database of release labels and defect IDs.

CVS is a little different from other CM systems in that you don't have to exclusively lock a file in order to edit it. Because the files a programmer is working on aren't marked as "locked," programmers sometimes forget to check in all their changed files to the central repository; the result is that a programmer might fix a defect in his private "sandbox" but not fully propagate the fix to the central repository.

CVS isn't for programmers who need a graphical front−end on their CM system. Graphical user interfaces (GUIs) are available for CVS, but they do not fully support all the CVS features available on the command line. Using CVS via the command line is thus more powerful than using it via a GUI, but with command line use it can be easy to make mistakes.

2.2 Defect tracking system

Problem: A defect tracking system is another important tool on large projects. Without a defect tracking system, it is easy to forget to fix defects you know exist. Another problem is that one team member can be unaware that another team member is actively solving a problem; the two team members might duplicate each other's efforts. "Number of defects" is a basic quality metric, but it is difficult to measure without a defect tracking system. Finally, without a defect tracking system there is no specific history of what went wrong (the defects injected) while developing the system and how the team addressed the defects.

Opportunistic solution: Again before I joined, the team put JitterBug [Tridgell] in place as the project's bug tracking system. JitterBug is a free defect tracking system used on open source software projects such as `jikes` [IBM]. The team used JitterBug sporadically and didn't record all defects in it. I became the manager of the bug tracking system, making sure all defects were recorded and addressed. We didn't forget to fix any of the defects we needed to repair, so the product we delivered was better than it otherwise would have been.

Results: JitterBug is not as good as low−end commercial defect tracking systems, but it served our needs nonetheless. By the end of the project, we identified and recorded 177 defects in JitterBug, including defects identified during the first phase of the project, duplicates, and issues that turned out not to be defects. The customer perceived a high level of quality in the product we delivered; the perceived level of quality undoubtedly would have been lower if we had not been so careful about tracking and fixing defects.

2.3 Automated nightly builds

Problem: Infrequent source code builds are a quality problem. "Code rot" can occur: when old code is not compiled frequently, new or modified code can make the old code uncompilable. Broken builds can be difficult to repair if the code changes were introduced too long ago for the programmer to remember how he changed the code.

Opportunistic solution: An easy way to assure quality is to appoint a build master who makes sure he can build the code, with no compiler errors, at regular intervals. The interval we chose was one day. Programmers agreed that by the end of each day, any code changes they had checked in to the CVS repository would build correctly.

The best way to build the system regularly is to do it automatically. Before I joined the team, they used a set of build scripts to compile all the source code and build EJB components. The scripts were interactive, prompting the user for which components he wanted to build. As build master, I automated the build scripts, replacing interactive prompts with command line arguments. I added a wrapper script that would backup the previous instance of the source tree, check out the latest source code, and build the whole system. I used `cron` to run the wrapper script automatically each weekday night at 2:00 AM. The wrapper script emailed a log of the build to me; each morning I reviewed the log for build errors, isolated the source of each error, and asked the programmer who injected the error to fix it. This system was known as the "nightly build."

Results: We found a number of defects with this method. The usual problem was that the programmer didn't check in all of his changes, so some source files in the repository were inconsistent with each other. Because we found the defects within a day of the code change, it was easier for the programmers to fix the defects than it would have been if we hadn't built the

code regularly. In addition, builds usually succeeded when they were needed most, such as for an emergency patch release.

2.4 Automated nightly testing

Problem: The programmers had built a number of semi−automated tests for particular sections of code. We wanted to run the tests regularly to help us identify newly introduced defects, but there were too many tests to be able to run them manually.

Opportunistic solution: I initially attempted to automate the existing tests by adapting them to the Java Test Driver [Kasperowski]. Because the tests were not designed for that kind of test framework, however, it was time consuming, and I soon realized that I wouldn't be able to finish porting the tests before the ship date. Instead, I built a wrapper script to simply execute the existing tests in sequence. I added this script to the nightly build, which became the "nightly build−and−test."

To analyze the results of each nightly build−and−test, I built a means of automatically comparing the current results log with the results log of any other run of the nightly build−and−test. My first attempt to do this was to use `diff` for a simple file comparison. There were two problems with this approach: (1) the application's running log contained time stamps, which were expected to be different every day, and (2) the application's running log contained verbose Java garbage collector messages, which by their nature are nondeterministic. I minimized (but did not completely solve) this problem by augmenting the simple `diff` with a number of `sed` filters to produce a reasonable build−and−test log comparison tool.

Over time, I added tests of other important sections of code, using the Java Test Driver as the test framework for the new tests.

Results: Automated nightly testing helped us find new defects within a day of a broken code change, making it easier for the programmers to fix the defects than it would have been if we had not built the code regularly.

2.5 QA web site

Problem: The nightly build−and−test logs were each over one megabyte in size; I grew tired of receiving these huge logs as email attachments. I also didn't want to have to remember the long command line required for comparing two sets of nightly build−and−test logs. Finally, I was the only team member with a view of the results of the nightly build−and−test; I wanted the whole team to have a view of the quality of each night's build−and−test.

Opportunistic solution: I built a QA web site for the team, using the Apache web server [Apache], the Apache JServ servlet engine [JServ], and Java servlets [Davidson]. The web site, shown in Figure 1, consisted of the most recent automated test results, a way to compare any set of test results with any other set of test results, hyperlinks to web pages that could be used for manual testing, and hyperlinks to written procedures.

Results: Using servlets to build a web site was relatively easy, but it was difficult to learn how to configure JServ. (My motivation for implementing this web site with servlets was that the project used servlets in its implementation, and I wanted to understand how they worked.) The web site made it very easy for me to review nightly build−and−test results.

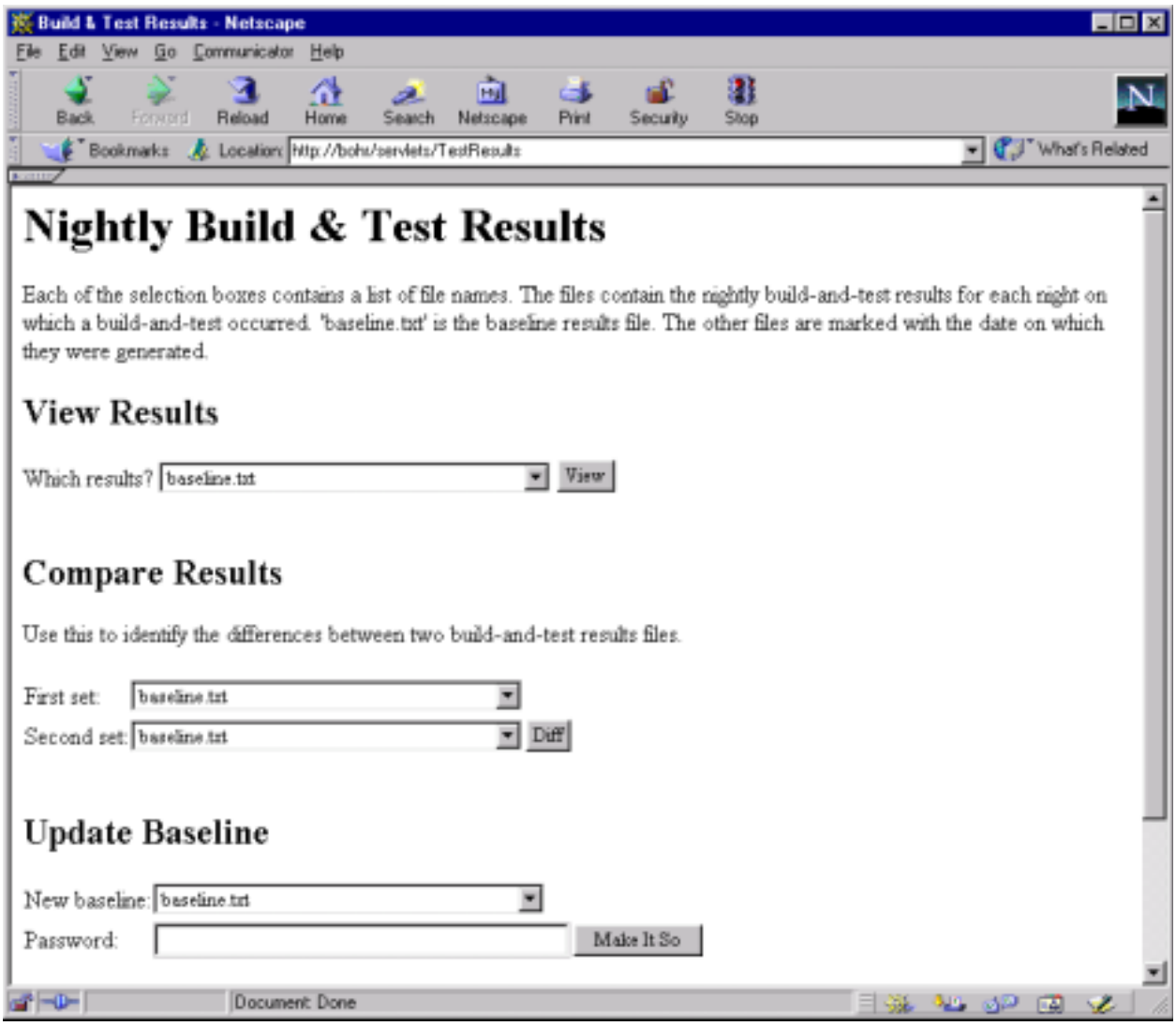


Figure 1: The nightly build−and−test web site

The test results web site grew into a more general project web site, with hyperlinks to sample test bills and deployment procedures. The hyperlinks to web pages that could be used for manual testing turned out to be extremely useful for the other team members; they vociferously complained whenever the site was down.

Unfortunately, few team members used the test results part of the web site. They were more interested completing the implementation than in viewing the day−to−day results of the nightly build−and−test.

2.6 Manual testing

Problem: We wanted to evaluate the quality of the GUI regularly, but it wasn't practical to automate GUI testing. One reason was that the team had not acquired automatic GUI testing tools and it would have taken too long to purchase one. Another reason was that the GUI design was not frozen, so it would have been a poor use of our time to develop and maintain automated GUI tests.

Opportunistic solution: A written script for testing the GUI was left over from the first phase of

the project, and I adapted it for use with the current version of the product. I made it a policy that the script had to be executed at least once per week, and I rigorously followed the policy.

Results: With this regular planned testing, augmented by ad hoc testing from other team members, we identified many defects. In fact, manual GUI testing identified more defects than any other part of the quality program. The success of the manual GUI testing effort probably stems from the fact that all team members participated; the other quality initiatives were executed by me alone.

2.7 Source code compiler

Problem: Different Java compilers have different characteristics. One important characteristic is whether the source code they accept conforms to the Java Language Specification [Gosling]. If a compiler accepts non−conforming source code, the compiled code's behavior might be unpredictable. In addition, non−conforming source code is more time−consuming to maintain than conforming source code.

Another important characteristic of a Java compiler is the amount of time it takes to compile the source tree. Programmers are unlikely to use the project's build scripts to test whether their code is buildable if it takes too long to execute a whole−system build.

A third characteristic of a Java compiler is its cross−platform portability. The usual Java compiler, Sun's `javac` [Sun], is portable in that it is written in Java, but the format of the path names in its command line parameters varies depending on whether it is running on Windows or UNIX. Because of this, we were maintaining two sets of build scripts, one for Windows and one for UNIX. As the build scripts evolved, it was difficult to keep the Windows−specific build scripts synchronized with the UNIX−specific ones. Programmers sometimes complained that the Windows−specific build scripts were broken; the scripts were indeed broken, because we often forgot to update them to match the behavior of the UNIX−specific build scripts.

Opportunistic solution: One of the team members suggested that we use `jikes` [IBM] as our Java compiler. `jikes` rejects non−conforming source code. `jikes` is written in C++ and is compiled into a platform−native executable, so its execution speed is faster than that of `javac`. `jikes` accepts UNIX−style path names (that is, paths with forward slashes) on both Windows and UNIX. I modified the nightly build−and−test scripts to use `jikes` by default.

Results: The amount of time it took to build the whole system decreased from 63 seconds to 16 seconds, an improvement of nearly four times. (Times were measured on a Sun Enterprise 250 with 2 CPUs, 512M bytes of RAM, and 27G bytes of total disk space.)

Because `jikes` accepts UNIX−style path names on both Windows and UNIX, I was able to retire the Windows−specific build scripts. I no longer had to maintain two sets of build scripts, struggling to keep them synchronized. Programmers were more likely to use the build scripts because the single set of scripts was more likely to succeed.

We identified a few instances of non−conforming Java source code. Two kinds of errors were typical: unreachable code and uncatchable exceptions.

2.8 Improved delivery

Problem: Delivering the product to the customer and installing it on the customer's machine was an ad hoc procedure. It was difficult to repeat successful instances of deployment. The

installation guide was too difficult to follow, so the person installing the system usually made mistakes. It was time consuming to debug each unsuccessful deployment.

Opportunistic solution: I developed a set of scripts to automatically build and install the product, as well as a procedure for making the delivery; this drastically reduced the number of deployment mistakes. I began by carefully recording the steps I followed to build, deliver, and install the system. I used this list of steps as the de facto procedure for deployment, refining it every time I deployed the system. When I was comfortable deploying the system manually, I built a script to do the work for me. Thereafter, the deployment procedure consisted of my following a few manual steps and running the script.

Results: Deployment was almost always successful. There were no defects injected by poor deployment.

2.9 HTML validation

Problem: The product is web−based, but there weren't enough resources to be able to test the GUI on every combination of web browser and operating system. How could we be confident that the application would generate valid web pages for arbitrary combinations of web browser and operating system, and for arbitrary customer billing information?

Opportunistic solution: On a previous project, I discovered a tool called `weblint` [Bowers] to automatically check the validity of the HTML code that the product delivered to users' web browsers. `weblint` validates HTML files against the World Wide Web Consortium's HTML 3.2 standard [W3C]. `weblint` also has parameters to validate the HTML extensions accepted by Microsoft's Internet Explorer and Netscape's Navigator browsers.

The billing application generated web pages dynamically based on the user's monthly account information and on what kind of page he requested. `weblint` can only validate static HTML files, though. One of the team members had built a simple web spider that visited parts of a customer's bill to ensure they exist. I enhanced the spider to visit every page of a customer's bill, saving the dynamically generated HTML files to disk. I created a large number of representative bills, saved their HTML to disk, and ran `weblint` on the disk files to validate the generated HTML. I added the execution of the web spider and `weblint` to the nightly build−and−test.

Results: The generated HTML was surprisingly good—we identified only a small number of defects this way. I write "surprisingly good" based on my experience with other web testing projects, where the HTML was largely non−conforming with respect to the standard.

When I ran `weblint` in Netscape−compatibility mode, it identified many instances of non−compliant HTML code. These were not considered defects because the web pages were designed to use Internet Explorer HTML extensions.

2.10 Opportunities not taken

There were a few techniques we considered using, but did not use. They were good ideas and might have identified defects, but we did not have the resources to execute them.

One of our ideas was to automate GUI testing. Unfortunately, we didn't already have a GUI testing tool in our possession, and we probably didn't have enough time to develop and maintain automated GUI tests anyway. In addition, the GUI was in a state of flux, so adjusting the tests to

match a given day's instance of the GUI would have been costly. "The cost of automating a test is best measured by the number of manual tests it prevents you from running and the bugs it will therefore cause you to miss;"[Marick] by this measure, automating the GUI testing would not have been a good investment.

We considered performing code coverage analysis to help determine the goodness of our existing tests and to guide new test development. We went so far as to evaluate tools, select one, and place an order for it. In this case, big−company bureaucracy impeded us. Two months after my placing the order, the tool had not arrived; the project was nearing its end date, and we would not have enough time left to use the tool effectively. In retrospect, this delay might have been a good thing: we had plenty of other development work to do, and the use of the code coverage tool might have interfered with other quality assurance activities.

We did not build many new tests. I began building a tool that would identify dependencies between Java classes. I wanted to test the classes exhaustively, in order from those with the least number of dependencies to those with the greatest number of dependencies, using a levelization technique similar to the one described in [Lakos]. However, while building this tool, I would not have been identifying defects, so I dropped the idea. We were fairly comfortable with the existing code anyway, despite that it had not been extensively tested in the lab. Much of the code was already running in production and working well in that environment.

The semi−automated tests that existed before I joined the team exercised large sections of code and sent the results to stdout. I wanted to modify these tests so they would exercise smaller sections of code, to make it easier to identify the source of a defect when one was found. I also wanted the tests to be able to determine automatically whether their results matched the expected results, and report that information to the human tester. I began retrofitting the existing tests to work within the Java Test Driver framework. This proved to be too time consuming; I estimated that I would have spent all my time on this activity and nothing else.

Finally, we wanted to reorganize the Java source code tree. The existing source code tree did not follow the Java convention. The convention is that the source code tree is a tree of directories whose names match Java package names. The .java files that declare themselves to be in a given package go in the corresponding directory. Instead of following this convention, the files in the source code tree were grouped by module name, which was independent of the Java package names. This is a problem because Java compilers and other development tools do not work efficiently, if at all, unless the .java files are in a conventional Java source tree. It also makes maintenance difficult because experienced Java programmers have difficulty finding source code files if they are not in the conventional directory tree. I built a tool that takes an arbitrary Java source tree as its input and constructs a conventional Java source tree. The tool appeared to work, but we were afraid of the potential instability that reorganizing the source tree might introduce. We postponed this activity until the next phase of the project.

3. Results

We identified 127 potential defects during this phase of the project. Of these, 21 were duplicates or were issues that were not really defects, leaving 106 unique defects. The customer discovered at most 10 of these defects, or 9% of the total number of known defects. In fact, the customer discovered fewer than 10 defects: 10 is the sum of the number of defects identified by the customer and the number of defects we identified while investigating those defects. Although we did not establish a specific target at the beginning of the project, our delivering 10 defects to

the customer indicates that we were successful in delivering a high quality product.

The two best techniques for identifying defects were manual testing and automated nightly testing. Manual testing was responsible for finding 49 defects, or 46% of known defects, and automated nightly testing helped us find 28 defects, or 26% of known defects. The combination of automated nightly build and automated nightly testing together identified 38 defects, or 36% of known defects.

Table 1 and Figure 2 show the number of defects identified by each defect identification activity.

We delivered a relatively high quality product on time. The customer accepted the delivery and found very few defects. The opportunistic quality improvement strategy worked.

Means of identifying defects	Number of defects	% of total	% of non−trash
Automated nightly build	10	7.9%	9.4%
Automated nightly testing	28	22.0%	26.4%
Manual testing	49	38.6%	46.2%
Source code compiler	4	3.1%	3.8%
HTML validation	2	1.6%	1.9%
Other (Unknown, code review, etc.)	3	2.4%	2.8%
Customer discovered	10	7.9%	9.4%
Trash (duplicates, non−defects, etc.)	21	16.5%	
Totals	**127**	**100.0%**	
Totals, non−trash	**106**	**83.5%**	**100.0%**

Table 1: The usefulness of each means of identifying defects is indicated by the number of defects identified by each means. "Manual testing" includes both GUI and non−GUI manual testing. "Other" includes all ad hoc means. "Customer discovered" includes defects identified by the customer, as well as defects we identified while investigating those defects. "Trash" includes duplicates and issues that were not truly defects.

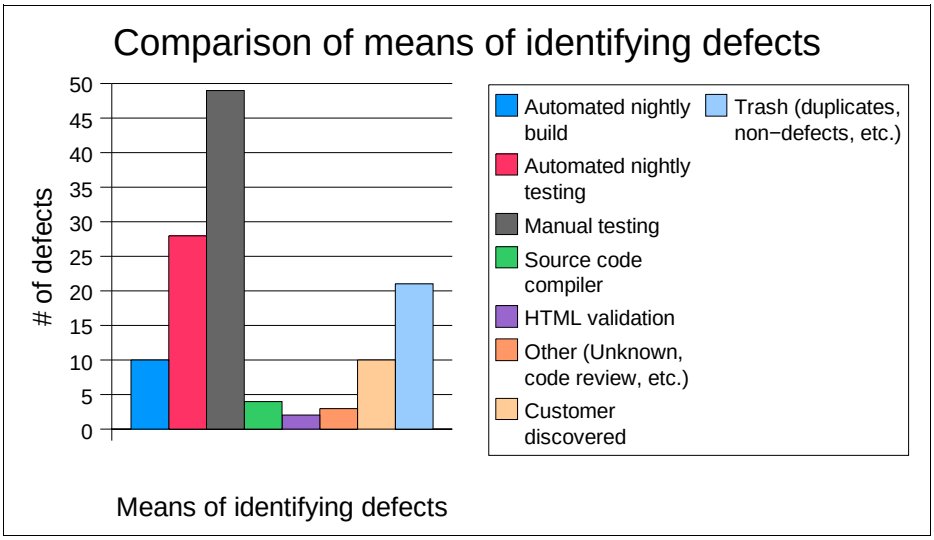


Figure 2: The relative utility of each means of identifying defects. Manual testing is clearly a productive means of identifying defects. The combination of automated nightly build and automated nightly testing is close behind.

4. References

[Apache] The Apache Software Foundation. The Apache HTTP server project. "`http://www.apache.org/httpd.html`".

[Bowers] Bowers, N. Weblint. "`http://www.weblint.org/`".

[Davidson] Davidson, J., and Ahmed, S. Java servlet API specification, version 2.1a. Sun Microsystems, Inc., 1998.

[Gosling] Gosling, J., Joy, B., and Steele, G. *The Java Language Specification*, edition 1.0. Sun Microsystems, Inc., 1996. "`http://java.sun.com/docs/books/jls/html/`".

[IBM] IBM. Jikes project. "`http://oss.software.ibm.com/developerworks/opensource/jikes/project/`".

[JServ] The Apache Software Foundation. The Apache JServ project. "`http://java.apache.org/jserv/`".

[Kasperowski] Kasperowski, R. The design and implementation of a Java test driver. In *Proceedings of the 16th International Conference and Exposition on Testing Computer Software (*Washington, D.C.), 1999; "`http://www.altisimo.com/research/design−implement−test−driver.html`".

[Lakos] Lakos, J. *Large−scale C++ Software Design*, Addison−Wesley, 1996, 203−324.

[Marick] Marick, B. When should a test be automated? In *Conference Proceedings: Eleventh International Software Quality Week* (San Francisco), May, 1998.

[SourceGear] SourceGear Corporation. CVS. "`http://www.sourcegear.com/CVS/`".

[Sun] Sun Microsystems, Inc. Java 2 SDK tools. "`http://java.sun.com/products/jdk/1.2/docs/tooldocs/tools.html`".

[Tridgell] Tridgell, A. JitterBug. "`http://samba.anu.edu.au/jitterbug/`".

[W3C] World Wide Web Consortium. HTML 3.2 Reference Specification. Jan., 1997. "`http://www.w3.org/TR/REC−html32.html`".

QW2000 Keynote 10P1

Mr. Marcelo Dalceggio
(Banco Rio de la Plata SA)

Automated Software Inspection Process

QWE'99 Best Presentation Award

BACK TO QW2000 PROGRAM

Presentation Abstract

Due to the huge effort that inspection process demands, we developed an automated inspection process based on our experience with Y2K code control in order to increase the efficiency of the inspection process.

Our systems have more than 10 million LOC and we produce/modify 900 programs per month and these programs have to be compliant with more than 150 rules established with different standards. It's almost impossible to inspect manually that volume of code and to pay attention to the rules during inspection.

Our inspection process has four basic steps: selection of the inspection candidates, identification of the attributes of each component, inspection execution and reporting, defect removal tracking and verification.

We built a compliance policy and rules database based on different sources: code standards, failure tracking lessons learned, technical books suggestions, prohibited commands, software reuse and we developed a system that reviews source code and matches rules with statements producing an inspection report.

About the Author

Marcelo R. Dalceggio is currently chief of the Quality Assurance Department at Banco Río de la Plata S.A. in Argentina. In this capacity, he is responsible for the development and implementation of defect prevention, improvement opportunities and quality assurance activities. Prior to this position he was the chief of the Development Department at DAZ Consulting and was responsible for software development.

He received his undergraduate degree from the Universidad Tecnológica Nacional and his postgraduate degree from the Universidad CAECE in Argentina. He has been working at Universidad Tecnológica Nacional since 1991 as a Professor of Software Engineering Fundamentals.

Contact: mdalceggio@intranet.bancorio.com.ar

BACK TO QW2000 PROGRAM

Automated Code Inspection

A practical approach to improve code inspection efficiency

Banco Río de la Plata work experience

Marcelo Dalceggio
Alvaro Ruiz de Mendarozqueta

June 2000 ~ Argentina

Grupo Santander

BANCO RIO

Presentation Overview

- Who we are
- Y2K Experience
- Manual Code Inspection
- How the idea originated
- Automated Code Inspection Implementation Process
- Automated Code Inspection Execution Process
- Where it fits in project life cycle
- Conclusions & Achievements

Grupo Santander

BANCO RIO

Who we are

- Banco Río de la Plata is one of the major private banks of Argentina and belongs to Banco Santander Central Hispano
- 250 branches executing one million transactions a day and providing universal banking services
- IBM mainframe based core systems
- COBOL & Assembler, CICS, VSAM, DB2
- The goal: Reduction of costs in the software development and maintenance

Grupo Santander 3 BANCO RIO

Y2K Experience

- Y2K Contamination Control
  - Windowing technique solution
  - Code revision to detect possible violations to the Y2K fix
  - Few Y2K Compliance rules to be applied in all the programs
  - SW-assisted inspection process
  - Very satisfactory outcomes

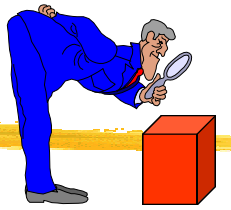
Grupo Santander 4 BANCO RIO

Code Inspection

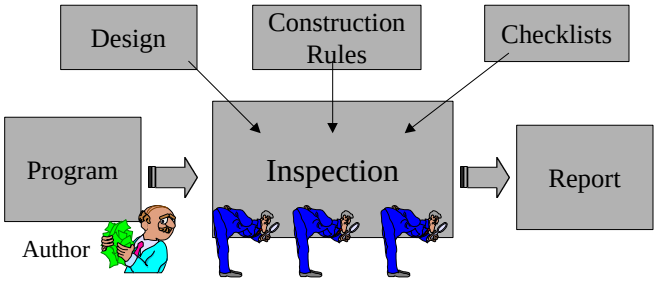
- There's no doubt of its advantages
  - Defects are detected before the testing process, saving time and money
  - Identifies defects that testing misses
  - Gives complete code coverage
  - Finds root cause defects (no symptoms)
  - Additional advantages
    - Improvement opportunities
    - Good & Bad practices are finally learned

Grupo Santander 5 BANCO RIO

Manual Code Inspection

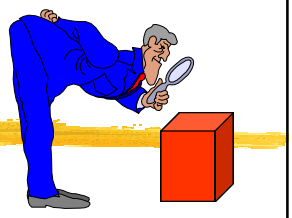


- Code Inspection formal outline

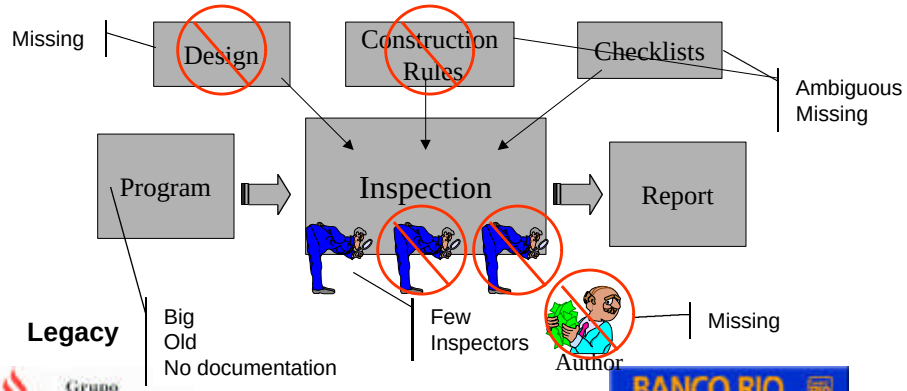


Grupo Santander 6 BANCO RIO

Manual Code Inspection



- Most common scenario

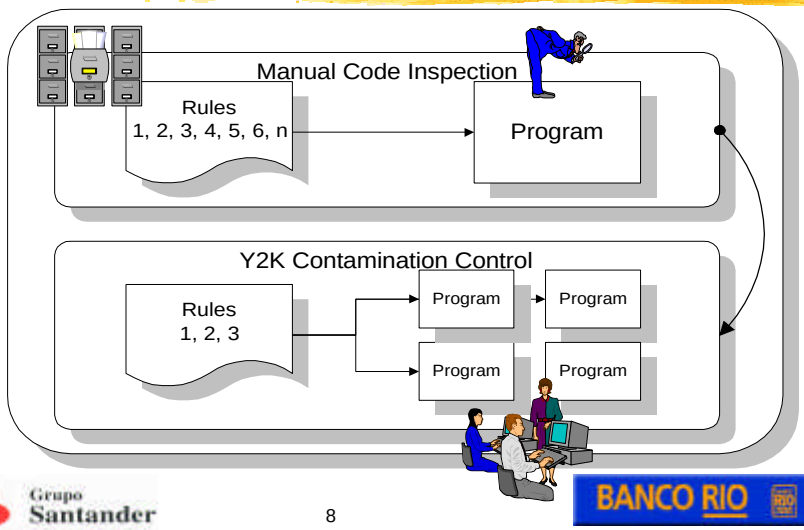


Grupo Santander

7

BANCO RIO

How the idea originated

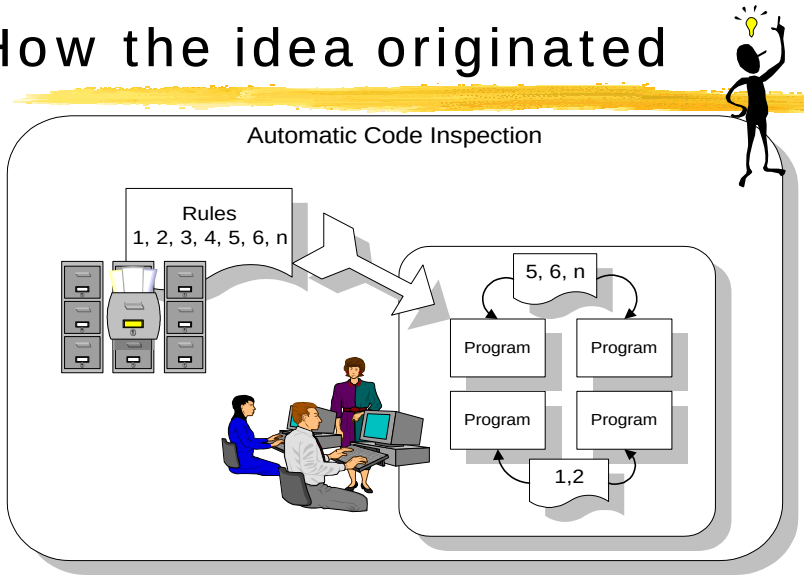


Grupo Santander

8

BANCO RIO

How the idea originated



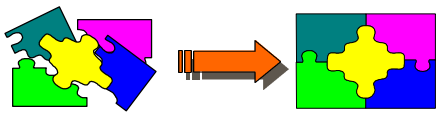
Grupo Santander

9

BANCO RIO

Automated Code Inspection Implementation Process

- Process applied
  - Source rules identification
  - Rules development & construction
  - Violated rules detection
  - Implementation decision



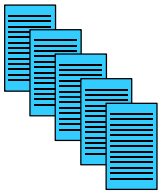
Grupo Santander

10

BANCO RIO

Automated Code Inspection Implementation Process

- Source rules identification (I)
  - Code Standards
  - SCM
    - Library of Reusable Modules
    - Existing /Discontinued Modules
  - Good practices
  - Failure tracking system
  - Classic Code Inspections (Fagan style)
  - Development area 'feedback'



Grupo Santander 11 BANCO RIO

Automated Code Inspection Implementation Process

- Rules development & construction (II)
  - Identify rules from each source
  - Identify application domain from each rule
  - Avoid ambiguities
    - "... Cobol paragraphs should not be long ..."
  - Validate rules with those in charge
    - Sponsor's support is very important
  - Record rules in the Compliance DB

Grupo Santander 12 BANCO RIO

Automated Code Inspection Implementation Process

- Rules development & construction (II)
  - Rules could have different formats:
    - Prohibited commands
      - Alter, Go to, ……
    - Data handling
      - Lack of return code checks
    - Exception handling
      - *When Other* clause in *Evaluate* statement
    - Configuration Management
      - Module no longer supported

Grupo Santander

13

BANCO RIO

Automated Code Inspection Implementation Process

- Rules development & construction (II)
  - Rules could have different formats: (cont.)
    - Performance
      - Avoid *Join* 4 or more tables
    - Cosmetic
      - How to indent certain commands
    - Naming Convention
      - Minimal length of characters in variable names
    - Module Size
      - Maximum number of statements it should contain

Grupo Santander

14

BANCO RIO

Automated Code Inspection Implementation Process

- Rules development & construction (II)
  - Considerations about rules
    - A rule violation doesn't mean avoiding a defect
      - It makes code more susceptible to the introduction of new defects
  - The code should be
    - Correct
    - Understandable
    - Efficient
    - Maintainable

Grupo Santander

15

BANCO RIO

Automated Code Inspection Implementation Process

- Violated rules detection (III)
  - Discover different ways in which rules could be violated within the code

- Implementation decision (IV)
  - Evaluate detection method
    - Scanning, parsing, control flow analysis, etc..
  - Define automated/assisted detection
  - Tool: Buy or build ?

Grupo Santander

16

BANCO RIO

Automated Code Inspection Implementation Process

- Implementation decision (IV)
  - Evaluate detection method
    - Scanning
      - Perform string searches looking for specific patterns
      - OK for 'prohibited commands'
    - Parsing
      - Understand the syntax of the language
      - OK for 'exception handling'
    - Control Flow Analysis
      - Follow the implications of each control statement
      - OK for 'dead code'

Grupo Santander 17 BANCO RIO

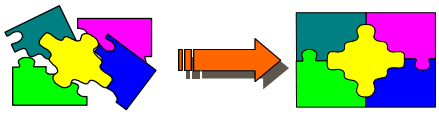
Automated Code Inspection Implementation Process

- Implementation decision (IV)
  - Define automated/assisted detection
    - Automated
      - A defect is identified without needing any later human analysis
    - Assisted
      - Code scanned by defined wild card search criteria to find potential defect candidates
      - Final decision through manual intervention
        - Environment information /Author's information
    - *Not everything can be automated !!*

Grupo Santander 18 BANCO RIO

Automated Code Inspection Execution Process

- Process applied
  - Candidate programs identification
  - Inspection execution
  - Defect reporting
  - Tracking



Grupo Santander

19

BANCO RIO

Automated Code Inspection Execution Process

- Candidate programs identification (I)
  - Criteria definition
    - New & Modified
    - Belonging to a certain domain
    - After or Before a given deadline
    - Reported as fixed
    - On demand
    - At random

Grupo Santander

20

BANCO RIO

Automated Code Inspection Execution Process

- Inspection execution (II)
  - Program domain checking
  - Rules compliance verification to each identified domain
  - Results review
  - Record defects in the defect tracking DB

Grupo Santander

21

BANCO RIO

Automated Code Inspection Execution Process

- Defect reporting (III)
  - Inform the ones in charge
  - Record the report delivery

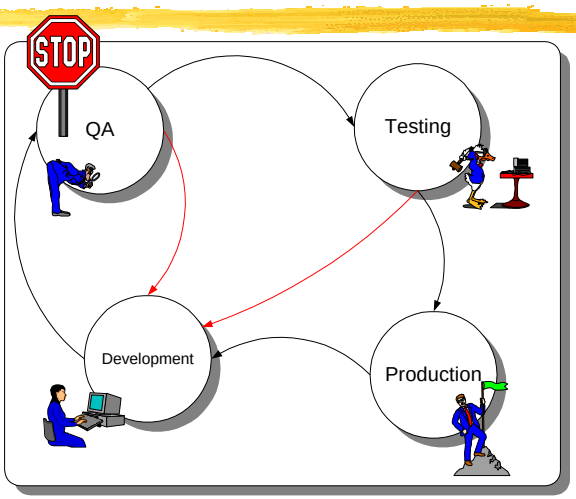
- Tracking (IV)
  - Verify the ones solved
  - Request unsolved defects
  - Metrics collection

Grupo Santander

22

BANCO RIO

Where it fits in project life cycle



Grupo Santander

23

BANCO RIO

Conclusions

- This kind of Inspection...
  - is a good alternative in today's schedule-driven projects
  - is more scalable than manual inspections
    - More rules and more programs
    - Few resources
    - Adding rules do not affect inspection efficiency
  - does not require 'special' skills
  - **does not** replace 'Fagan Style' inspections

Grupo Santander

24

BANCO RIO

Conclusions

- Once a new type of defect has been detected it is possible to...
  - prevent it from happening again, and
  - get rid of it from the existing programs
- Inspection ensures fewer trivial defects will delay the application testing
- Code quality & reliability metrics could be defined

Grupo Santander

25

BANCO RIO

Achievements

- 189 rules identified
- 820 KLOC per month with 3 QA reviewers
- 1.23 defects detected per KLOC
- Reporting 'on demand' for different critical projects
- Reporting 'on demand' for third party developments

Grupo Santander

26

BANCO RIO

Contact

- Marcelo Dalceggio
  - mdalceggio@intranet.bancorio.com.ar
  - phone (54) 11 4341-3515
  - fax (54) 11 4341-3480
- Alvaro Ruiz de Mendarozqueta
  - aruizdemendarozqueta@intranet.bancorio.com.ar
  - phone (54) 11 4341-3514
  - fax (54) 11 4341-3480

Grupo Santander

27

BANCO RIO

Automated Code Inspection

A practical approach to improve code inspection efficiency

Banco Río de la Plata work experience

Marcelo Dalceggio	mdalceggio@intranet.bancorio.com.ar
Alvaro Ruiz de Mendarozqueta	aruizdemendarozqueta@intranet.bancorio.com.ar
Banco Río de la Plata S.A.	Grupo Banco Santander Central Hispano

Executive Summary

Banco Río de la Plata is one of the major private banks of Argentina, with 250 branches executing one million transactions a day and providing universal banking services to local, regional, and international customers.

In 1997 it faced the challenging Y2K problem. It contracted different service providers to assess and remediate the code using automated tools. Some applications were replaced and others were solved in-house by manual process. The code was fixed using the 'windowing' technique (This solution carries a long term risk. 'Windowing' logic changes program code but not data). It performed Y2K baseline (regression) testing and post-Y2K testing. More than 7.000 kloc of Cobol and Assembler code were reviewed and a lot of new code was added to the legacy system making it vulnerable to the introduction of new defects, because those changes made the code larger and more complex. In this context, QA staff had the mission to provide a process that would let the Bank ensure that maintenance activities (changes and enhancements) wouldn't infect the programs. This process was called *Y2K Contamination Control*.

It was almost impossible to perform this activity manually. A tool-based inspection process was developed in order to achieve the goal. Good results encouraged QA staff to extend it to address other common defects (not only correct date processing) such as code standards & good coding practices violations.

This paper summarizes this process and how it was enhanced to become part of the ongoing software quality process, increasing productivity and reducing costs.

How the idea originated

Code Inspections advantages are very well known but they are very hard to implement, cost a lot, and critical projects not always have the opportunity to apply them.

During Y2K project we developed a 'Process' in order to control code contamination within the maintenance activities

Both experiences triggered the original idea

The Company

Grupo Santander has been for years the leader financial group in Spain and with the last merger with Banco Central Hispano has become the new *Grupo BSCH*, one of the biggest financial groups in the unified Europe today.

In Latin America, it has the major foreign bank commercial net with banks in many countries of the region. Two years ago, Grupo Santander acquired *Banco Río de la Plata* in Argentina, one of the major private banks in the country, which with its 250 branches executing one million transactions a day provides universal banking services to local, regional, and international customers.

Since its acquisition, Banco Río de la Plata has been continuously creating new and sophisticated financial products, demanding legacy systems transformations while increasing productivity and reducing costs to maintain a competitive edge and operational excellence in today's business world. Best software engineering practices are helping the organization to achieve these goals.

Y2K Experience

Y2K Project

The Y2K Conversion Project applied windowing technique in order to solve the two-digit data for the year. This technique reduced costs and allowed an easier implementation but one of its major problems is the risk associated with maintenance. During the modification

activities the modified code with windowing technique could easily be changed undoing the Y2K fix.

With the goal of reducing those risks we developed what we call 'Y2K Contamination Control Process'

Contamination Control

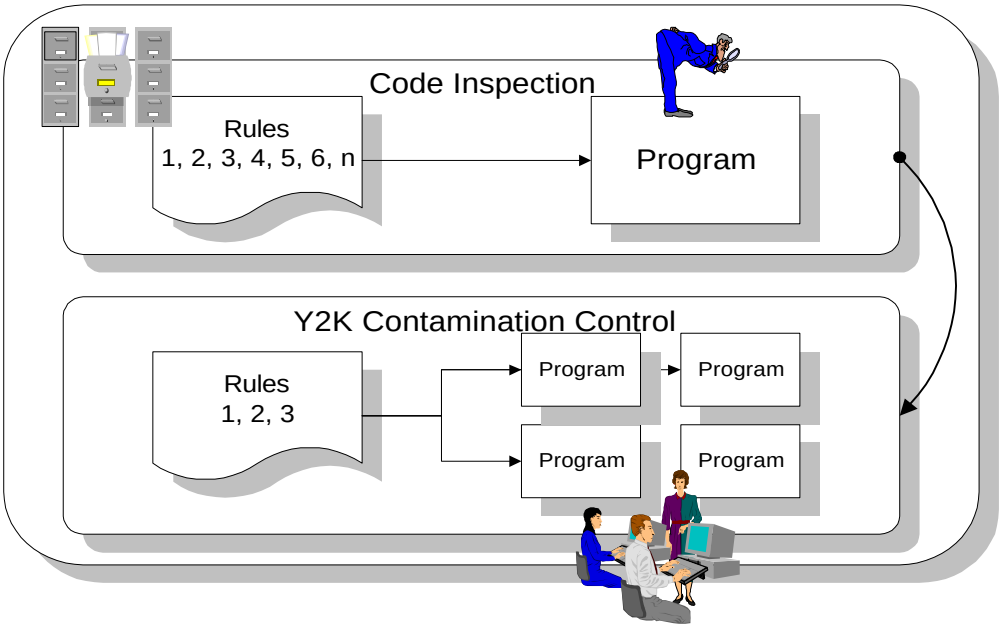
We established code revisions to detect possible violations to the Y2K fix during maintenance activities. We did a survey looking for different ways of introducing a defect within the code or certain situations that might be a risk for future modifications.

These few Y2K compliance rules had to be applied in all the programs, not only at the end of Y2K conversion project, but also during the everyday maintenance work. This was the real challenge and for that volume of work it was impossible to apply code inspections in the 'Fagan style'. Manual code inspections were unfeasible.

We developed a software-assisted inspection process with a very simple tool. We parsed the programs with it and in every situation of a probable 'fix violation', a QA staff analyzed the very nature of the problem.

We obtained very satisfactory outcomes and the results presented the following characteristics:

- Every system was reviewed and the results were reported to the managers.
- A weekly revision is being performed for all the new and modified programs.
- The final costs are cheaper than those of the consultant firms running in the market.
- Some findings were checked with manual code inspections and with consultant firms. The results were optimal.
- The contamination control process has to continue as long as windowing fixes remain within the code.
- The following picture shows how we moved from the manual code inspections (many rules in one program) into a software-assisted contamination control (few and stable rules in all the programs).



Some problems with the manual code inspections

Code Inspections advantages are very well known but very hard to implement, cost a lot, and critical projects not always have the opportunity to apply them.

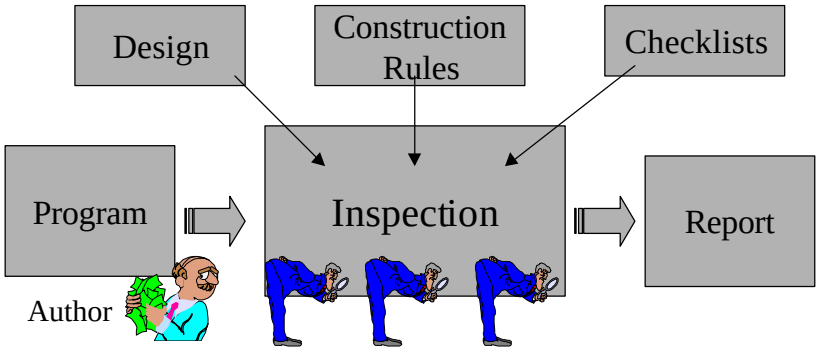
These are the main advantages:

- Defects are detected before the testing process, saving time and money.
- Identifies defects that testing misses.
- Gives complete code coverage.
- Finds root cause defects (no symptoms).
- Additional advantages: Improvement opportunities, Good & Bad practices are finally learned.

Other difficulties with personnel

- Programmers are not used to sharing code, neither during inspections, nor during coding activities.
- Programming as an artistic activity prevails.
- Managers encourage heroic programmers and schedule-driven results.
- Standards are not usually used. Programs are very big and count on no documentation.

Difficulties in Manual Code Inspections

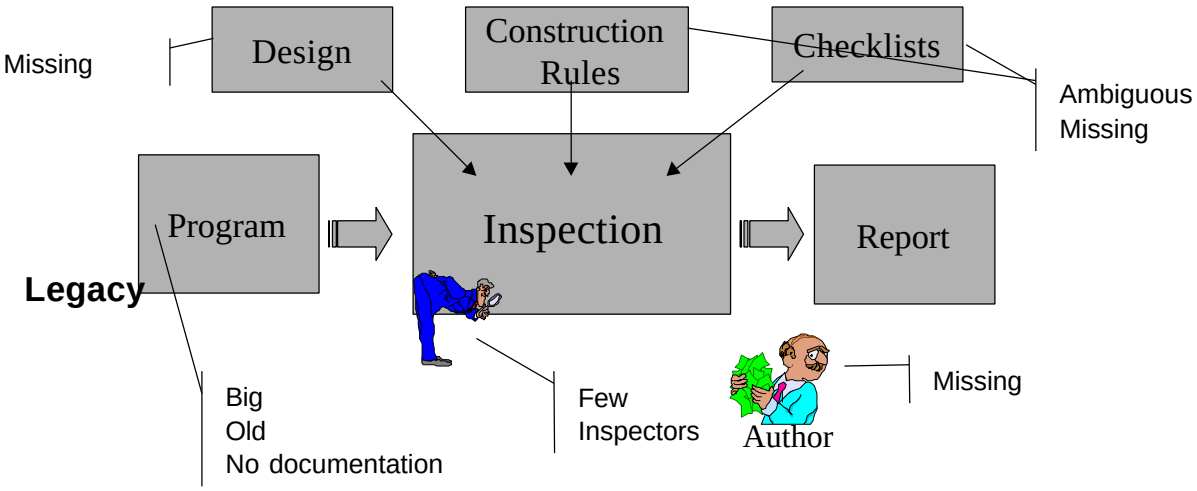


In manual code inspections the well-known authors recommend the following components:

the program that will be the object of the inspection, the design document that originated the program, the construction rules taken into account during the programming activities, the checklists for the inspection, the author's program and the inspectors.

In our environment we have very old programs. It is very hard to find the design documents. The author may no longer belong to the bank or may be doing other activities rather than programming. (He might not even remember the program structure). It is also very difficult to trace the rules applied during programming. Some of them depend on projects or individual criteria. Checklists include a lot of ambiguity.

Inspectors candidates are assigned to critical projects. They have demanded skills which makes it impossible to assign them to inspections.



Difficulties related to our work environment

Our System Department creates or modifies and average of 900 programs per month. Each program has an average size of 1000 lines of code (LOC).

The monthly demand for inspections is:

900 pgms * 1KLOC/pgm = 900.000 LOC monthly

Taking into account that industry considers an average inspection rate of 100 lines of code reviewed per hour, the total amount of hours needed in our case is:

900.000 LOC / 100 LOC/hour = 9.000 hours

Assuming that we would like to inspect the total amount of programs we need:

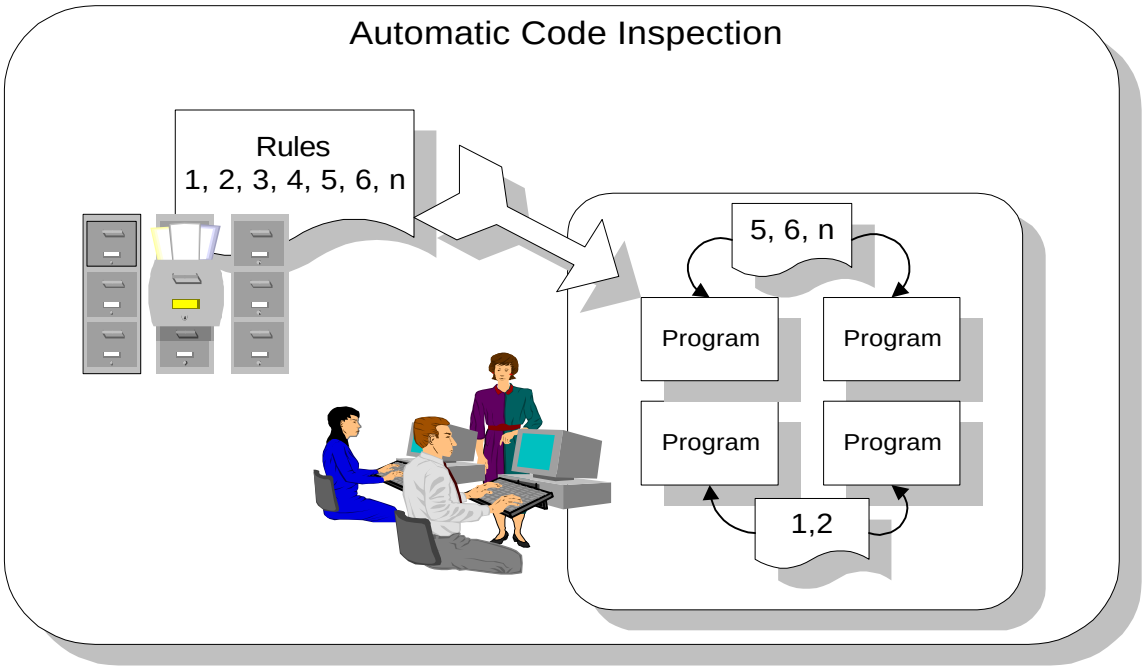
9.000 hours / 160 hours/staff = 56 staff

This clearly shows it is economical unfeasible to inspect all the programs. Assuming only 10 percent of the total amount would demand 6 staff per month leaving the 90 percent without revisions.

The idea

With the aim of inspecting the total number of programs, we combined our experience with the automated part of 'Y2K Contamination Control Process', with the one from 'Manual Code Inspections'.

We merged both ideas and wondered if we could check the total number of rules known in all the programs.



We developed what we called **'Automated Code Inspection Process'** .

Automated Code Inspection Implementation Process

The process followed to get ready for the inspection.

It doesn't matter the way you are going to inspect the code, the most important thing is to recognize which elements make your code vulnerable to the introduction of defects.

Depending on the level of accuracy you expect from the automation is the level of sophistication you need from the tool you buy or build.

Source rules identification

The first task to fulfill is the search of all the sources from where you can obtain the rules to be controlled.

Some of the sources identified are the following:

- Code Standards: They're the most important sources. Language std. (Cobol), transactional monitor std. (Cics), database std. (DB2), security issues std., etc… .
- Library of reusable modules: Transforming code to allow a migration that requires important code changes often need to maintain modules that are going to be discontinued and new ones that are going to be used gradually. This information is managed by SCM and it's the source of several controls that have to ensure that the new written code is not 'legacy code' and that each program that's taken out of Production environment is returned to it working with the new modules.
- Failure Tracking System: It's another important source. There are a lot of simple defects that could be easily identified. For example, a typical case is the lack of end-of-file checks. It's very important to find the root cause of the defect, as in this way we can develop a rule and verify its compliance in the rest of the programs, because of the new tendency to create new code by using 'cut & paste' technique.

- Classic Code Inspection ("Fagan style"): Although they're not performed frequently, they always allow us to identify new defects.

Rules development and construction

After identifying all the sources, they should be analyzed so as to define the rules to be controlled. We should avoid ambiguity. Rules like "… *Cobol paragraphs should not be long* … "or "… *SQL commands should be simple* … "are useless.

Those rules which can be identified could have different formats. For example:

- Prohibited commands: such as ALTER, GO TO, SELECT *.
- Interface: such as the lack of return code checks.
- Data handling: such as the lack of I/O return codes.
- Exception handling: such as WHEN OTHER clause in EVALUATE statement, AT END clause in SEARCH statement.
- Configuration Management: such as the use of a module that is no longer supported.
- Performance: such as the JOIN of four tables or more.
- Cosmetic: such as how to indent certain commands.
- Naming convention: such as the minimal length of characters in variable names.
- Module size: such as the maximum number of statements it should contain.
- Etc.

A rule violation doesn't mean avoiding a defect, but if violated it makes the code more susceptible to the introduction of new defects during maintenance activities. In some cases there is the assertion about a potential defect, but it can also be a process improvement suggestion or a matter that requires attention. (Remember that the code must be correct, but it should also be understandable, efficient and maintainable).

Each rule belongs to a certain domain. This means that the control of the rule can be performed over the programs that belong to that domain. For example, there are certain commands that are allowed in a batch environment but they're not in an on line environment. Or, naming convention standard has to be followed by all the programs developed in the organization but cannot be demanded to third party developments already built.

Examples of different domains are:

- Batch & On Line

- New programs & Legacy programs
- In-house development & Third party developments
- DB2 / DLI / Vsam files

Once the rules have been developed and their domains identified, they should be validated with the people in charge.

Finally, the rules should be recorded in the Compliance Database (specifying their source and the date they come into effect).

Violated rules detection

The different ways in which rules could be broken within the code have to be evaluated. This task is not easy, and depending on the rules to be controlled, special skills to perform the task will be needed.

The ideal profile of the analyst is:

- Rule domain expertise
- Not less 3 years development experience'
- Survived a bad implementation of an application (Understands the pain of a poor quality application)

Some defects are easy to catch while others are extremely difficult. The language syntax should be well known in order to identify all the situations.

Implementation decision

Taking the decision which rules will be automated with what tool.

Two different kinds of inspection were identified.

- Automated: when a defect is identified without needing any later human analysis.
- Assisted: when a defect is identified but it needs later human analysis. For example, the code can be scanned for certain defined wild card search criteria (pattern-matching technique based) that finds potential defect candidates and then the final decision is made by the analyst through manual intervention. (Some defects can only be found by looking at every line of code and understanding the flow of logic and data within and between programs).

Many rules are controlled through assisted inspection. Sometimes this is because the information obtained from the program is not enough and some environment information is needed to decide if the rule is violated or not. This is the case of DB2 performance related rules. Depending on the number of files of the tables, there will be SQL commands that will be prohibited or not. In these cases, these commands are first detected and later the analyst will decide if there's a problem.

There are other cases when the author's program is needed. For example, this was the situation while searching date related fields in the Y2K conversion project. Variables were analyzed by name, definition and use. If the name of a field referred to 'date', 'year', 'yymmdd' or other such clue, it became a potential suspect. But unfortunately, it is well known that programmers use a great variety of reasonable and unreasonable names for their variables. Correctly finding all true candidates required knowledge about all ways that a date could enter a program and a thorough analysis of everything that could happen to those dates within the program.

So, not everything can be automated. But even less sophisticated tools (developed or bought) are faster and more accurate than performing the same task manually.

To buy or build your own tool is a personal decision that depends on a lot of factors (staff skills, time, money, etc.). The market offers a great variety of tools that help to identify defects accurately which could cause application failure, data corruption or unpredictable results (such as arithmetic overflow, unintialized variables). Selecting a tool is a hard activity. The difference in product cost is often due to product quality and reliability, vendor experience and level of available support (it's important to pay attention to capabilities rather than price). Depending on the level of accuracy you expect from the tool is the level of technique sophistication you will need from the tool (scanning, parsing, control flow analysis).

Automated Code Inspection Execution Process

A four-step process to execute the inspection.

Candidate programs identification

Several ways to select the programs to be inspected were identified.

- New programs / Modified programs: Every program that is taken into Production environment (either a new or an existing one with enhancements), should be inspected. This is the criteria used to perform the activity daily and to ensure that Production environment is not going to be contaminated.
- Programs belonging to a certain domain: In this case, all the programs that belong to a certain domain are inspected. This criteria is often applied when a rule is discovered and we want to ensure than there are not existing programs in Production environment that violate it.
- Reported as fixed: Each program that is reported as fixed is inspected in order to verify the defect removal and that no new defect has been added.
- On demand: Key projects ask for revisions early in the construction phase of the life cycle in order to evaluate quality and standard compliance during the construction phase (validating code from the very beginning is helpful)
- Random: Programs are often selected at random from Production environment in order to ensure programs remain defect-free.

Inspection execution

Once the set of programs to be reviewed has been selected, the inspection is executed. First, program's domain should be identified so as to perform the appropriate rules. Analysts start the inspection process, running the tools which automate the different rules.

After the inspection has been completed, analysts check the tool-generated results, reviewing defects with the application owner and making adjustments if necessary. Then, only defects (statements that violate the rules) are recorded in the defect tracking DB.

Defect reporting

Defects are reported to the ones in charge. Reporting date and responsible name should be recorded perform later tracking activities.

The report contains the following data:

- Inspection date
- Analyst
- Program name
- Reason for inspection selection
- Incident description
    - Line number
    - Violated rule description
    - Program statement/s that violate the rule
    - Comments

Tracking

A defect is completely removed only when the program is returned to Production environment and analysts verify it has been fixed satisfactory.

Unsolved defects are periodically requested to be fixed.

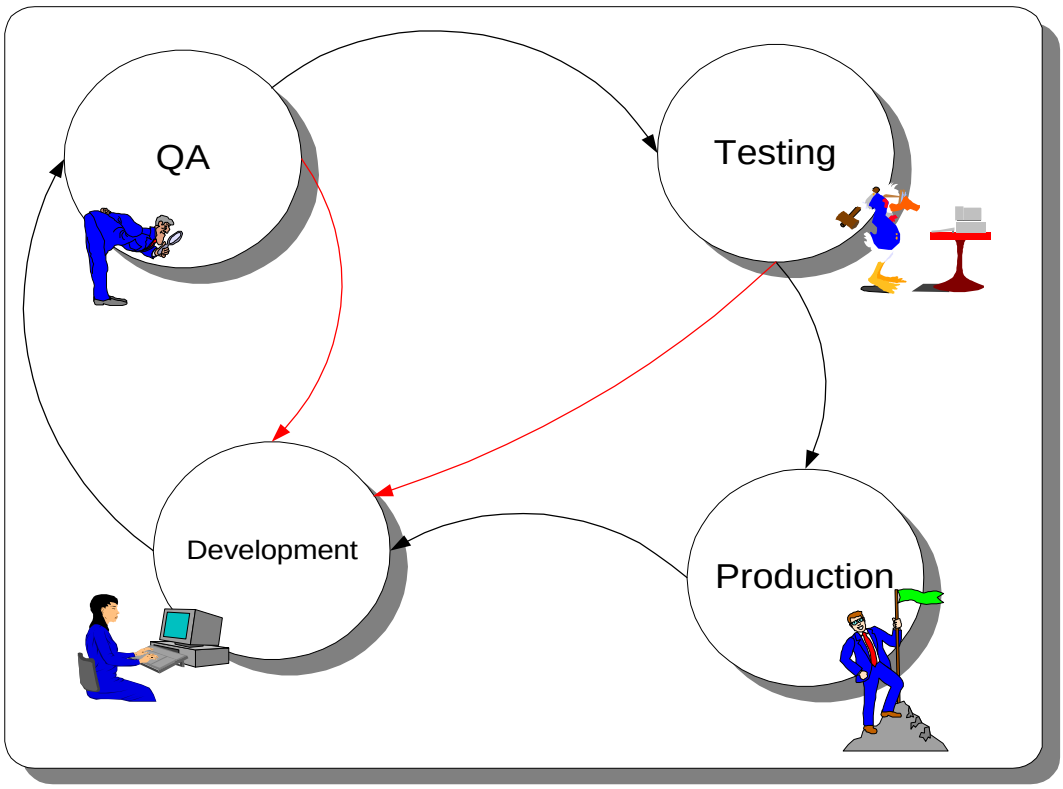
Metrics collections are performed to help the company to control, monitor and assess its software quality goals.

Where it fits in project life cycle

Inspection should be executed before testing, saving time and money.

The Development Cycle

Code is inspected prior to testing stage. All common and identified defects should be removed before a program leaves QA stage. In this way, inspection ensures that fewer trivial defects will delay the application testing, saving time and money (they are found faster and at a lower cost than in the testing environment).



Conclusions

Code Inspection is such a good practice that, even applying it partly, lets you obtain good results.

Inspection is the most useful and cost-effective form of error removal. Everybody agrees consistently in the value of inspections.

We experimented an alternative to classic inspection (Fagan style). The Y2K challenge and our today's schedule driven projects force us to find another approach that lets us reach inspection well known benefits. But this kind of inspection doesn't replace 'Fagan style' one. It complements it.

Detecting certain classes of defects in all the existing programs instead of searching for all the possible defects in one program gave us very satisfactory outcomes.

This is our experience. We know that the process has weaknesses and we have to continue improving. But we gave an important step and we came to the conclusion that there is no excuse for not doing inspections. We hope you agree.

QW2000 Keynote 10P2

Mr. Sanjay Jejurikar
Director of Test for Windows 2000
(Microsoft)

The Engineering Process of Windows 2000

BACK TO QW2000 PROGRAM

Presentation Abstract

This talk will give an overview on the engineering process used by Microsoft to develop and test Windows2000 product with a specific focus on the QA process. It will give a glimpse into the daily QA activity taking place in the teams while working on a project like Windows2000, and provide high level information on the way testing and deployment effort was staged.

About the Author

As a Director in the Windows Division, Sanjay is responsible for overseeing the building, testing and quality assurance of the Windows 2000 family of products.

Sanjay joined Microsoft in July 1989 as a software design engineer in the Windows NT test team and is part of this group since then in a variety of roles. His most recent role is managing the groups responsible for build, QA/OS testing , business applications integration testing , and logo testing for Windows NT products as a Director of Test.

Prior to joining Microsoft, Sanjay was a post graduate student in Computer Engineering at Syracuse University, New York. He holds a Masters degree in Computer Engineering from Syracuse University and a Bachelor of Electrical Engineering degree from VJTI, Mumbai, India.

BACK TO QW2000 PROGRAM

This talk will give an overview on the engineering process used by Microsoft to develop and test Windows2000 product with a specific focus on the QA process. It will give a glimpse into the daily QA activity taking place in the teams while working on a project like Windows2000, and provide high level information on the way testing and deployment effort was staged.

QW2000 Keynote 10P3

Gene Spafford
Director, CERIAS (Purdue University)

Information Security Requires Assurance

BACK TO QW2000 PROGRAM

Presentation Abstract

As computerization has become more ubiquitous, and as more critical societal functions have been entrusted to computers, we have become more aware of shortcomings in the security of our information infrastructure. Many (if not most) of these shortcomings can be traced to poor software development practices, lack of testing, and faulty design. Vendors claim -- in so many words -- that there is no economic reason to produce higher quality software because consumers want features before more security.

In this talk, I will examine these issues and discuss some future consequences of the lack of focus on quality. This will include discussion of how the issue may be addressed in law in the not-too-distant future.

About the Author

Eugene H. Spafford is a professor of Computer Sciences at Purdue University, the university's Information Systems Security Officer, and is Director of the Center for Education Research Information Assurance and Security. CERIAS is a campus-wide multi-disciplinary Center, with a broadly-focused mission to explore issues related to protecting information and information resources. Spaf has written extensively about information security, software engineering, and professional ethics. He has published over 100 articles and reports on his research, has written or contributed to over a dozen books, and he serves on the editorial boards of most major infosec-related journals.

Dr. Spafford is a Fellow of the ACM, Fellow of the AAAS, senior member of the IEEE, and is a charter recipient of the Computer Society's Golden Core award. Among other activities, he is chair of the ACM's U.S. Public Policy Committee, a member of the Board of Directors of the Computing Research Association , and is a member of the US Air Force Science Advisory Board. He regularly serves as a consultant on information security and computer crime to law firms, major corporations, U.S. government agencies, and state and national law enforcement agencies around the world.

Complete bio at: www.cerias.purdue.edu/homes/spaf.

BACK TO QW2000 PROGRAM

As computerization has become more ubiquitous, and as more critical societal functions have been entrusted to computers, we have become more aware of shortcomings in the security of our information infrastructure. Many (if not most) of these shortcomings can be traced to poor software development practices, lack of testing, and faulty design. Vendors claim -- in so many words -- that there is no economic reason to produce higher quality software because consumers want features before more security.

In this talk, I will examine these issues and discuss some future consequences of the lack of focus on quality. This will include discussion of how the issue may be addressed in law in the not-too-distant future.

QW2000 Workshop W1
Technology Track

Mr. Douglas Hoffmann
(Software Quality Methods LLC)

Oracle Strategies for Automated Testing

BACK TO QW2000 PROGRAM

Abstract...

Software test automation is often a difficult and complex process. The most familiar aspects of test automation are organizing and running of test cases and capturing and verifying test results. A set of expected results are needed for each test case in order to check the test results. Verification of these expected results is often done using a mechanism called a test oracle. This paper describes classes of oracles the author has used for various types of automated software verification and validation. Several relevant characteristics of oracles are included with the advantages, disadvantages, and implications for test automation.

When the topic of Oracles was first presented at Quality Week 1998 it had caused ripple effect. A two day LAWST (Los Altos Workshop on Software Testing) session was dedicated to discussion of oracles and test automation and several new papers on the subject were generated and presented by other people at conferences in 1999. I have updated and expanded the original work based on the LAWST workshop and further investigation, comparison, refinement, and critical reevaluation of oracles as used in automated software testing.

Real world oracles vary widely in their characteristics. Although the mechanics of various oracles may be vastly different, a few classes can be identified which correspond with automated test approaches. These types of oracles are categorized based upon the strategy for verification using the oracle. Thus, an oracle strategy using a lookup table to generate expected results can be the same as one that uses an alternate algorithm implementation to compute them. Four types of oracle strategies (and not using any oracle) are identified.

Author Bio...

Douglas Hoffman is an independent consultant with Software Quality Methods, LLC. He has been in the software engineering and quality assurance fields for over 25 years and now is a management consultant in strategic and tactical planning for software quality. For five years he served as the Chairman of the Santa Clara Valley Software Quality Association (SSQA), a Task Group of the American Society for Quality (ASQ). He has been a presenter and participant at dozens of software quality conferences and has been Program Chairman for several international conferences on software quality. He is a member of the ACM and IEEE and is active in the ASQ as a Senior Member, participating in the Software Division, the Santa Clara Valley Section, and the Software Quality Task

Group. He has earned a BA in Computer Science, an MS in Electrical Engineering, an MBA, a Certificate from ASQ in Software Quality Engineering, and has been a registered ISO 9000 Lead Auditor.

Douglas' experience includes consulting, teaching, managing, and engineering across the computer systems and software industries. He has over fifteen years experience in creating and transforming software quality and development groups, and has worked as an independent consultant for over ten years . His work in corporate, quality assurance, development, manufacturing, and support organizations provides a broad technical and managerial perspective on the computer industry.

BACK TO QW2000 PROGRAM

QW2000 Workshop W2
Applications Track

Mr. Cem Kaner

Bug Advocacy

BACK TO QW2000 PROGRAM

Key Points

Finding a bug is just the first part of the process of getting it fixed. This tutorial looks at the bug reporting process as a persuasive process -- finding ways to make the programming staff more interested in fixing defects and less likely to rely on excuses to not fix them. The tutorial looks at

- Technical issues (such as ways to find more powerful variations of a defect and ways to replicate a hard-to-replicate failure)
- Organizational issues (for example, find out which group in the company will pay the costs associated with this defect because they're your best advocates for getting it fixed)
- Persuasive writing (who are you trying to influence with this report and how does your writing advance your case?)

About the Moderator

Dr. Cem Kaner consults on technical and management issues, and practices law within the software development community. He is senior author of two books, Testing Computer Software, and Bad Software. He recently accepted an appointment as Professor of Software Engineering at Florida Institute of Technology. Also, in the talk with Marick and Bach, my name is misspelled as Kem Caner.

BACK TO QW2000 PROGRAM

QW2000 Workshop W3
Internet Track

Dr. Edward Miller
(Software Research, Inc.)

Achieving WebSite Quality

BACK TO QW2000 PROGRAM

Key Points

- How is WebSite quality assessed?
- What are WebSite failure modes?
- How can WebSite problems be predicted? Prevented? Compensated for?
- What are the best and worst WebSites in terms of quality?

Presentation Abstract

WebSite Quality measured by a user consists of many facets: Visual appeal, response time, quality of content, correctness of response, reliability, security, etc. This workshop investigates alternative routes to achieving WebSite Quality. Attendees will learn the state of the art in WebSite Quality approaches and technology and gain practical knowledge about

---

WORKSHOP OUTLINE:

- What *is* WebSite Quality
- WebSite Categories
  - Complexity: Static, Passive, Collective, Interactive, Responsive
  - Size
  - Criticality
  - Other Attributes
- Browser Dependencies
- Failure Modes
  - Time
  - Content
  - Other
- Static Analysis/Testing
  - HTML Checking
  - Link Checking

    - Content Checking
    - Other
- Dynamic Analysis/Testing
    - Validation Processes
        - Text
        - Images
        - Dialogs
        - Sequences
    - Performance Timing
        - Single and Multiple Download Timings
        - Overall User-level Response Times
        - Perceived User-level Response Times, Thresholds
- Load and Capacity Checking/Testing
    - Load Imposition
    - Load Measurement
    - User Scenarios
    - Realistic vs. Non-Realistic Experiments
    - Client-Based vs. Server-Based Experiments
- Recommended Procedures & Discussion

About the Author

Edward Miller is President of Software Research, Inc., San Francisco, California, where he has been involved with software test tools development and software engineering quality questions. Dr. Miller has worked in the software quality management field for over 25 years in a variety of capacities, and has been involved in the development of several families of automated software testing and analysis support tools.

Dr. Miller has chaired the SR/Institute Quality Week Conferences (QW, in the San Francisco Bay Area in late May, and QWE, in Brussels, Belgium, in November) since their beginning. He was chairman of the 1985 1st International Conference on Computer Workstations, and has participated in IEEE conference organizing activities for many years. He is the author of Software Testing and Validation Techniques, an IEEE Computer Society Press tutorial text. Dr. Miller received his Ph.D. (Electrical Engineering) degree from the University of Maryland, an M.S. (Applied Mathematics) degree from the University of Colorado, and a BSEE from Iowa State University.

BACK TO QW2000 PROGRAM

QW2K Workshop W4
Management Track

Mr. Robert Sabourin
(Purkinje Inc.)

The Effective SQA Manager -- Getting Things Done

BACK TO QW2000 PROGRAM

Presentation Abstract

This interactive tutorial walks you through several "down to earth" practical aspects of running an SQA team.

The tutorial is presented in parable form. In this tutorial the audience will experience the real life problems encountered by a NOGO.COMs neophyte SQA Manager "Fred". "Fred" must turn around an enthusiastic but severely under staffed and under budget team of SQA professionals working in a chaotic development environment into a productive effective team! "Fred" is under the gun - he has to get things done!

"Fred" is experiencing common pressures which dominate the software industry today:

* Tight time frame projects: "must ship it now", "Internet Time"
* Frequently changing requirements: "cannot sell without this last minute change"
* Communications problems: "they changed the code but not the spec and then they didn't tell the test team"
* Hiring and keeping staff: "there is a shortage of qualified staff on market - lot's of pressure to move - if your key SQA lead gets upset he can be in a new job tomorrow!"
* Keeping up with technology: "which tools should we use?"
* Test automation: "what should be automated, when - who has time to automate?"
* Understanding shifting business priorities: "Why is this project so important anyhow? How do I tell how to split resources within my team?"
* Co-ordination with other teams: "who did what when to whom and why?"
* Not enough people to test: "short staffed"
* Not enough time to test: "under the gun"
* When we have people to test we do not have a product ready for test: "why are we always waiting"
* Test coverage: "what should and should not be covered?"
* Estimating: "time and effort?"
* Making and keeping commitments: "What to commit to, when and how?"
* Capturing important data for downstream process improvement: "we are already using

every second to get as much of the job done as we can - how can be start capturing metrics?"
* SQA is always getting blamed and squeezed because the product is LATE!

"Fred" of course cannot "do it alone". In search of wisdom "Fred" gets advice from many mentors - each of whom helps "Fred" conquer another hurtle! Incrementally "Fred" builds his organisation into a world class effective team famous for "getting things done!" consistently "On-Time", "On-Quality" and "On-Budget".

The tutorial covers practical aspects of SQA management and organisation:

* Goals
* Business drivers
* Roles and responsibilities
* Decision makers, decision making, dealing with bugs
* Getting buy in from other teams, paradigm shifts
* Test planning, prioritisation, execution
* Staffing, triage, creative resource usage, getting some free resources
* Automation what, when, how
* Incremental process improvement
* Starting a practical, useful and used metrics program
* The many faceted role of formal inspections
* Effective software requirements and specifications
* Focus on doing "the right things" "the right way" (tuning tunnel vision)
* Marketing and promoting excellence
* Building team spirit - involving people in business decisions

Several practical examples are given.

About the Author

Robert Sabourin has been involved in all aspects of development, testing and management of software engineering projects. Robert graduated from McGill University in 1982. Since writing his first program in 1972, Robert has become accomplished software engineering management expert. He is presently the Director of Research and Development at Purkinje Inc; a Montreal-based international firm specializing in the development of sophisticated, critical medical software used at the point of care. Previously, Robert managed Software Development at Alis Technologies for over ten years. He has built several successful software development teams and champions the implementation of "light effective process" to achieve excellence in delivering on-time, on-quality, on-budget commercial software solutions.

Robert has championed many complex international multilingual software development and globalization efforts involving several intricate business partnerships and relationships including international government (Czech, Egypt, France, Morocco, Algeria...) and commercial entities (Microsoft, IBM, AT&T, HP, Thompson CSF, Olivetti...). Systems included concurrent coordinated multilingual multiplatform product releases.

Robert's pioneering work with Infolytica Corporation led to the development of the first commercially

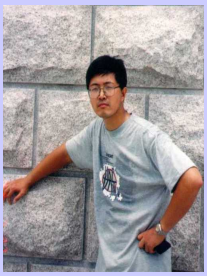
available platform independent graphics standard GKS and several toolkits which allowed for cross platform development and porting of complex CAD, Graphics, Analysis and Non-Destructive Simulation systems.

Robert is a frequent guest lecturer at McGill University where he relates theoretical aspects of Software Engineering to real world examples with practical hands-on demonstrations.

In 1999, Robert completed a short book illustrated by his daughter Catherine entitled "I Am a Bug" (ISBN 0-9685774-0-7). Written in the style of a children's book, "I am a Bug" explains elements of the software development process using a fun metaphor. Throughout his career, he has also been the author of several articles and papers, and has given presentations relating to software development at a number of international conferences. Most recently, Robert presented an interactive half-day tutorial on Bug Priority and Severity at Software Quality Week in Belgium - November 1999.

Robert has received professional recognition for many accomplishments over the years. These include Byte Middle East's 1992 Product of the Year for the AVT-710 product family achieving a ZERO FIELD REPORTED software defect rate with over 15,000 units installed. (Project involved over 27-man month's effort!); Quebec Order of Engineers' recognition for creating and managing the Alis R&D Policy Guide - Development Framework and process; CANARIE Gold Proposal for CANARIE PHASE I (Internet en français); and CANARIE Gold Proposal for CANARIE PHASE II (multilingual internet).

BACK TO QW2000 PROGRAM



QW2000 Standby Paper SB2

Mr. Yongzhong Tu & Mr. Wei-Tek Tsai, Baisu Huang, Raymond Paul (University of Minnesota)

An Approach to Testing Component-Based Software

BACK TO QW2000 PROGRAM

Key Points

- Test scripts should be delivered with components
- Component vendors and component users use different views of the test scripts of the components
- Using ripple effect analysis to reuse test scripts for component testing can significantly reduce testing cost
- MtSS/MgSS/MfSS+ are good language for component specification, test script generation

Abstract

Currently many object-oriented (OO) applications are developed using software components. It imposes more pressure on the integration testing because of the different perspective of component suppliers and users. How to reuse test scripts and how to reduce test redundancy are the main issues in testing component-based software.

In this paper, an enhanced message framework sequence specification (MfSS+) is used as hierarchical test scenario templates to generate test scenarios for the component-based software testing. MfSS+ was an extension of MtSS and MgSS originally proposed for testing OO applications and it was proposed to test OO design patterns, frameworks, and their applications. This paper uses the MfSS+ to specify test specifications for components. The specification can be used by component suppliers to test the component before delivery, and it can be used by component users during application development and maintenance. Techniques in test specification slicing, test scenario generation and applying traditional testing techniques in test scenario generation are introduced.

Component suppliers and component users have different responsibilities in the component-base system testing, the former test the component itself and provide component test specifications for component users, the later verify the configured component's functionalities and test the interactions between the component and its client objects. So, they use different test scripts in testing. Test scripts used by component suppliers concern with testing component interfaces with internal views, while those used by component users concern only with the correctness of used part of the functionalities of the component, thus test component interfaces with external views of the component.

Even more, test scripts vary in different phase of the integration process and contain different sets of

An Approach to Testing Component-based Software

Yongzhong Tu, Wei-Tek Tsai, Baisu Huang
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
{Tu, Tsai}@cs.umn.edu

Raymond Paul
OASD, Y2K Office
Washington, D.C.
PaulR@osd.pentagon.mil

Overview of the paper

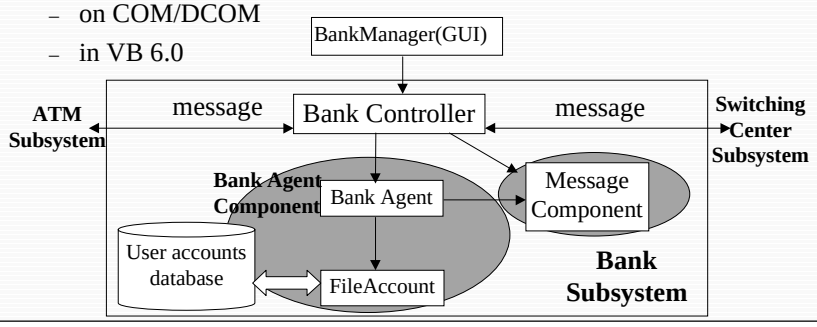
- Component Test Specifications
- Test Scenario Generation from Test Specification
- Test Script Derivation
- Component Testing with Different Views
- Component Integration Testing
- Test Case Reuse by Using Ripple Effect Analysis
- An Example
- Conclusion

Techniques proposed

- Component test specifications
  - Using MfSS+ to specify component
- Test scenario generation from component test specifications
  - Applicable to component supplier and user
  - Various kinds of test cases can be generated
- Different views of the component test specifications
  - Component user and component supplier use different views of the component test specifications
- Application integration testing with component test specifications
  - Hierarchical test specifications
  - Use different test specifications in different test phases
  - Test case reuse

Example: ITSS Bank Subsystem

- Inter-bank Transaction Switching-center System (ITSS)
  - Follows ISO 8583
  - on COM/DCOM
  - in VB 6.0



MtSS, MgSS, MfSS and MfSS+

- In MfSS+, various data constraints such as dynamic data constraints, static data constraints, sequence constraints and object relationship constraints, are specified

	Message Source	Message Destination	Constraints
MtSS	Multiple classes	Single class	Sequence that methods of a class are called
MgSS	Single class	Multiple classes	Sequence that a method calls other methods
MfSS	Multiple classes	Multiple classes	Sequence that objects should follow Dynamic binding and dynamic typing involved in the sequence are specified.
MfSS+	Multiple classes	Multiple classes	Data constraints: preconditions and post-conditions of an operation, data invariants through the whole lifecycle of an object

Why using MfSS+

- IDL specifies the interface of component
- MfSS+ is more powerful in testing
  - Formal specifying the sequence constraints
  - Various testing strategies can be used
  - Data constraints facilitate the input and expected output derivation of test cases
  - Hierarchical arrangement of test specification facilitates test driver and stub construction
  - Different views can reduce test case redundancy

Using MfSS+ in testing

- Specify component in MfSS+
- Derive test specifications from component specification
  - Specify test scenario templates in MfSS+
- Generate test scenarios from test scenario templates
- Derive test cases from test scenarios
  - Derive input and expect output from data constraints

Using test strategies in generation

- Partition testing, boundary testing, negative testing, positive testing, stress testing, random testing
- Special test scenarios
  - Testing dynamic typing and dynamic binding
  - Extensibility
  - Communication
  - Exception handling

Test case generation

- Different views of test specifications
- Generate test scenarios
  - Plug in concrete objects
  - Expand branches to full sequence
  - Determine iterations
  - Determine preconditions and post-conditions
- Derive input and expected output from data constraints

Test script

```
//Test script for component integration
#Mode = Integration testing
#MfSS+ for component BankAgent
BankAgent<sub>create</sub> • ({# Message.getBankId() ==
BankAgent.BankId #}BankAgent<sub>process</sub> • ({#
Message.getTransactionType() == ”withDraw” #}
Account<sub>withdraw</sub> | {# Message.getTransactionType() ==
”checkBalance” #} Account<sub>checkBalance</sub>) | {#
Message.getBankId() != BankAgent.BankId #}
BankAgent<sub>forward</sub>)* • BankAgent<sub>close</sub>
//……
#end of MfSS+
#Test cases for component BankAgent
#1: POSITIVE
objSendMsg=CreateObject(“Message.Imessage”);
objSendMsg.setAttributes(“Withdraw”,”0000111122”,”Password”,4287);
objBankAgent=createObject(“BankAgent.Iwithdraw”);
objectBankAgent.process(objSendMsg);
#/1
#2//test case 2
//…….
#end of test cases for component BankAgent
//……..
#end of test script
```

Comments

Unit testing or integration testing

Begin of MfSS+ for the component

MfSS+

End of MfSS+

Begin of test cases for a component

Test case #

Test case

End of a test case

End of test cases

End of test script

Component testing with different views

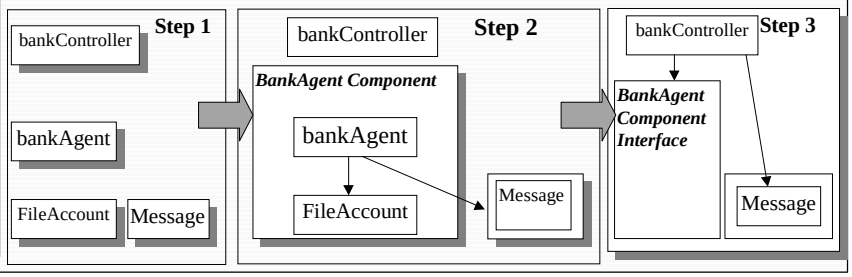
- Component suppliers
  - Test component with internal views
  - Supply component with specifications
- Component users
  - Verify the configured component functionalities
  - Test the interactions between component and its application with external views

Component testing with different views (Cont')

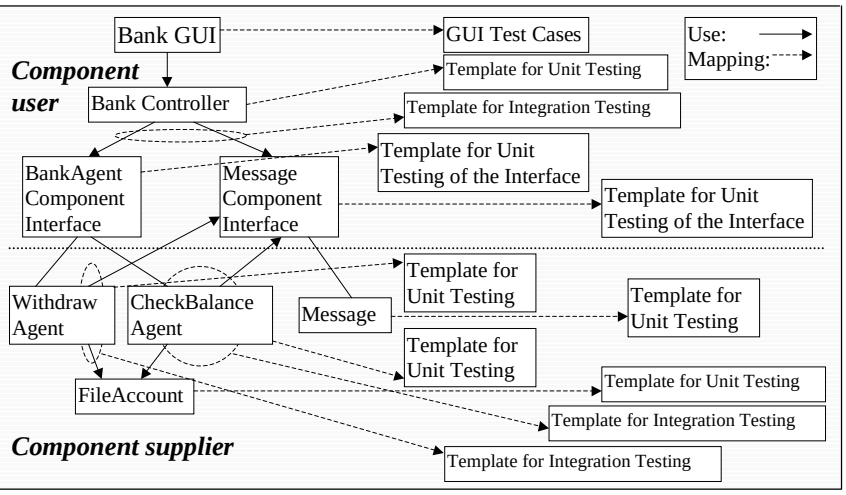
- Component suppliers
  - Component specification
  - Unit testing and integration testing
- Component users
  - Reconfigure component services and slice test specifications
  - Test the component interface as a unit
  - Test the integration of the component with application

Component Integration Testing

- Object unit testing
- Component integration testing
- Test component interface as a unit
- Integrating component into the application



Software structure and test scenario template hierarchy



Test case reuse through ripple effect analysis

- Object interface not changed
  - Reuse all test cases and redo unit testing
- Component interface not changed
  - Component user can reuse all test cases and run them to test the component as a unit
- External view of the component interface not changed
  - Component user can reuse all test cases and run them to test the component as a unit
- External view changed
  - Regenerate test cases and redo everything

Example:test scenario generated

# of test scenarios		*Bank Controller*	*Message Component*	*Withdraw Agent*	*CheckBalance Agent*	*FileAccount Component*
Object testing		124	32	78	55	65
Component Integration	Sliced	None	None	48		
	Unsliced	None	None	278		
Component Unit Testing	Sliced	None	32	42		
	Unsliced	None	32	278		
System Integration	Sliced	63				
	Unsliced	Up to thousands of test scenarios can be generated				

- Most test scenarios are negative
- Reduce redundant test cases significantly
- Enhance efficiency

Example: test result

- 96 bugs seeded
- Various kinds of bugs
  - Boundary mismatching bugs, branch condition bugs, wrong expression bugs, loop bugs, wrong sequence bugs, variable bugs, operation-missing bugs and type bugs

Objects	# of bugs seeded	# of bugs detected			
		Object	Component Integration	System integration	Total
BankController	17	16	No testing	5	17
Message Component	8	7	No testing	8	8
WithdrawAgent	12	11	10	7	12
CheckBalance Agent	10	10	9	6	10
UserAccounts Manager	8	8	4	2	8

Example: reuse test cases

	Change1 FileAccount	Change2 FileAccount	Change3 Withdraw Agent	Change4 Withdraw Agent	Change5 Withdraw Agent	Change6 BankCont roller
Object specification	No	Yes	No	Yes	Yes	No
Component interface	No	No	No	No	Yes	No
Component external view	No	No	No	No	No	No
Inner object	Yes	Yes	No	No	No	No
Integration testing by vendor	No	Regenerate test cases	Rerun	Regenerate test cases	Regenerate test cases	
Unit testing by vendor	Rerun Test cases	Regenerate test cases	Rerun	Regenerate test cases	Regenerate test cases	
Integration testing by user	No	No	No	No	Regenerate test case	Yes
Unit testing by user	No	No	Rerun	Rerun	Regenerate test cases	Yes

Conclusion

- MfSS+ facilitates component specification for testing
- Component test specification can be used to generate test cases either by component supplier and user thus significantly save costs
- Component user and component supplier use different views of the component test specifications
- Hierarchical test specifications and ripple effect analysis enhance test case reuse in integration testing

Acknowledgement

- This project is supported by OASD Y2K Office through Army Research Laboratory.

test cases derived from the test scenarios generated from the specific MfSS+. By using ripple effect analysis and test specification slicing techniques, test cases can be reused and test redundancy can be reduced.

A COM-based inter-bank transaction switching-center system (ITSS) is used as an example to illustrate the techniques present. The test results show that the approach proposed in this paper significantly reduced the test redundancy and enhanced the test script reuse.

BACK TO QW2000 PROGRAM

AN APPROACH TO TESTING COMPONENT-BASED SOFTWARE

Yongzhong Tu, Wei-Tek Tsai, Baisu Huang

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455

Raymond Paul

OASD, Y2K Office
Washington, D.C.

ABSTRACT

Currently many object-oriented (OO) applications are developed using software components which imposes more pressure on testing because of the different perspectives of component suppliers and users. How to reuse test cases and how to reduce test case redundancy have become the main issues in testing component-based software. In this paper, an enhanced message framework sequence specification (MfSS+) is used as hierarchical test scenario templates to generate test scenarios for component-based software testing. Test scripts vary in different phases of the integration process and Ripple effect analysis techniques can be used to enhance test case reuse and reduce test case redundancy in regression testing. A COM-based inter-bank transaction switching-center system (ITSS) is used as an example to illustrate the techniques presented. The test results show that the proposed approach significantly reduce test case redundancy and enhanced test case reuse.

Keywords

Object-oriented testing, Component testing, integration testing, test script reuse, unit testing

1. Introduction

Components are objects that are built following the specifications of a component platform [Szyperski 1997]. Several component platforms have been proposed and implemented including CORBA, JavaBeans, and COM/DCOM [Brown 1996]. A component-based system is composed of reusable components and other software. Components have well-defined interfaces, often work in a well-defined infrastructure, and most existing components are based on object-oriented (OO) technology.

This paper considers a component-based system that consists of three parts: the user application, the components, and the infrastructure that provides communication channels between user application and components. The user application communicates with components through interfaces mapped by the communication infrastructure. A component provides multiple sets of functionality and each set of the functionality can be accessed by a configuration.

This paper discusses testing a component-based system from two perspectives: component provider's and component user's. A component provider may have limited knowledge of how components will be used by component users. A component user, in contrast, may not have complete knowledge of the component supplied, but must know how components are used in the application. The component user is thus concerned with only those aspects of components that are relevant to the application.

Component-based software introduces new problems in software development and maintenance, hence, testing component-based programs has led to a new research direction. Integration testing, regression testing and functional testing of component-based software have not yet been addressed in the literature. Only a few guidelines are available to test applications developed using components where state machine and data-flow approaches [Harrold 1999, Buy 1998 and Sohn 1999] or specifications of components are proposed [Cho 1998, D'Souza 1999].

This paper presents the following techniques to address this problem:

 ©Copyright 2000. All rights reserved by authors

1) Component test specifications: a component often has its functional or design specification, e.g., a component may use CORBA's IDL to specify its input and output. In addition to these specifications, this paper proposes a test specification for each component that uses the enhanced Message Framework Sequence Specifications (MfSS+) [Tsai 1999, Tu 1999]. The specification can be used by component suppliers to test the component before delivery, and it can be used by component users during application development and maintenance.
2) Test scenario generation from component test specifications: test scenarios can be generated from test specifications. Both component suppliers and users can use these scenarios to generate test cases and scripts in testing component-based software.
3) Different views of the component test specifications: component users may configure the functionality that the suppliers provided; some services may not be used by the application. Thus only the sliced test specifications, which are called MfSS+ views, are used as test scenario templates to generate respective test scenarios.
4) Application integration testing by using component test specifications: An important issue in testing component-based software is integration testing. An approach is to develop a set of hierarchical test specifications for the application by using component test specifications. In different phases of integration, we use different test specifications as test scenario templates to generate an efficient set of test cases. Redundant test cases can be recognized and need not be used again, addressing the reuse issue.

2. An Example – Bank Subsystem of ITSS

This section presents the bank subsystem of a component-based software on COM/DCOM: inter-bank transaction switching-center system (ITSS). The system follows the ISO 8583 standards and is implemented using Visual BASIC version 6.0. This paper uses the bank subsystem for illustration.

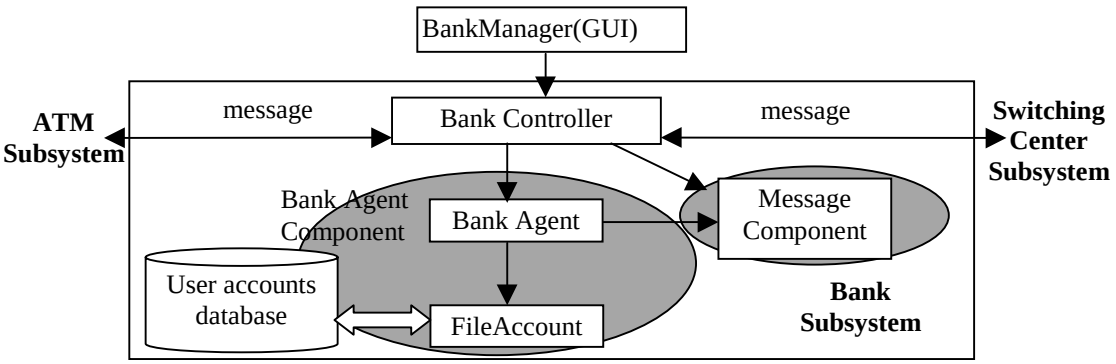


Figure 1: Architecture of the bank subsystem

Transactions and request approval or denial occur at the bank subsystem. The architecture of the subsystem can be shown as Figure 1. This subsystem has two components: *Message* component and *bankAgent* component. The *Message* component, used by all subsystems, manipulates the messages transferred on the net. The *FileAccount* object manages the user accounts database and is only used by the *bankAgent* object and those integrated with it to form the *bankAgent* component. The *bankAgent* object has two different concrete objects, *WithdrawAgent* and *CheckBalanecAgent*, which handle respective transactions. Both components are used by the *BankController* object, which is responsible for: 1. accepting sign in/out commands from the GUI of the bank manager and sending the requests to the SC it is connected; 2. accepting transaction messages sent by ATM or SC, forwarding them to SC if they are not to it or creating respective *bankAgent* objects to handle the requests otherwise.

 ©Copyright 2000. All rights reserved by authors

3. Component Test Specifications

To facilitate component-based system testing, testing issues need to be considered in the design phase. In our approach, component testing is based on component specifications and component-based system testing is based on system specifications. We use MfSS+ to specify the components as well as the system using them.

- **MtSS, MgSS and MfSS+**

Method sequence specification (MtSS) is a language used to describe the constraints on the order in which the methods of a class are invoked. Message sequence specification (MgSS) is another kind of language to describe constraints on the order in which the methods of other classes are invoked within a method of a class. MfSS+ is an extension of MtSS and MgSS, and it specifies the sequence constraints on the interactions between objects. Further more, in MfSS+, various data constraints such as dynamic data constraints, static data constraints, sequence constraints and object relationship constraints, are specified in the preconditions and/or post-conditions in logic expressions [Tu 1999]. They were proposed to test OO design patterns, frameworks and their applications. They have been extended to concurrent applications [Wang 1997] and real-time applications [Vishnuvajjala 1996]. The differences are listed in Table 1.

	Message Source	Message Destination	Constraints
MtSS	Multiple classes	Single class	Sequence that methods of a class are called
MgSS	Single class	Multiple classes	Sequence that a method calls other methods
MfSS	Multiple classes	Multiple classes	Sequence that objects should follow Dynamic binding and dynamic typing involved in the sequence are specified.
MfSS+	Multiple classes	Multiple classes	Data constraints: preconditions and post-conditions of an operation, data invariants through the whole lifecycle of an object

Table 1: Differences of MtSS, MgSS, MfSS and MfSS+

- **Some Concepts about Specifying Components**

Test specification for a component contains a set of test scenario templates for test scenario generation and test script derivation. Although any software comes with its unique specifications, it is necessary that different test specifications be used for different purposes (unit testing or integration testing) and for different perspectives (component suppliers or component users).

A test scenario is a typical sequence of messages sent to objects. Test scenario templates are templates used to derive test scenarios. One test scenario template can generate a family of test scenarios. Because MfSS+ specifies the sequence constraints on the messages among application objects, it can be used to specify test scenario templates. A test scenario template is not simply copied from the respective MfSS+, it is usually sliced from the respective MfSS+ for certain reasons. The sliced MfSS+ of a MfSS+ is called MfSS+ view. All test scenarios generated are represented with the same grammar as that of the MfSS+ except:

1) No undetermined symbols appear. For example, all iteration marks are replaced by concrete numbers, all abstract classes are replaced with concrete objects of their subclasses.
2) The logic expressions appear in the preconditions and post-conditions are no longer data constraints but tell us the status at that time.
3) No branches appear in the scenarios.

Applications are often organized hierarchically; the architecture of the applications affects the integration process and the organization of the test scenario templates in the test specifications [Tu 1999]. Generally, each object has a test scenario template for unit testing and another one for

 ©Copyright 2000. All rights reserved by authors

integration testing. As for component, the services offered by the component suppliers may be configured by the component users, but not all services are used by the application. Thus the test scenario templates that the component suppliers provided are necessarily sliced to new test scenario templates.

Generally, MfSS+ is used to specify components where the specification is intentionally sliced to different test specifications which contain test scenario templates, and finally, test scripts are derived from the templates to provide test scenarios or test cases.

- **Why Specify Components in MfSS+?**

Nowadays, component suppliers often use Interface Definition Language (IDL) to specify their components. With IDL, component developers can specify the interfaces of their components. In other words, the IDL file specifies the services that the component provides and the input and output of the component. However, from the testers' point of view, it is not sufficient enough. Component testers need sequence specifications of the component to perform testing. These sequence constraints are not specified in the IDL file, let alone the data constraints for the sequence.

On the contrary, MfSS+ is more powerful in testing than IDL file, thus it is used as a supplement in component-based system testing. It has the following advantages:

1. MfSS+ can specify sequence constraints in a formal way, all sequences that can not be derived out from the MfSS+ specification can be regarded as invalid.
2. Various testing techniques such as partition testing, random testing, negative testing and boundary testing can be used in test scenario generation from MfSS+.
3. By studying preconditions and post-conditions of the MfSS+ for a sequence, test input and expected output can be inferred out.
4. Hierarchical arrangement of the MfSS+ can facilitate the test driver and stub construction.
5. By using different MfSS+ views as test scenario templates, testers can generate different sets of test scenarios for different purposes. When a change is made, not all test cases need to be run again. Testers can use a relevant MfSS+ to generate related test scenarios. This method can systematically reduce redundant test cases in integration testing and component/ component-based system maintenance.

4. Test Scenario Generation from Test Specifications

MfSS+ can be used as a scenario template from which test scenarios can be generated [Tsai 1999, Tu 1999]. Given the scenario templates, testers can focus their testing on different aspects of the application by using common testing techniques such as partition testing, boundary testing, random testing, positive testing and negative testing. Test scenarios can be generated to test dynamic typing and dynamic binding testing, extensibility and communication testing with MfSS+ [Tsai 1999].

- **View Generation by Test Specification Slicing**

Component specifications in MfSS+ systematically describe the sequence constraints of the component along with the detailed data constraints. To reduce the number of redundantly used test cases, testers often focus on features of the specifications.

One can apply the program slicing techniques to sequence specifications to obtain a subsequence related to certain aspects. The test scenarios used for unit testing of *WithdrawAgent* are derived from its MfSS+ according to the object type, and the test scenarios used for integration testing are derived by slicing out the messages related to inter-object communication.

 ©Copyright 2000. All rights reserved by authors

By using test specification slicing, one can get test scenario templates from the sequence specifications. Because the so-generated test scenario templates focus on the given aspect, test case reuse can be facilitated and redundant test cases can be reduced. At the end of this paper, test results are given to illustrate the power of this technique.

- **Generate Test Scenarios from Templates**

To derive real test scenarios from a test scenario template, testers need to do the following:

1. Plug in concrete objects for abstract objects. For example, the abstract object *BankAgent* will be replaced with concrete object *WithdrawAgent* or *CheckBalanceAgent* to get the scenario.
2. Expand a branch to a full sequence. The final test scenario has no optional operations. Thus a template that contains branches is sure to generate more than one test scenario.
3. Undetermined iteration marks should be replaced with concrete numbers. For example, the fuzzy iteration constraint symbol "*" should be replaced with a concrete number, for example, zero or one to get the test scenarios.
4. The logic expressions appear in the preconditions and post-conditions are determined to stand for the concrete status at that time.

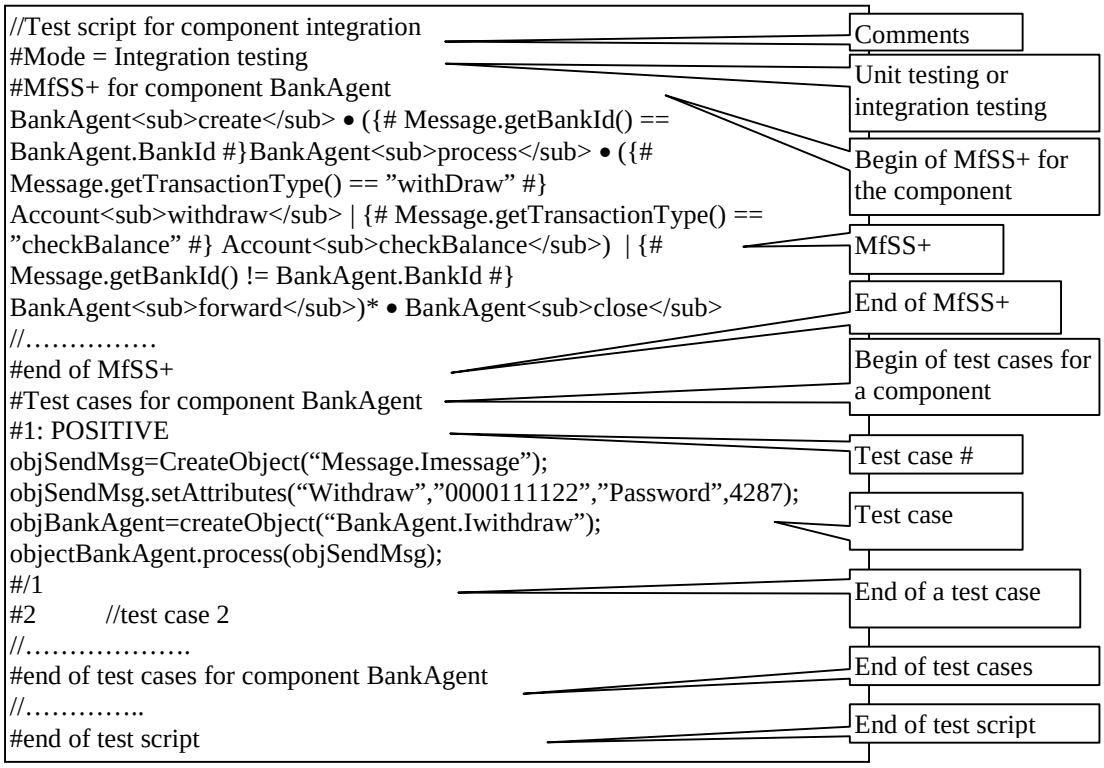


Figure 2: An example test script

5. Test Script Derivation

To automate the testing process, preparation of a well-organized test script is very important. In our approach, each test case contains a set of methods that are called according to a certain sequence and preconditions and post-conditions for each method.

The test script derivation process can be like the following:

1. Slice proper MfSS+ view from the test specification of the component;
2. By using different test strategies, generate test scenarios;

 ©Copyright 2000. All rights reserved by authors

3. By studying test scenarios, infer the input and the expected output to generate appropriate test cases;
4. Write test cases into test scripts in a certain format.

For the bank subsystem example, a test script is developed for system integration testing, component integration testing, object testing and unit testing of the component objects, respectively (Figure 2).

6. Component Testing with Different Views

Component suppliers and component users have different responsibilities in the component-based system testing, the former test the component itself and provide component test specifications for component users, the latter verify the configured component's functionalities and test the interactions between the component and its client objects. Thus, they use different test scripts in testing.

- **Testing Component Interfaces with Internal Views**

The component suppliers are responsible for the component specification testing, unit testing of the component objects and the integration testing of the component. All services the component is designed to provide should be tested. The test scenario templates used by the component suppliers are more complex than those used by component users. These templates are internal views of the component.

- **Testing Component Interfaces with External Views**

Component users may get the component from the Internet. They need to test the interface of the component and its specification to verify its services and the method to get the service. In addition, it is necessary that component users use the component in a different context from that of the component suppliers. Users might not require all services from the component and care only about the correctness of a part of the functionalities of the component. In this case, it is obvious that the component users need not run all the test scenarios originally designed to test all services.

From the component users' point of view, one component can be regarded as a unit to be tested. The reason is that its source code may not be available, and no matter how complex the component is constructed, only the specifications of the interface can be seen. In this case, the component poses the same place as an object does. Thus the idea is to treat the component as an object. The users apply unit testing techniques on the component at first, and integration testing techniques when it is integrated with the application.

The templates used by component users are external views of the component. Unused services can be sliced out from the MfSS+. One component has only one internal view but may have several different external views in different contexts (Figure 3). Different users may have different sets of test cases.

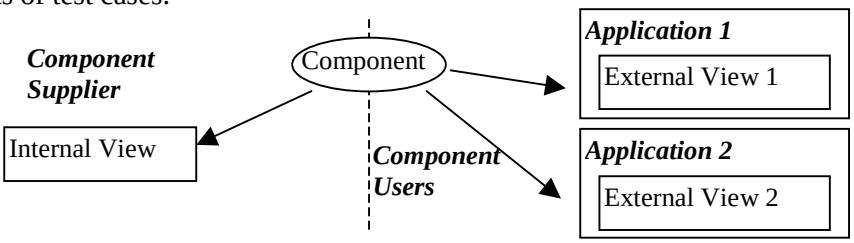


Figure 3: Different views of a component

 ©Copyright 2000. All rights reserved by authors

7. Component Integration Testing

One issue for component-based system testing is how to do integration testing without running too many redundant test cases [Jorgensen 1994]. Because each object that composes the component has had unit testing, not all test cases need to be run again. Only the test cases that are related to the communication between objects integrated are to be run again. Thus the test cases that have no tight relationship with the integration are significantly reduced.

On the other hand, when doing system integration testing, the component is regarded to have been tested as an object. The external view of the component is used to generate test scenarios for the integration.

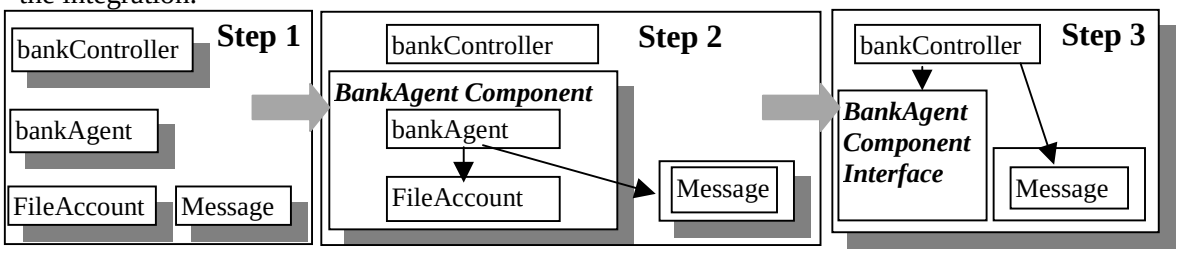


Figure 4: Three steps of integration testing of component-based system

In general, unit testing focuses on the functionality testing of objects and integration testing focuses on the communication between the newly integrated objects. In each integration step, a MfSS+ view can be used as a test scenario template to generate test scenarios for the integration testing of the step. Figure 4 illustrates three steps of the integration testing. In the first step, component suppliers and component users do unit testing for each objects they developed with test scenario templates for the objects. In the second step, the supplier of the *bankAgent* component integrates *bankAgent* object, *FileAccount* object and *Message* component together to form the *bankAgent* component, a special test scenario template is used for this step. Step three shows that the component user integrates the *bankAgent* component and *Message* component with *bankController* object, another special MfSS+ view is used as a template to generate test scenarios for this step.

8. Test Case Reuse by Using Ripple Effect Analysis

Another issue for component-based system testing is the ability to reuse test cases when a change is made to a component. The goal is to run the fewest test cases as possible.

Test scenario templates are often organized according to the hierarchy of the system. For component-based system testing, each object including component objects, has a test scenario template for unit testing and a test scenario template for integration testing. A component has three different sets of test scenario templates: one for the unit testing of the component objects, one for the integration testing of the component and another for unit testing of component interface. These test scenario templates compose a hierarchy similar to the software structure in Figure 5. This hierarchy indicates the dependency relationship as well [McGregor 1997].

According to Figure 5, the testers can do ripple effect analysis (REA) [Wang 1996] for test case reuse when an object is modified. The REA for component testing mainly considers the following issues:

1. Is the interface of the object changed? If the interface of the object is changed, the specification of that object is also changed, thus testers need to derive new test scenario templates and regenerate the test scripts for unit testing and related integration testing. If the interface is not changed, all test cases can be reused. Test cases that need to be run again are those for unit testing of the object, integration testing of the object integrated with the used and using objects.

 ©Copyright 2000. All rights reserved by authors

2. Is the interface of the component changed? As for component, the interface of the component is separated from the implementation of the component. It is necessary that when the interface of the implementation of the component is changed while the interface of the component is kept stable. In this case, all the test cases used by component users can be reused while the component supplier has to regenerate test scripts for the changed component implementation.
3. Is the external view of the component changed? Since the component users actually use external view of the component to do testing. It is possible that the change of the interface of an object may not require the regeneration of test scripts for component users because the change may not affect the external view of the component or the service is not directly used by the application.

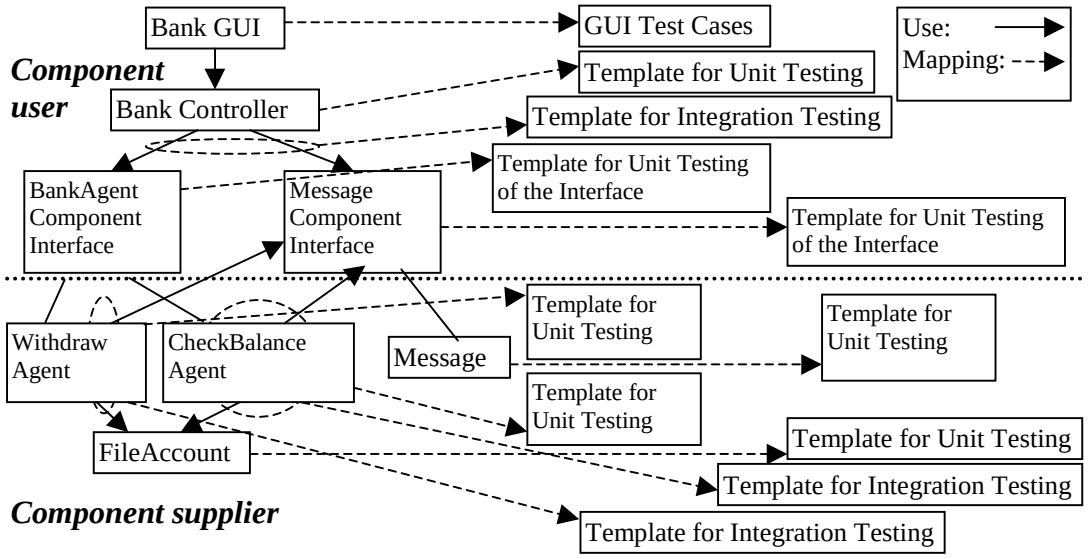


Figure 5: Test scenario template hierarchy of the bank subsystem

By using REA, it is unnecessary to run all test cases again when a change is made to the component and sometimes only related test scripts need to be regenerated, saving a lot of time and money.

9. Test Results

- **Bank subsystem testing**

The number of the test scenarios we used in each phase and those generated from the original specification are listed in Table 2. Most test scenarios (up to 70%) are negative test scenarios. The unsliced MfSS+ is too complex to generate test scenarios, which has eleven operations without counting the constraints. From the table, we know that the technique of MfSS+ slicing significantly reduces the number of test cases for integration testing.

To illustrate the effect of the techniques we present in this paper, we seeded ninety-six bugs into the subsystem (not at same time), and performed the whole test process. The seeded bugs include various kinds of bugs, such as boundary mismatching bugs, branch condition bugs, wrong expression bugs, loop bugs, wrong sequence bugs, variable bugs, operation-missing bugs and type bugs.

The test results are listed in Table 3. From the table we know:
1) Object testing missed two bugs related to the communication,
2) Component integration testing detected less bugs than that of object testing, but detected out the missed bug,

 ©Copyright 2000. All rights reserved by authors

3) System integration testing detected much less bugs, but the last missed bug was found,
4) All bugs were detected out,
5) Eighty twenty rules applied here: 20% test cases can detect out 80% bugs. Actually, after only 10 test cases were run, all seeded bugs seeded in the *WithdrawAgent* were detected out.

# of test scenarios		***Bank Controller***	***Message Component***	***Withdraw Agent***	***CheckBalance Agent***	***FileAccount Component***
Object testing		124	32	78	55	65
Component Integration	Sliced	None	None	48		
	Unsliced	None	None	278		
Component Unit Testing	Sliced	None	32	42		
	Unsliced	None	32	278		
System Integration	Sliced	63				
	Unsliced	Up to thousands of test scenarios can be generated				

Table 2: Number of test scenarios for subsystem testing

Objects	**# of bugs seeded**	**# of bugs detected**			
		Object	**Component Integration**	**System integration**	**Total**
BankController	17	16	No testing	5	17
Message Component	8	7	No testing	8	8
WithdrawAgent	12	11	10	7	12
CheckBalance Agent	10	10	9	6	10
UserAccounts Manager	8	8	4	2	8

Table 3: Test result of the testing of the bank subsystem

- ## Reuse test cases

We also made changes to check whether our approach can successfully detect seeded bugs without repeating all tests (Table 4). Tests for all these cases are successful in detecting all bugs. The result tells us that the approach that uses specification and generation techniques with the help of ripple effect analysis is efficient in test case reuse and test redundancy reduction.

	Change1	Change2	Change3	Change4	Change5	Change6
Object changed	FileAccount	FileAccount	Withdraw Agent	Withdraw Agent	Withdraw Agent	Bank Controller
Object specification	No	Yes	No	Yes	Yes	No
Component interface	No	No	No	No	Yes	No
Component external view	No	No	No	No	No	No
Inner object	Yes	Yes	No	No	No	No
Integration testing by vendor	No	Regenerate test cases	Rerun	Regenerate test cases	Regenerate test cases	
Unit testing by vendor	Rerun Test cases	Regenerate test cases	Rerun	Regenerate test cases	Regenerate test cases	
Integration testing by user	No	No	No	No	Regenerate test case	Yes
Unit testing by user	No	No	Rerun	Rerun	Regenerate test cases	Yes

Table 4: Changes made to illustrate the test case reuse

 ©Copyright 2000. All rights reserved by authors

10. Conclusion

This paper discusses issues that arise in the component-based system testing which can be viewed from both the component vendor's and the component user's point. Test case redundancy and reuse is a difficult problem for integration testing with components. MfSS+, which is used to specify component and application, is presented in this paper as an approach to solve these issues. From the specifications, test scenarios can be generated both for unit testing and for integration testing of components as well as their applications. In addition to traditional test techniques, MfSS+ view, the hierarchical organization of test scenario templates and the ripple effect analysis can all facilitate test case reuse and significantly reduce test redundancy. The bank subsystem of ITSS was chosen as an example. The whole test process detected out all seeded bugs which shows that the approach presented in this paper is practical and efficient.

Acknowledgement

This project is supported by OASD Y2K Office through Army Research Laboratory.

References

[Brown 1996] A.W. Brown and K.C. Wallnau, "Engineering of component-based systems", In A. W. Brown, editor, *Component-based Software Engineering*, pages 7-15, IEEE Press, 1996.

[Buy 1998] U.Buy, A. Orso and M. Valsasna, "A framework for testing object-oriented components", Technical report, Politecnoco di Milano, Milan, Italy, December 1998.

[Cho 1998] I. Cho, "Testing Components Using Protocols", OOPSLA'98 Doctoral Symposium.

[D'Souza 1999] D. D'souza and A. Wills, *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Addison Wesley Longman, 1999.

[Harroald 1999] M. J. Harroald, D. Liang and S. Sinha, "An approach to analyzing and testing component-based systems", the first international ICSE workshop, Los Angeles, California, USA, 17 May 1999, P134-140.

[Jorgensen 1994] P. Jorgensen and C. Erickson, "Object-oriented integration testing", Communications of the ACM, 37(9):30-38, September 1994.

[Kirani 1994] S. H. Kirani and W.T. Tsai, "Specification and Verification of Object-Oriented Programs", Technical Report, Department of Computer Science, University of Minnesota, December, 1994.

[McGregor 1997] J. D. McGregor, "Parallel Architecture for Component Testing", Journal of Object-Oriented Programming, May 1997.

[Sohn 1999] H.W. Sohn, D. C. Kung, P. Hsia, "CORBA Components Testing with Perception-based State Behavior", COMPSAC'99, Phoenix, AZ, October 27-29, 1999.

[Szyperski 1997] C. Szyperski, *Component Oriented Programming*, Addison-Wesley, 1997.

[Tsai 1999] W. T. Tsai, Y. Tu, W. Shao and E. Ebner, "Testing Extensible Design Patterns in Object-Oriented Frameworks through Hierarchical Scenario Templates", IEEE COMPSAC, 1999.

[Tu 1999] Y. Tu, W.T. Tsai, and Y. Xiang, "Specifying Data Constraints in MfSS", 6th Asia-Pacific Software Engineering Conference, Dec.7-10, 1999, Takamatsu, Japan.

[Vishnu 1996] R. Vishnuvajjala, W. T. Tsai, R. Mojdeh and L. Elliot, "Specifying Timing Constraints in Real-Time Object-Oriented Systems", Proc. of High-Assurance System Engineering, 1996, pp. 32-40.

[Wang 1996] Y. Wang, W. T. Tsai, X. P. Chen and S. Rayadurgam, "The Role of Program Slicing in Ripple Effect Analysis", in Proc. of Software Engineering and Knowledge Engineering, 1996, pp. 369-376.

[Wang 1997] Y. Wang, R. Vishnuvajjala and W. T. Tsai, "Sequence Specification for Concurrent Object-Oriented Applications", Proc. of WORDS, 1997.

 ©Copyright 2000. All rights reserved by authors

QW2000 Standby Paper SB3

Mr. Ron Silacci
(Lucent Technologies Inc.)

Principles of Multi-System Integration

BACK TO QW2000 PROGRAM

Lucent Technologies
Bell Labs Innovations

Principles of Multi System Integration

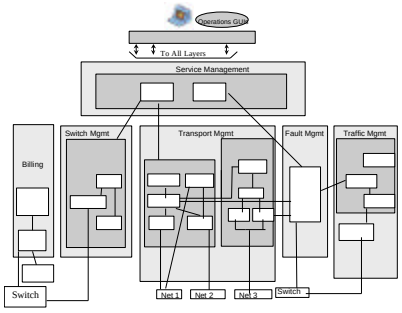
R. C. Silacci

June, 2000

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

Architecture



Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

Integration Lab

- Make it a Separate Environment
- As Close to Production As Possible
- Apply Configuration Management
- Control the Remote Access
- Communication Links Needed?
- Must Have an Entrance & Exit Criteria

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

Testing Phases

- Unit Test
- Pairwise Test
- System Test
- Interface Test
- Integration Test
- Customer Acceptance Test
- Installation Test

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

Organization

- Release/Project Management
- Requirements
- Architecture
- Integration Test
- Training & Documentation
- Deployment
- Customer Technical Support

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

Change Control

- Document Process and Forms
- Cant Install Without It!
- Make It Part of the Entrance Criteria
- Use the Same Process for Both the Integration Lab and the Production Site

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

User Interface

- Make It Appear to the User As One Big System
- Common Look and Feel
- Front End System to All Systems or
- All Systems Follow GUI Standard
- Ability to Launch Applications
- Systems Must Co-display
- Support PC and Work Station

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

System Interfaces

- 90% of the Bugs
- Have Interface Agreements Between Systems
- If Possible Use Standard Interface Tools (CORBA)
- If Changes to the Interface, Both Systems Must Change
- Watch Out for the Data Dependencies and
- Data Synchronization

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

OAM&M Strategy

- Operations, Administration, Maintenance and Monitoring
- Take Data Center Operations Perspective
- Startup & Shutdown
- Backup/Recovery
- High Availability
- Hardware/Software Maintenance Contracts
- Hardware and Application Monitoring

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

Performance/Load/Stability Test

- A Must Do!
- Need Performance Requirements
- Individual Systems Must Do Their Own
- Integration Team Performance Tests Areas of Vulnerability
- Run a 48 Hour or Longer Test
- Problems
  - Systems Slow Down
  - Systems Hang
  - Queues Overflow
  - CPU at 100%
  - Thrashing
  - Crashing
  - Timeouts

Copyright Lucent Technologies Inc. April 2, 2000

Lucent Technologies
Bell Labs Innovations

Conclusion

- **Common Hardware and Software Platforms**
- **Systems Use the Same DBMS**
- **GUI Standards and Development Tools**
- **Integrated OAM&M Strategy**
- **Implement Standard Interfaces**
- **Common Table Maintenance Strategy**

Copyright Lucent Technologies Inc. April 2, 2000

Principles of Multi-System Integration

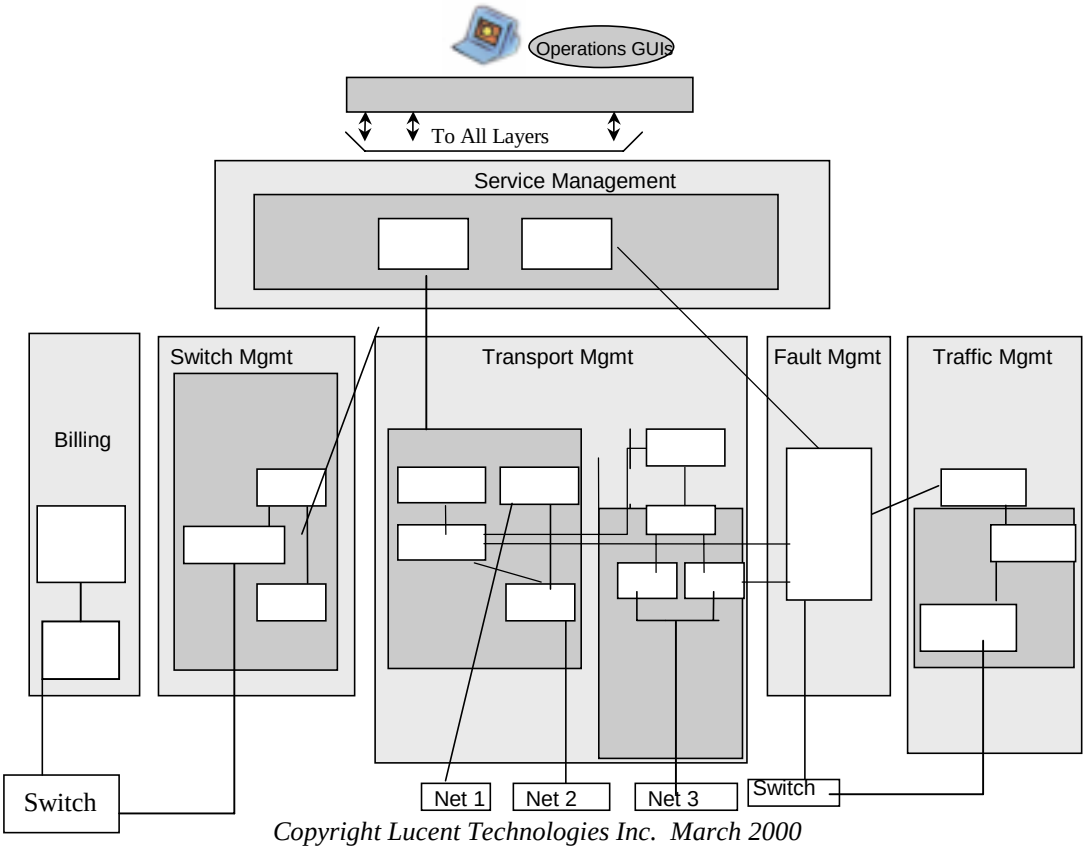
R. C. Silacci
Lucent Technologies Inc.

System integration is becoming a vital component of delivering a complete solution to customers. The current competitive environment requires individual system teams to work together with an Integrator in delivering a complete solution to the customer. In an effort to meet this challenge, more and more pressure is being put upon the systems integration effort. From practical experiences in integrating many telecommunication systems, System Integration Principles have emerged. This list contains the principles that will make integrating multiple systems successful and a testers work life easier.

The environment in Lucent consists of many system product teams that develop individual systems to meet specific customer needs. Examples are service provisioning systems, network trouble reporting systems, network management systems, and billing systems. In order to deliver a complete solution to customers, all of these types of systems have to work together in the following fashion:

- Provision a service on a network.
- Make a call using that service on the network.
- Bill the call usage on the network
- Manage the network.

This paper discusses the principles that have been used in recent integrated offerings to customers. Below is an example of an architecture diagram containing multiple systems that need to be integrated.



Copyright Lucent Technologies Inc. March 2000

1. Build an Integration Test Lab

This is obvious, but it takes preparation and planning to configure this. It should be a separate environment from development and resemble, as close as possible, the customer's production site. All systems make deliveries to this test lab. This is the place where, for the first time, all of the systems come together for an integration party. A Physical Architecture document drives the hardware and network that is needed. This lab has to be administered with expertise in Operating System Administration, Data Base Administration, system application administration, data center administration and network administration. There has to be close communication between the Integration test team and the Lab support team. In constructing this lab there are questions that need to be answered.

- Does the Customer site have a firewall?
- Does the customer hardware really need to be duplicated or can a "lite" version suffice?
- Are external communication links needed?
- Will it be used as a demo site?
- Will there be room to grow?
- Who pays for 3rd party software licenses

2. Develop Integration Test Processes

To manage a large multi-system integration, all system involved need a uniform test process to follow.
This test process should contain the participants, their roles and responsibilities and the test process flow.
All the phases of testing should be defined and explained. Pairwise Testing, Interface Testing and Integration Testing are examples of the phases of testing an integrated solution. On the next page is the definition of each testing phase.

Testing Phases

Test Phase	Definition
Unit Test	New development is tested by the developers, on a unit or module basis, to verify that each feature works as required. The purpose of Unit Test is to ensure that the code delivered to System Test is the highest quality possible and that the code, at the individual developer level, works as stipulated in the requirements.
Pairwise Test	Interfaces between systems are tested (developer to developer) to verify that there is connectivity and that basic messages flow .. Pairwise testing is conducted after Unit Test. Pairwise Test cases are typically written by the system that is originating the message.
System Test	The product is exercised to show that the individual features, now integrated, function as specified in the requirements, and that all requirements are implemented and complete. Simulators are used to simulate responses from other systems or external interfaces. System Test is done after Unit Test and can be done concurrently with Pairwise Test. However, System Test completion is dependent upon completion of Pairwise testing.
Interface Test	Interface Test, which is system to system testing can take place in parallel with System Test. Its purpose is to test out all the functionality of the two systems in regards to what is sent across the interface and to ensure that the interface requirements are met.
Integration Test	Integration testing is performed to validate the flow of data through multiple systems and to verify that customer requirements have been met across involved systems. Testing is performed on an "end-to-end" or "flow through" basis. The Integration Test environment is designed to simulate the production environment. Test scenarios are designed to cover production user scenarios. Included in Integration Test is the assessment of performance of systems against performance indicators/metrics. Examples might include transaction volumes, transaction process time, response time, etc. Integration Test is ideally conducted after Interface Test, but can overlap with it.

3. Organize for Integration

The following roles and responsibilities are needed:

- **Release/Project Management**
  An overall release manager must coordinate all of the systems involved in a release so that they march along to a common schedule. Individual schedules are gathered up from the system teams and a consolidated integrated schedule is built. Tracking of each system's critical milestones and weekly project meetings are

important to keep the entire project on schedule. The project manager must also be the liaison to the customer.

- **Customer Requirements Team**
  The customer requirements are gathered by this team and these requirements are distributed by the architect to the different systems that compose the integrated solution.

- **Architecture**

  Both logical and physical architecture have to be established. The logical architecture denotes the functions and the system interfaces. The physical architecture stipulates the hardware, network, software, and performance characteristics. The physical architecture can identify hardware and 3rd party software that can be shared. The architecture also determines the common user interface and ensures that all systems are compliant with this common set of standards. All systems have to follow the architecture.

- **Integration Test**
  This team plans the integration, turns the requirements/user scenarios into test cases and expected results and executes the tests. IT also writes trouble reports and works closely with the Integration Lab and system teams.

- **Training**
  The users have to be trained on all the systems that are in their workflow and they have to understand the data flow between the systems. They must also learn to populate tables with their particular set of values.

- **Deployment/Customer Technical Support**
  Deployment to the customer site is a major work effort. The following tasks are involved:
  - Site preparation.
  - Hardware and software provisioning.
  - Systems installation schedule and coordination.
  - Change Control and software media distribution.
  - Provide responsive Customer Technical support.

4. Develop and Implement Change Control Processes

Change control is the overall process of managing the changes to systems. It also takes into account when and where the changes are implemented in the Integration lab and the customer production site. It starts with a change control process. The same process should be used for both the Integration Test Lab and the Production site.

5. Develop to a Common User Interface

Something has to glue all of these systems together so that the user sees it as one large system.
A single window into all the systems in a particular work flow needs to be provided so that one user can view multiple systems at once on their terminals. There either needs to be a system that front ends all of the other systems involved or all systems have to code to the same GUI standard using the same GUI development tools. Both solutions have to be able to support multiple system windows being displayed on workstations and/or PC's.

6. Implement Standard System Interfaces

It is difficult to integrate multiple system interfaces when the interfaces are of different types. Types means the protocol and message format used in sending transactions between systems. If all system interface using a common language/tool, such as CORBA, integration is much easier.

7. Have an Integrated Operation, Administration, Maintenance and Monitoring (OAM&M) strategy

From a data center operations standpoint, it would be nice if all of the systems were of the same hardware platform using the same backup and restore procedures. However, with different hardware platforms, different data base managers and different Operating Systems, an integrated OAM&M is difficult. A common strategy for system backups and restores is the ideal solution. A common way to monitor all of these systems is also desirable.

8. Execute a Performance /Load /Stability Test.

Functionally a set of systems may look like they are working fine together. Put the systems under a load test and behold what gets uncovered. In order to execute a performance test, one must have performance requirements from the customer. How many transactions per hour must the systems process? What is the peak volume and hour? What is the expected GUI response time? How long can one transaction take to process? These are just a few questions that need to be answered. Each system involved in the flow of this testing must perform an individual performance test to determine the limits of their system and must meet these performance requirements by themselves. A simulator or driver to generate transactions at different rates is also required. Note that the systems have to have some logging or tracing mechanism so that time stamps of transactions can be used later for analysis. Then run the test and watch what happens. One of several things will happen.

1. Flow through performance slows down depending on load.
2. A system will hang
3. A Queue will overflow
4. A timeout will occur

5. A Core dump will occur
6. The CPU needle will get pinned on one or several systems
7. A system will thrash
8. After a certain amount of time performance will slow to a crawl.
9. Everything will work fine

As you can see, more things may go wrong than right.

Conclusion

If we had to do it all over again, the following recommendations would be made:

- All systems use a common hardware platform with the same version of the OS.
- All systems use the same DBMS.
- All systems implement the GUI the same way (Java for example)
- All systems implement the same OAM&M strategy.
- All systems implement the same hardware and software monitoring systems.
- All systems implement a standard interface between them. (CORBA for example).

QW2000 Standby Paper SB4

Dr. Giora Ben-Yaacov
(Synopsys Inc.)

Effective Software Testing at a Leading Electronic Design Automation (EDA) Company

BACK TO QW2000 PROGRAM

Presentation Abstract

This paper is based on the author experiences with "real world" deployment of software testing processes and methods at a large high-tech software company that produce very complex software products that require rapid innovation changes and time-to-market constraints. The testing techniques and methods presented require true "testing partnerships" among software developers, testing and application engineers, and customers. The paper also covers subjects such as "six steps for developing defect-free code", "goals and processes for effective software testing" and a descripttion of the "software testing life-cycle".

About the Author

Giora's 25 years of experience in software quality improvement programs, software testing, software development management, program management, and customer support include a spectrum of results ranging from that of individual contributor, through project leader, program manager, maintenance & training coordinator, instructor, consultant, and, since August 1998, a quality specialist with Synopsys Inc,. At Synopsys, Giora holds responsibility for leadership and implementation of software quality & business improvements programs, and he is working closely with the various software development and testing communities at Synopsys striving to achieve performance excellence.

Dr Giora Ben-Yaacov is a senior member of IEEE and a member of the American Society for Quality. He has served for 6 years on the editorial board of the IEEE CAP journal, has authored 30 technical papers, and made presentations at various conferences and industry meetings.

BACK TO QW2000 PROGRAM

SYNOPSYS

Effective Software Testing at a Leading Electronic Design Automation (EDA) Company

Giora Ben-Yaacov

Corporate Quality Dept, Synopsys, Inc

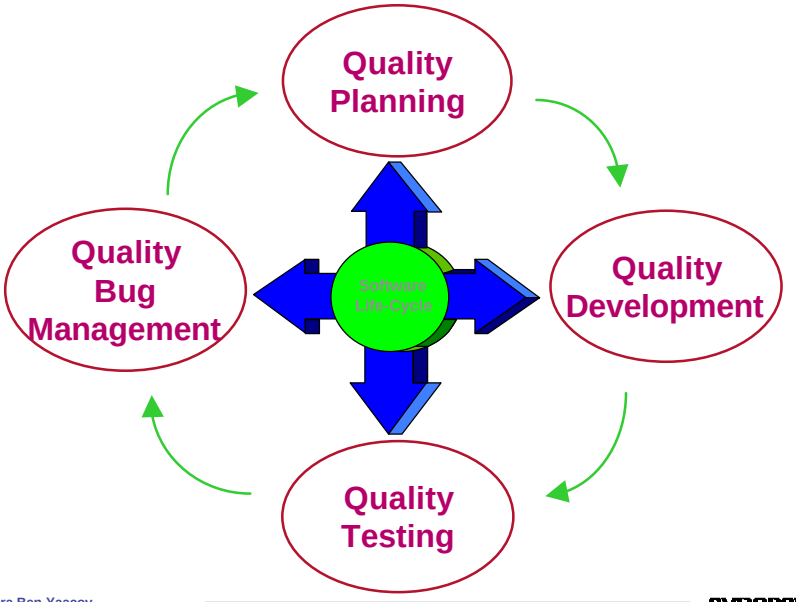
giora@synopsys.com

What is Software Quality?

© 1997 Synopsys, Inc.
Synopsys Confidential

SYNOPSYS

Synopsys Roadmap for Process Improvement



Giora Ben-Yaacov

SYNOPSYS®

Software Defects

Product Development		Post-Release
R&D Development (Industry average: 40%-60%)	Alpha/Beta Testing (Industry average: 30%-40%)	Customer Use (Industry average: 5%-10%)

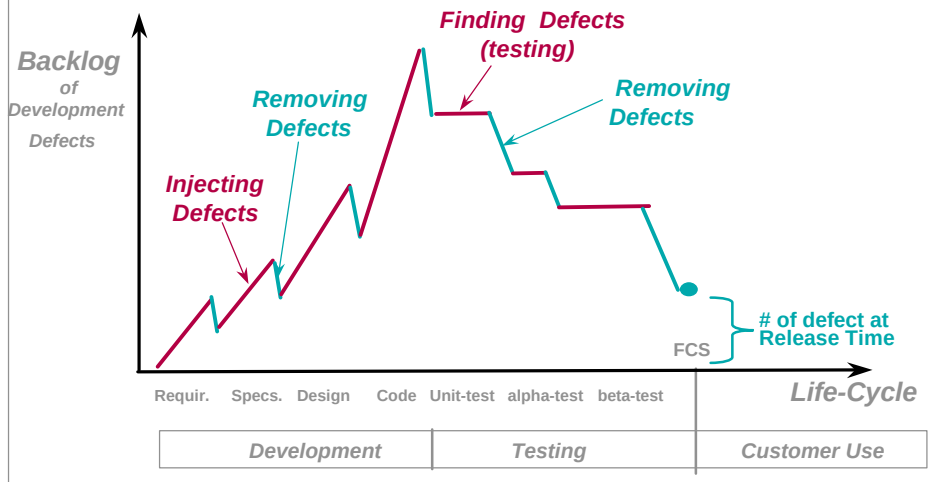
- **R&D Development**
  Goal: Build QUALITY into the product (Prevent & Remove development defects)
- **Alpha/Beta Testing**
  Goal: Improve QUALITY by Testing (Find MORE defects, fix MORE before release)
- **Customer Defects (after Release)**
  Goal: Improve RESPONSE TIME, Reduce BACKLOG of opened defects, and Improve COMMUNICATION (R&D-CAE -field-customers)

Giora Ben-Yaacov

SYNOPSYS®

SYNOPSYS

Goal: Finding and Eliminating Software Defects During the Development and Testing Phases



Defect Detection Rate: (Software Industry Average)	30%-70%	20%-50%	5%-10%

Giora Ben-Yaacov

SYNOPSYS®

Synopsys Six-Steps Process for Developing "Defect-Free" Code

- **Step 1: Commit to deliver defect-free code**

  Making a firm commitment by developers to delivering defect-code free in spite of schedule and other pressures is absolutely necessary (additional benefit - you will also see real improvements in schedules)

- **Step 2: Specifications and Design for simplicity and reliability**

  A clean and well structured functional and design simplifies the development of reliable code. A poor functional and design specs make it impossible to achieve defect-free code. Specifications and design reviews are important to ensure correctness and simplicity.

- **Step 3: Review code** (typically 2 engineers: developer+senior, or developer+manager)

  Code reviews have consistently been shown to be the single most cost-effective way of removing bugs from code. The process of showing and explaining a new section of code to another engineer has several positive impacts: (a) confirms that the design is functioning as intended, exposes inefficient code, shows code that is working for a wrong reason, etc.; (b) forces the engineer to articulate assumptions -- many times the developer himself/herself suddenly realized that he/she had made an invalid assumption; © encourages cross-training and sharing of techniques

- **Step 4: Create regression test suite**

  The most effective testing for delivering defect-free code that we use at Synopsys is to create a fully automated regression test suite that are run after each build of the software. The tests are designed to exercise every part of the software and produce a success/failure report. For each new release, the regression test suite is modified and expanded with new test cases that cover new functionality, bug fixes or any other new capability that were added to the new release.

- **Step 5: Build and test daily**

  Daily builds and running the regression test suite after every build give developers quick feedback about the changes they are making. It also gives management clear and objective feedback about the project status.

- **Step 6: Use automated tools to check the code**

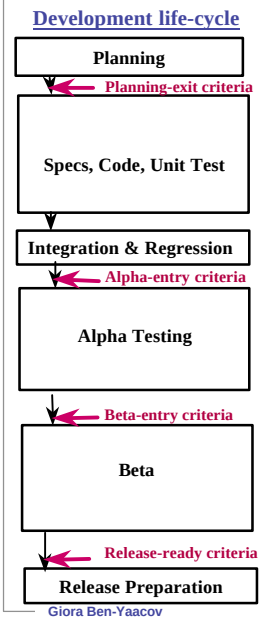
  Automated tools are used in addition to clean design and good coding practices. Using automated checking tools alone will never turn poorly designed and buggy code into defect-free code. However, the automated checking tools find a lot of bugs that would otherwise take a lot of testing time and effort to find and fix.

Giora Ben-Yaacov

SYNOPSYS®

SYNOPSYS

Synopsys Test Process



Test Activities

-- Quality plan

-- R&D prepare and execute new test cases (new unit tests, new designs, …)
-- R&D add tests to regression suite and run regression
-- CAE prepare Test plan for each project (meet specs for new functionality)
-- CAE create test cases per test plan (for the alpha testing)
-- CAE review Level I, II, and III QoR Test Suites (and add/modify, when needed)
-- CAE prepare plan, resources & schedule for Alpha testing

-- R&D (with Ops) deliver integrated, stable and executable code on all platforms
-- R&D regression suite >95% pass

-- CAE testing (functional test per test plan, QoR test (Level I,II,III)
Duration 2 weeks (1 week CAE concentrated testing, 1 week additional testing)
-- R&D fix bugs
-- CAE+Mkt prepare plan for beta, sign-up customers, obtain ACs commitment, ..

-- Mkt coordinate customer beta testing
-- CAE coordinate AC testing (2-5 days concentrated testing plus additional testing)
-- R&D fix bugs

-- Production-Ready testing

-- Post Mortem

Giora Ben-Yaacov

SYNOPSYS®

Regression Test Suites

- **Quality Regression Test Suite**
  - Owner: R&D
  - Nightly
- **QoR Regression Suite** (performance improvements from release to release)
  - Owner: CAE
  - Once or more per release
- **Customer Designs Regression Suite(s)**
  - Owner(s): CAE and/or R&D and/or Flow Test
  - Once or more per release

Giora Ben-Yaacov

SYNOPSYS®

SYNOPSYS

Other Testing

- Testing in customer environment (independent computer environment, installation, licensing, … …)
- GUI testing
- Stress/boundary testing
- Flow testing
- Documentation
- Release-ready testing
- … … … .

Giora Ben-Yaacov

SYNOPSYS®

Effective Software Testing at a Leading Electronic Design Automation (EDA) Company

By:
Giora Ben-Yaacov
Corporate Quality, Synopsys Inc.

Contact: Giora Ben-Yaacov, Ph.D., Synopsys Inc., (650)584-1410, giora@synopsys.com

==========

Customers of Electronic Data Automation (EDA) companies such as Intel, IBM, HP, Siemens, or Motorola often ask "what does The EDA industry do to improve the quality of their software products and of their responsiveness to customer quality issues"? This paper is intend to describe one important element of the overall quality improvement effort that had been deployed at a leading market-driven EDA company (Synopsys Inc.) -- The deployment of effective software testing techniques and methods that represent real-world customer issues. The testing techniques and methods presented here require true "testing partnerships" among software developers, testing and application engineers, and customers.

The paper outline below includes short descriptions and figures for the subject areas that will be covered in detail in the full paper and in the presentation materials:

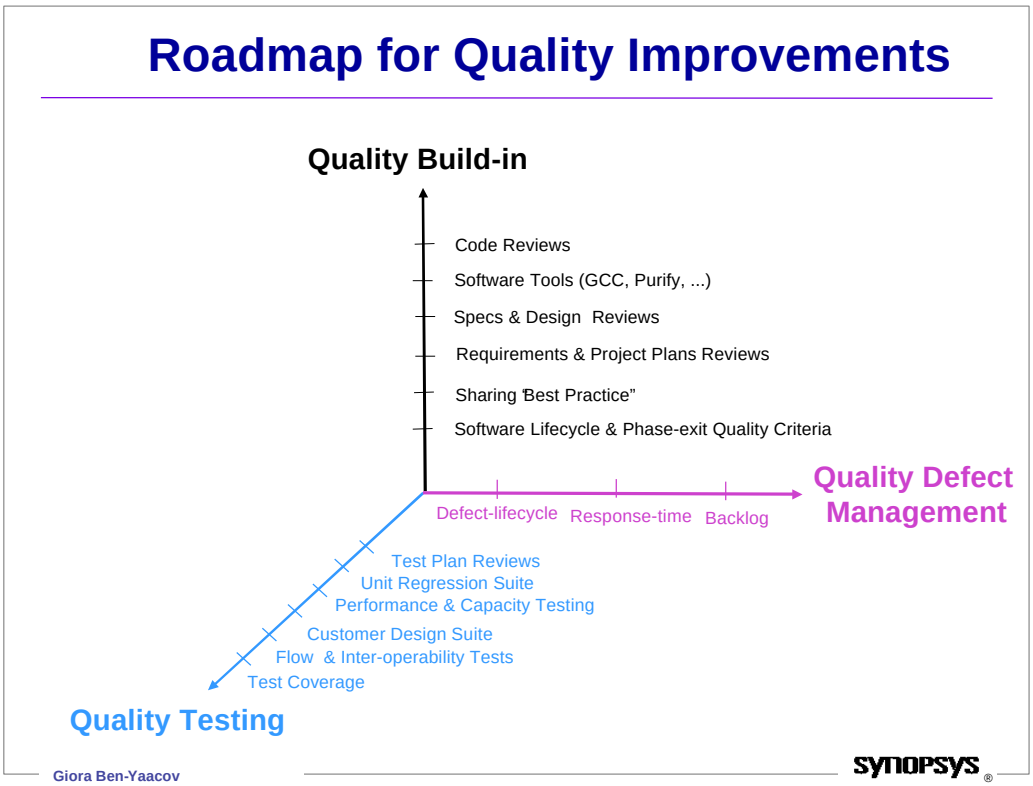
(1) About Synopsys and its commitment to Quality

(a) Synopsys, Inc. provides comprehensive software tools and technology for the product development requirements of the world's leading electronics companies. Synopsys is the largest supplier of software tools used to accelerate and manage the design of semiconductors, computer systems, networking and telecommunications equipment, consumer electronics, and a variety of other electronic-based products. With nearly 3,000 employees and 1999 annual sales of over $800 million, Synopsys has software development and research facilities around the world. The company is headquartered in Mountain view, California and traded on Nasdaq under the symbol SNPS. More information about the company, its products and services may be obtained at www.synopsys.com.

(b) In a recent EE Times survey, Synopsys was ranked as the top leader in EDA. In order to stay on top in this highly competitive market, Synopsys must not only lead in technology, it must also make a continuing commitment to quality improvement. As the company Chairman and CEO noted in his recent message to all employees: "In the eyes of most customers, we are viewed as the most reliable, highest quality, most technically savvy EDA company in the world. This is precisely the brand that we intend to perpetuate"

© Quality management concepts have been around for many years - under various guises and with varying degrees of popularity. Since the business success of Synopsys' customers is largely driven by technology, time-to-market, and quality -- improving our quality systems and processes is essential for our own competitive strength. Moreover, as competition among EDA tool vendors increases, quality becomes the distinguishing factor. Customers have indicated that many EDA products today offer similar technical capability and functionality. Therefore, selection is often based upon the perceived quality of the vendor organization.

(2) Synopsys Roadmap for Quality Improvements



(3) Synopsys six steps for developing "defect free" code

Synopsys Six-Steps Process for
Delivering "Defect-Free" Code

- **Step 1: Commit to deliver defect-free code**

  Making a firm commitment by developers to delivering defect-code free in spite of schedule and other pressures is absolutely necessary (additional benefit - you will also see real improvements in schedules)

- **Step 2: Specifications and Design for simplicity and reliability**

  A clean and well structured functional and design simplifies the development of reliable code. A poor functional and design specs make it impossible to achieve defect-free code. Specifications and design reviews are important to ensure correctness and simplicity.

- **Step 3: Review code** (typically 2 engineers: developer+senior, or developer+manager)

  Code reviews have consistently been shown to be the single most cost-effective way of removing bugs from code. The process of showing and explaining a new section of code to another engineer has several positive impacts: (a) confirms that the design is functioning as intended, exposes inefficient code, shows code that is working for a wrong reason, etc.; (b) forces the engineer to articulate assumptions -- many times the developer himself/herself suddenly realized that he/she had made an invalid assumption; © encourages cross-training and sharing of techniques

- **Step 4: Create regression test suite**

  The most effective testing for delivering defect-free code that we use at Synopsys is to create a fully automated regression test suite that are run after each build of the software. The tests are designed to exercise every part of the software and produce a success/failure report. For each new release, the regression test suite is modified and expanded with new test cases that cover new functionality, bug fixes, any other new capability that were added to the new release.

- **Step 5: Build and test daily**

  Daily builds and running the regression test suite after every build give developers quick feedback about the changes they are making. It also gives management clear and objective feedback about the project status.

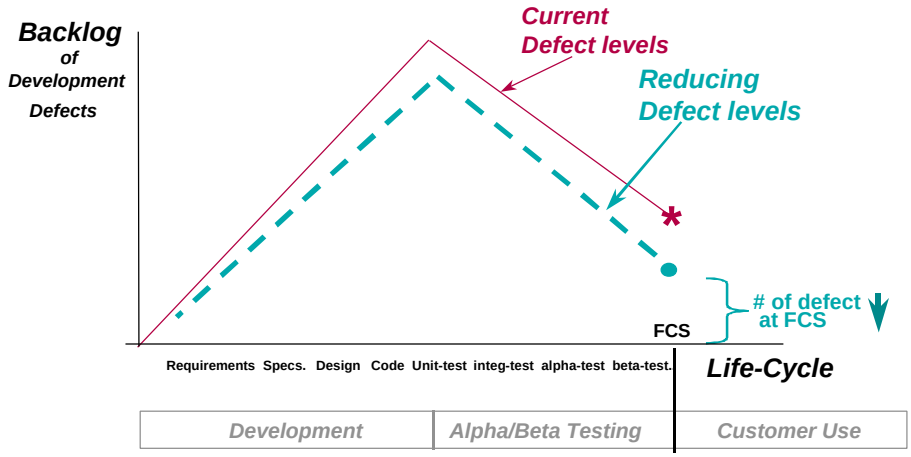
- **Step 6: Use automated tools to check the code**

  Automated tools are used in addition to clean design and good coding practices. Using automated checking tools alone will never turn poorly designed and buggy code into defect-free code. However, the automated checking tools find a lot of bugs that would otherwise take a lot of testing time and effort to find and fix.

Giora Ben-Yaacov SYNOPSYS®

(4) About goals of the testing processes

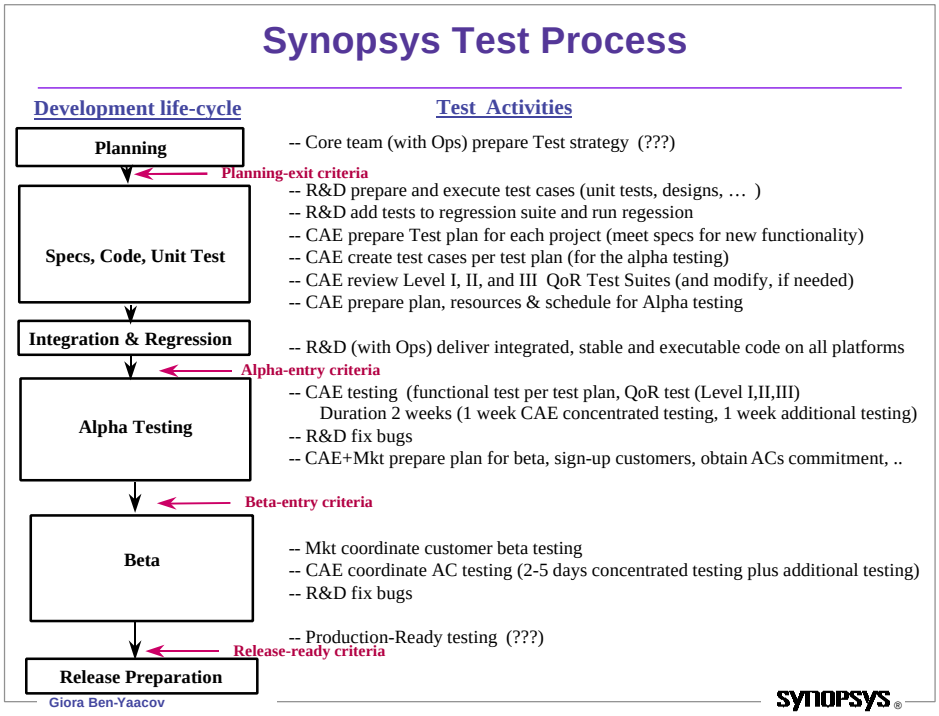
Testing Goal: Reducing Software Development Defects



Defect Detection Rate: (Software Industry Average)	40%-60%	30%-40%	5%-10%

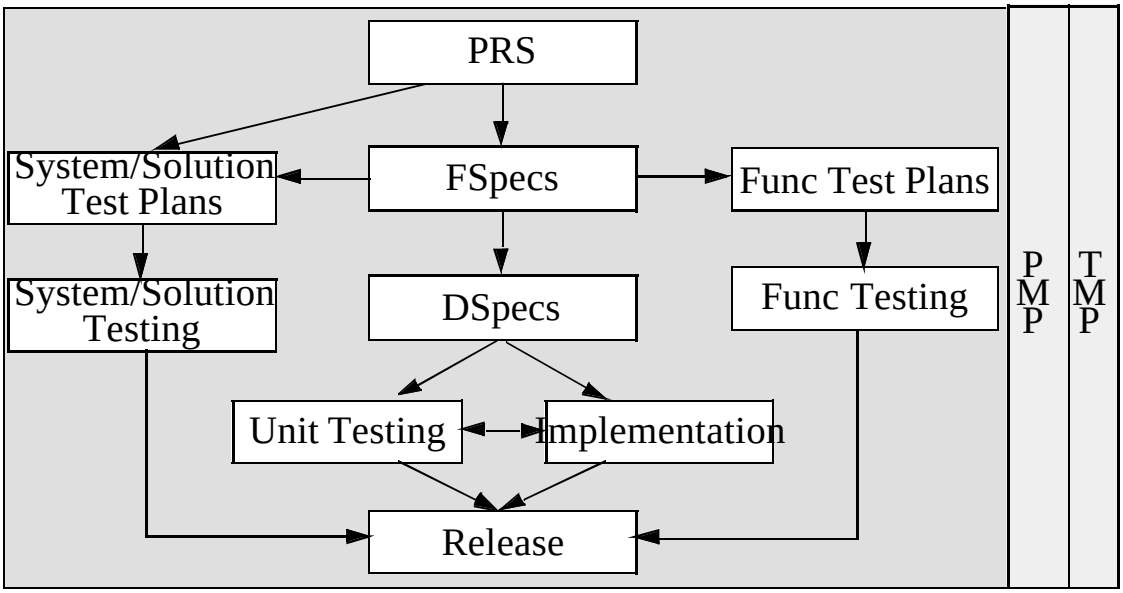
Giora Ben-Yaacov SYNOPSYS®

(5) About the testing process life cycle



Our software life-cycle activities as applied to software testing is illustrated in the figure below. It encompasses the following software development tasks:

- Product Requirement Specifications
- Functional Specifications
- Design Specifications
- Implementation
- Unit Testing
- Functional/Feature Testing
- Systems/Solution Flow Testing
- Documentation (Tech Publications)
- Release
- Post-Mortem Analysis



(6) About Different Categories of Testing

The various types of testing performed during the release cycle include Feature Test Plans, which cover feature testing and its regressions, and System/Solution Test Plans, which cover all the other types of testing.

Typically the defects uncovered through validation are reported via "Synopsys Technical Action Requests" (STAR) and are tracked during the release cycle. The STARs are used as metrics, but more importantly, they record critical problems that must be fixed before the product is released. These STARs also provide data for performing root-cause and post-mortem analyses. The Defect Tracking and Help Desk Management system are used by Customer Support, Software Engineering, Electronic Publishing, Applications Engineers and Software Developers. Any Synopsys employee may enter a STAR defect or enhancement request.

The Table in the next page shows the main different categories of testing that are performed by the developers and the application engineers:

Action	Description
Unit Testing	Unit testing is usually at the code level, where the developer wants to test the C-function(s) created. Some of these tests may be added to the regression.
Feature Testing	Also referred as function testing, this generally focuses on the product features. Ideally, a test should be developed for every feature in the product. However, in practice, a number of small features will undergo a single test. The development teams are expected to do all feature-level testing. The R&D team can negotiate with the testing team to do some or all feature testing. These tests are typically added to a feature regression suite. One of the criteria for product release is to ensure that 100% of feature regression passes.
Extended Feature	Similar to feature testing, but a few features are used together in logical order, representing a typical customer task. If, for example, a simulator is being checked, the feature testing may look at the ability to set a breakpoint, while the extended feature testing might use the ability to set breakpoints while debugging. All extended feature tests are automated and added to the regression.
Flow Testing	Flow testing addresses product-to-product interactions and is based on information from customer use models and flow documents. Flow testing tries to verify that the customer can pass data from one tool to another in the given use model. It would be ideal to automate these tests; however, operational/tool issues usually prevent this. As a result, flow tests remain mostly manual in the regression.
Regression Testing	As new test cases are created, they are automated and added to regression so that they can be executed for every release. This ensures that adding features or fixing bugs has not interfered with the rest of the product. The regressions can include any type of testing.
Performance Testing	The product performance is compared against the previous versions and against the stated goals of performance improvements. Performance measurements include speed in completing the task, memory consumption (swap), and accuracy of results.
Stress Testing	Stress testing (also known as capacity testing), subjects the product against known boundary conditions or large customer designs with the objective of characterizing the product stress levels.

ABOUT THE AUTHOR

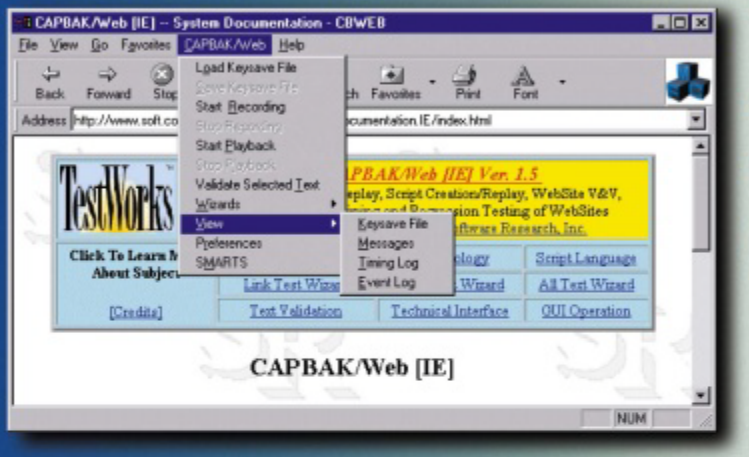
Giora Ben-Yaacov, Ph.D. (giora@synopsys.com)
Giora's 25 years of experience in software quality improvement programs, software testing, software development management, program management, and customer support include a spectrum of results ranging from that of individual contributor, through project leader, program manager, maintenance & training coordinator, instructor, consultant, and, since August 1998, a quality specialist with Synopsys Inc,. At Synopsys, Giora holds responsibility for leadership and implementation of software quality & business improvements programs, and he is working closely with the various software development and testing communities at Synopsys striving to achieve performance excellence. In the two and a half years prior to his tenure at Synopsys, Giora was a Quality Architect at Cadence Design Systems, Inc. At Cadence, Giora held responsibility for implementation of software development processes, software verification & validation methodology, and deployment of industry quality standards such as ISO 9000 and the Software Engineering Institute (SEI) assessment methodology.

Dr Giora Ben-Yaacov is a senior member of IEEE and a member of the American Society for Quality. He has served for 6 years on the editorial board of the IEEE CAP journal, has authored 30 technical papers, and made presentations at various conferences and industry meetings.

Make your website 100% reliable

and your e-business 100% successful!

— automatically



You can't afford

a broken website and

frustrated customers.

Find out before they do.

No miracles.
Just solid solutions.

Quality Testing of Websites:

Recorded & Scripted Playback

100% Object Mode + TrueTime

Fully Synchronized Playback

Content Validation

Helpful Test Wizards

SMARTS™ Interface

Site Verification

SR

Software Research, Inc. We know how.

901 Minnesota Street, San Francisco, CA 94107

www.soft.com (800) 942-SOFT info@soft.com

eValid ™ -- The Internet Quality Authority ™

Client-Side Browser-Based WebSite Quality Checking,
Testing, Validation, Tuning, Loading, 24x7 Monitoring
Training, Consulting, Seminars

© Copyright 2000 by eValid, Inc.

eValid -- The Internet Quality Authority

eValid enhances your e-business success by assuring that your WebSite is trouble-free, reliable, speedy, and available 24x7. In a Web-paced world your WebSite is your key asset. eValid checks, protects and insures.

eValid Products

eValid's Test Enabled Web Browser™ is a test engine that provides you with browser based 100% client side quality checking, dynamic testing, content validation, page performance tuning, and webserver loading and capacity analysis.

This new cutting-edge technology 100% client side based and is completely object-oriented. eValid offers a unified approach to WebSite testing that is unique in its simplicity, power, efficiency, effectiveness, and superior ease of use.

By focusing entirely on the users' view of WebSite quality, eValid results are accurate, complete, repeatable, and highly effective -- all as experienced by your users. The eValid test engine is available in several product configurations.

- **Testing:** eValid test scripts can exercise the key parts of your site, confirm links, check content, and simulate users' activities. *Make sure your customers get the right message!*
- **Validation:** eValid can confirm selected content, validate document properties, images and applets. *Have confidence that you are delivering correct information!*
- **Tuning:** eValid timing capabilities let you identify slow-loading pages so you can "tune up" your site for optimum performance. *Keep customers from clicking away!*
- **Loading:** eValid load testing scenarios can simulate 100's or 1000's of users. *Can your WebSite handle the traffic when a serious crunch comes?*

eValid Services

eValid website quality services are all based on the eValid test engine, and are are supported through training, consulting, and technical seminars.

- **Standard Monitoring:** eValid monitoring, based on the eValid test engine, runs standard tests on your site. eValid's 24x7 website performance monitoring provides for email and/or pager/beeper alert service, plus customer access on our WebSite to historic testing and monitoring data. *Be the first to know whenever your site is misbehaving.*
- **Custom Monitoring:** Use eValid test services to contract us to run tests you have recorded and proved out yourself using the standard eValid test engine. Custom eValid test executions run on standard intervals, in varying time zones, and are all 24x7. *Make sure your **own** tests run successfully all the time.*
- **WebSite Testing, Qualification, Verification, Loading:** eValid consulting services include WebSite testing, test suite development, WebSite qualification, e-commerce verification, and WebSite loading and capacity checking exercises. All work is based on application of the eValid test engine plus other non-released WebSite analysis facilities.
- **WebSite Quality Consulting & Seminars:** eValid website quality experts can work along side your web developers to make sure your site meets the highest reliability, quality, performance, and capacity standards. eValid seminars and workshops are aimed at bring your own team up to speed.

eValid -- Your E-Business Partner

eValid -- offering products and custom services -- is your one stop solution provider for WebSite quality. eValid is your true e-business partner.

eValid, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Phone [+1] 415.550.3020
FAX [+1] 415.550.3030
evalid@soft.com.

Are you a good enough developer to be a Microsoft Tester?

microsoft.com/**jobs**
/test.htm

Microsoft®
Where do you want to go today?®

© 2000 Microsoft Corporation. All rights reserved. Microsoft and *Where do you want to go today?* are registered trademarks of Microsoft Corporation in the United States and/or other countries. Microsoft is an equal opportunity employer and supports workplace diversity.

MICROSOFT CORPORATION

The reason so many people from various disciplines come to Microsoft is because we provide the most exciting challenges in the industry.

Since our inception in 1975, Microsoft's mission has been to create software for the personal computer that empowers and enriches people in the workplace, at school and at home. Microsoft's early vision of a computer on every desk and in every home is coupled today with a strong commitment to Internet-related technologies that expand the power and reach of the PC and its users. You can help shape the industry in nearly countless ways, all while gaining invaluable experience. Our casual environment lends itself to free thinking and therefore, creative problem-solving. However, the coolest part of all is the diversity of opportunity. You're guaranteed not to get bored.

http://www.microsoft.com/jobs/test.htm

WARE LANTE IBM ECAMPUS.COM CALICO COMMERCE HEWLETT-PACKARD RECRUITSOFT.COM ROCKWELL SOFTW
TWEB IXL **HEWLETT-PACKARD** EZENIA! **ERICSSON** TRELLIX **ALCATEL** AUCTIONROVER.COM **LANTE** BOOKSONL
CASE PARTS **J.D. EDWARDS** COMMERCE ONE **ICG COMMUNICATIONS** FOOD.COM **DELOITTE CONSULTING** IXL SUN
OSYSTEMS TICKETWEB **TRELLIX** MICROSOFT **ECAMPUS.COM** PEOPLESOFT **ROCKWELL SOFTWARE** LANTE DELO
ULTING EZENIA! **PEOPLESOFT** J.D. EDWARDS LANTE **AUCTIONROVER.COM** ERICSSON ALCATEL **IBM** BOOKSONL

THE INTERNET DEPENDS ON SOFTWARE FROM COMPANIES LIKE THESE.

COMPANIES LIKE THESE DEPEND ON RATIONAL.

PUS.COM **NETWORK ASSOCIATES** EZENIA! **MICROSOFT** FOOD.COM **SUN MICROSYSTEMS** ALCATEL HEWLETT-PAC
SOFT **IXL** ERICSSON **FOOD.COM** SUN MICROSYSTEMS RECRUITSOFT.COM **CALICO COMMERCE** COMMERCE ONE
SOFTWARE **COMMERCE ONE** IBM **TICKETWEB** MICROSOFT **RECRUITSOFT.COM** TRELLIX **EZENIA!** J.D. EDWARDS
TTE NSULTING **CASE PARTS** AUCTIONROVER.COM **BOOKSONLINE.COM** CASE PARTS **CONTEXT INTEGRATION** ECAM

Companies creating software in the electronic, Internet-driven economy are faced with a modern e-software paradox: how do you create software faster – in Internet time – and achieve the high quality you need to satisfy customers? The answer is e-development from Rational® Software. Thousands of companies around the world increase their software development speed and quality by applying Rational's software engineering best practices, unified tools and services. That's why Rational has been the leader in multiple segments of the software development life-cycle management market for many years*. Want to build software faster and better? Visit us at www.rational.com/success.

* IDC 1999 Report: Development Life Cycle Management

© 2000 Rational Software Corporation. All trademarks are the property of their respective owners.

Rational®
the e-development company™

RATIONAL SOFTWARE

Rational Software, the e-development company, helps organizations develop and deploy software for the Internet through a combination of tools, services and software engineering best practices. Rational's e-development solution helps organizations overcome the e-software paradox by accelerating time to market while improving quality. In 1999, International Data Corporation recognized Rational as the leader in multiple segments of the software development life cycle management market.
www.rational.com

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Web: www.rational.com

NTS-XXCAL

NTS-XXCAL is the oldest and largest testing laboratory of it's kind, having tested Software & Hardware since 1982, Environmental testing since 1961. Lab locations are in Los Angeles, London and Japan. NTS IS AN "NRTL" CERTIFIED (NATIONALLY RECOGNIZED TEST LAB) - A US GOVERNMENT RECOGNIZED LAB. NTS-XXCAL tests Software, Hardware & Peripherals. Decrease your time-to-market! We help you validate the integrity of your product by covering your target market thus leaving your valuable development resources free to work on your new products. This makes our testing service extremely cost effective!
www.ntsxxcal.com

ZD LABS

ZD Labs (www.zdlabs.com) leads the industry in Internet and technology testing. Building on Ziff-Davis Publishing's history of leadership in product reviews and benchmark development, ZD Labs brings independent testing, research, development, and analysis directly to publications, Web sites, vendors, and large IT organizations everywhere.
www.zdlabs.com

Vanteon e-Quality Solutions

Vanteon e-Quality and Test solutions are as creative as the software you write, as innovative as the web site your business depends on.

For more than fifteen years, Vanteon has developed and refined the industry's most effective QA and testing methodologies, processes, documentation and tools to meet the business, market and quality requirements of our clients. As business and technology platforms constantly change, Vanteon continuously evolves its methodologies to keep pace with the rapid deployment of new web technologies, and hardware and software designs. At Vanteon, we not only implement QA and testing best practices for our internal product development, we create them for our clients as well.

Through its diverse experience, Vanteon has built a repository of knowledge accessible to companies looking for customized Quality Assurance and Testing solutions. Vanteon specializes in QA process consulting, test planning, test documentation development and test execution, including strategies and scripts for Automated Desktop (capture/playback) and Automated Web Testing (functional, load, stress, performance, benchmark). Compatibility testing is performed in our Configuration Test Labs. Vanteon solutions span eBusiness, wireless peripherals, desktop/consumer, networking, client/server, embedded and print/imaging. Our professionals are experts in a range of technologies, including Windows, Mac, UNIX, LINUX, Windows CE, VxWorks, QNX and PSOS.

Vanteon is the premier national provider of comprehensive engineering solutions that generate revenue for clients ranging from the Fortune 500 to hot.com start-ups. In our seven centers of engineering excellence, Vanteon has assembled one of the industry's most proficient integrated teams of engineering and consulting professionals in e-business, quality assurance, commercial software development, and hardware and embedded systems. For more information, call 1.800.266.5046 or visit www.vanteon.com.

Ziff Davis Labs (www.zdlabs.com) is the independent, for-hire testing service of Ziff Davis Publishing. At our facilities in Silicon Valley and North Carolina's Research Triangle area, we perform two distinct but related kinds of work:

- providing top-quality independent, for-hire **testing** of Internet and technology products, and
- developing and distributing for Ziff Davis Publishing its industry-standard **benchmark software**

We provide the trusted, independent testing services you need to move at top speed in the Internet economy. Whether you want us to test your Web site, your desktop or server system, or any other Internet or technology product, we have the expertise, experience, and equipment to do the job.

We live and work at the heart of the Internet economy, testing for companies large and small, established and up-and-coming. We'll put your product--and, if you wish, your competitor's--through its paces. Our experts will give you an objective assessment and warn you about any bugs or potential problems. We'll examine your product from a fresh perspective, go as deeply into the details of its operation as necessary, and put our findings in a meaningful, action-oriented context.

If you'd like to keep our findings to yourself, we'll provide strict confidentiality. If you want to publicize the results, our marketing team can help you leverage our testing and analysis efforts to your best competitive advantage. When your customers see that Ziff Davis Labs has tested your Internet or technology product, they will know they can trust the results.

Beat the clock

LIVING ON INTERNET TIME?

Are you fighting to earn customer trust? Facing the challenges of hiring expert testing staff and buying costly test equipment?
Turn to ZD Labs. You'll save time, valuable resources, and energy by testing with ZD Labs, the independent, trusted, testing service designed to give you a competitive edge. For more detailed information and a complete listing of ZD Labs' testing services, go to www.zdlabs.com.

Compuware Corporation is a leading worldwide provider of software products and professional services.Compuware productivity solutions help 14,000 of the world's largest corporations more efficiently maintain and enhance their most critical business applications. Providing immediate and measurable return on information technology investments, Compuware products and services improve quality, lower costs and increase the speed at which systems can be developed, implemented and supported. Compuware employs more than 11,000 information technology professionals worldwide, including more than 6,600 in its professional services organization. With calendar 1998 revenues in excess of $1.5 billion, Compuware is the world leader in client/server development technology.

The Compuware QACenter product family offers the best in automated testing technology for today's enterprise needs. From mainframe to client/server to web, from test management through validation, from unit through load testing, for Year 2000 and euro conversions-QACenter tools are helping companies achieve consistent, dependable application performance

The Products:

QACenter products are available separately or as a part of an integrated family of products that includes:

QARun- comprehensive capabilities for thorough, accurate client/server testing.

QALoad- Load testing for client/server applications

QADirector- managing the testing process across the enterprise.

QAHiperstation- broad capabilities for testing VTAM-based applications.

QABatch- regression testing for MVS batch applications.

QAPlayback- advanced capabilities for testing CICS-based applications.

QAHiperstation+ - testing mainframe applications from the desktop.

QAPlayback+- windows-based interface for testing mainframe-based applications from the desktop.

QASolutions- process, people and products for quality testing.

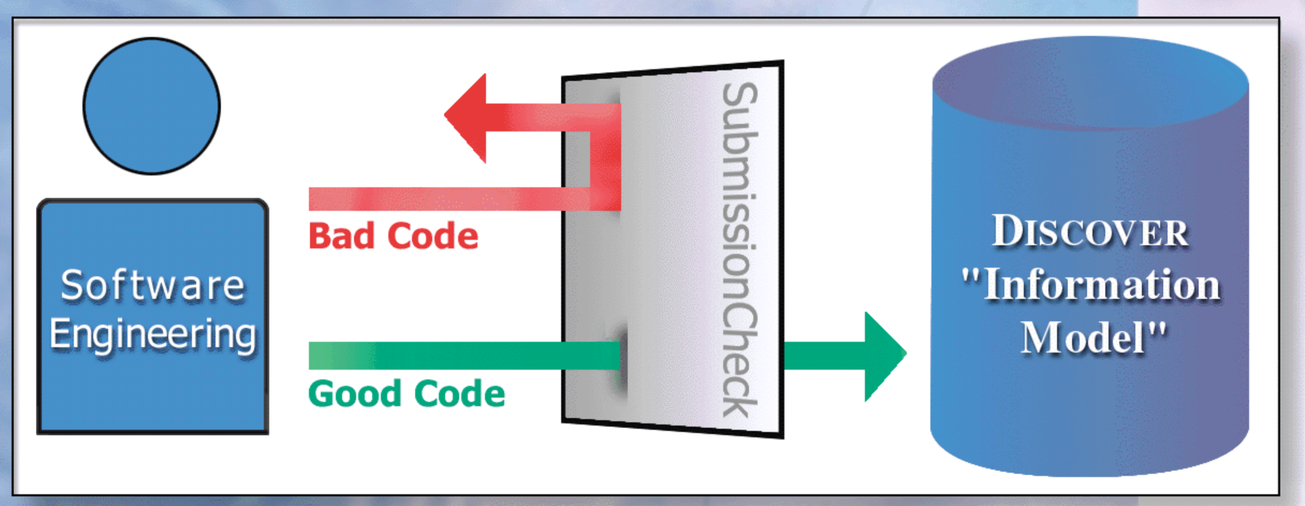
COMPUWARE CORPORATION

With trailing 12-month revenues of more than $2 billion, Compuware is a world leader in the practical implementation of enterprise and e-commerce solutions. Compuware productivity solutions help 14,000 of the world's largest corporations more efficiently maintain and enhance their most critical business applications. Providing immediate and measurable return on information technology investments, Compuware products and services improve quality, lower costs and increase the speed at which systems can be developed, implemented and supported. Compuware employs more than 15,000 information technology professionals worldwide. For more information about Compuware, please contact the corporate offices at (800)521-9353. You may also visit Compuware on the World Wide Web at www.compuware.com.

COMPUWARE CORPORATION
31440 Northwestern Hwy
Farmington Hills, MI 48334

Software Emancipation

2000 Quality Week
Booth number 312



DISCOVER®'s SubmissionCheck checks compliance with predefined quality standards **before** submission.

Please join us for...
a technical presentation

"Transforming Software Quality Control To Quality Assurance"

Thursday, June 1, at 11:00 a.m.

and product demonstation at 2:00 p.m.

Visit us at booth number 312 and GET A FREE GIFT

DISCOVER®
Development Information System

Attacks Industry Software Quality Problem

DISCOVER® is a software development information system consisting of a comprehensive, integrated set of applications and tools. DISCOVER analyzes source code and creates a database (the Information Model™) that captures the interrelationships between all entities in the code base.

DISCOVER offers the means to analyze, enforce, control, monitor, and improve software quality early in the development cycle.

DISCOVER's new quality process-control and management solutions empower management with unprecedented control, giving them precise, predictable, and measurable information about the quality of their software as it is being developed, enabling them to make business decisions about the development project on the basis of objective data, rather than intuitive assessment.

SOFTWARE EMANCIPATION TECHNOLOGY

Headquarters: 15 Third Ave. Burlington, MA 01803
Phone: 781.359.3300 **Fax:** 781.359.3399
Email: info@setech.com **Website:** www.setech.com

Vanteon e-Quality and Test Solutions
QA with a high IQ.

QA and Test solutions as creative as the software you write, as innovative as the web site your business depends on.

Discover the significant advantages of partnering with Vanteon, who has the depth of experience, resources and collaborative spirit to work closely with you to create innovative and highly effective QA and Test solutions.

For more information about Vanteon, please visit BOOTH #301.
www.vanteon.com
QA@vanteon.com
800.266.5046

QA/Test Services
- QA Consulting
- Web Performance
- System (Black Box) Testing
- Unit (White Box) Testing
- OS/Browser/Platform Compatibility
- Automation
- Functional Localization
- Rapid Quality Evaluation
- FDA/ISO/CE Mark Process Consulting

For
- eBusiness
- Wireless Peripherals
- Desktop/Consumer
- Networking
- Client/Server
- Embedded
- Print/Imaging

Technologies
- Windows
- Mac
- UNIX
- LINUX
- Windows CE
- VxWorks
- QNX
- PSOS

VANTEON™

Running Low On Internal Test Resources? Tell Us How We Can Help!

Compatibility & Functionality Testing • Performance Tests • E-Business & Web Site Testing
Environmental (EMI / FCC, CE Mark)
Localization (35 Languages)
Certification Testing

- Daily detailed test reports
- A comprehensive final summary report
- Quick project turnaround (5-21 days)
- Certification tests include a media kit

Call or email today for a free test proposal
(written specifically for your product)

NTS-XXCAL

Tel: (310) 641-7700 • Fax: (310) 641-5827
www.xxcal.com • labsales@xxcal.com

©2000 NTS/XXCAL

NTS-XXCAL

NTS-XXCAL is the oldest and largest testing laboratory of it's kind, having tested Software & Hardware since 1982, Environmental testing since 1961. Lab locations are in Los Angeles, London and Japan. NTS IS AN "NRTL" CERTIFIED (NATIONALLY RECOGNIZED TEST LAB) - A US GOVERNMENT RECOGNIZED LAB. NTS-XXCAL tests Software, Hardware & Peripherals. Decrease your time-to-market! We help you validate the integrity of your product by covering your target market thus leaving your valuable development resources free to work on your new products. This makes our testing service extremely cost effective!
www.ntsxxcal.com

DON'T RISK

THE QUALITY OF YOUR CUSTOMER'S EXPERIENCE

THEY JUDGE YOU ON IT.

VISIT US AT WWW.WATCHFIRE.COM/QW
TO DOWNLOAD OUR SOLUTIONS AND
DISCOVER THE POWER OF QUALITY.

watchfire
Website Intelligence Solutions

MEDIA SPONSORS...

Software Development Magazine, published by Miller Freeman Inc., is the leading trade publication covering the people, products, and practices of corporate development. Software Development magazine provides practical solutions for technical team leads and project managers leading corporate application development projects. Editorial features strategies and solutions for evaluating and implementing development tools and solutions for evaluating and implementing development tools and technologies, software engineering and quality assurance practices, project and people management strategies, process improvement, as well as tips on Software Development magazine go to: http://www.sdmagazine.com.

Cutter Consortium, LLC helps organizations leverage IT for competitive advantage and business success. Cutter Consortium offers high-level advisory and publication services, journals, online resource centers, on-site assessments, consulting, and training to help organizations forge solutions to the IT challenges they face. What is unique about the Consortium is that its products and services are provided by the top thinkers in IT today -- a distinguished group of internationally recognized experts committed to providing top-level, critical, objective advice.

Software Business (description to be supplied).

"Ferreting Out Bugs"

Azor, Inc. is located in Palo Alto, California and specializes in outsource e-commerce testing and software application testing. Azor provides methodology trained software testing experts to do work on your site or in our lab. Azor has its own proprietary test automation and estimation tools (Ferret® and CodePlan™) and rents and sells testing equipment. Azor test engineers are trained and certified in a number of popular automated tools (Win Runner®, Load Runner®, RSW e-Test Suite®, McCabe Visual Testing®, Soffront Track®, Segue Silk®, etc.), and Azor maintains relationships with the leading tool vendors. Azor also has partnerships with some of the largest testing companies in the world, bringing to bear additional resources on your project, if necessary. Azor supplies Test Engineers, Trainers, Technical Writers, and Project Managers, to do Test Planning, Test Design, Test Case Layout, Test Execution, Test Automation, Load and Performance Testing, Test Reporting, Test Documentation, Test Training, and Project Leadership. Azor's approach to software quality is to develop long term relationships with repeat customers to understand your business and leverage this knowledge for efficient testing and support. Azor also has a **No Bugs, No Bucks** offer. Azor will test your product for a week, and if we can not find a defect, the week of testing is on us. Azor is located at 1032 Elwell Ct, Suite 240, Palo Alto, CA. 94303, 650-934-2869, info@azor.com, www.azor.com.

Who We Are

Cisco Systems is the worldwide leader in networking for the Internet. Cisco's networking solutions connect people, computing devices and computer networks, allowing people to access or transfer information without regard to differences in time, place or type of computer system. Every day, Cisco and its customers are proving that networking and the Internet can fundamentally change the way companies do business.

What We're Really Like

Ours is a culture of driven by innovation. Every day, Cisco products and solutions are enriching people's entertainment options, expanding their educational experiences, simplifying their complex lives, and deepening the interaction they have with others. It's challenging – we encourage individual thought, and reward initiative and creativity. It's exciting – we move fast around here, the pace of the wired world demands it. It's fun – being part of one of the world's most successful companies feels good, and it also allows us the resources and time to turn your ideas into realities.

What It All Means

The global economy is the Internet economy. And no company understands that better than Cisco Systems. The Internet Revolution of today is driving change and economic growth around the world. We're using technology and expertise to enable a world where everyone and everything are seamlessly connected; where people have access to information, resources, and other people in the easiest ways possible. Which means the question "Are you ready?" is about a lot more than asking you if you're ready to start. It's about reminding you that you already have.

COMPUTER ASSOCIATES INTERNATIONAL, INC.

Computer Associates International, Inc. (NYSE: CA), with headquarters in Islandia, N.Y., is the world leader in mission critical business software that develops, licenses and supports more than 700 interoperable solutions worldwide. These solutions provide powerful end-to-end, distributed enterprise management, visual application development, object-oriented and relational database management; completely customizable, distributed business applications, Internet/Intranet, storage management, antivirus, communications, financial, manufacturing, and human resources systems. CA provides business technical consulting worldwide through its Global Professional Services (GPS) organization. CA employs over 17,500 people in 160 offices in more than 100 countries and had revenue of $5.3 billion in fiscal year 1999. CA can be reached by visiting www.cai.com, emailing info@cai.com or calling 1-631-342-5224.

COMPUWARE CORPORATION

With trailing 12-month revenues of more than $2 billion, Compuware is a world leader in the practical implementation of enterprise and e-commerce solutions. Compuware productivity solutions help 14,000 of the world's largest corporations more efficiently maintain and enhance their most critical business applications. Providing immediate and measurable return on information technology investments, Compuware products and services improve quality, lower costs and increase the speed at which systems can be developed, implemented and supported. Compuware employs more than 15,000 information technology professionals worldwide. For more information about Compuware, please contact the corporate offices at (800)521-9353. You may also visit Compuware on the World Wide Web at www.compuware.com.

COMPUWARE CORPORATION
31440 Northwestern Hwy
Farmington Hills, MI 48334

Dunn Systems is a leading e-Business consulting firm who partners with our clients to transform the way they do business. We leverage our Quality Assurance, Strategic Marketing, Web Design, Application Development, Data Warehousing and Technical Education practice areas to deliver complete solutions quickly and with minimal risk. Dunn Systems Quality Assurance services include Test Planning, Defect Tracking and Management, Regression Testing and Performance/Web Testing. We have a proven track record of implementing, training and mentoring clients in the use of automated testing tools and an integrated testing methodology. Let Dunn Systems identify a functional Quality Assurance process and the necessary tools to ensure your enterprise and e-Business applications work flawlessly before reaching the end user. For more information, contact us at (847) 673-0900 or visit www.dunnsys.com.

eValid ™ Test Services Summary

BenchMark Testing
Transaction Testing
Security Testing
SpotCheck Testing

Single or Multiple URLs
Single or Multiple Text Area Checking
Single or Multiple User Interactions
Daily, Four-Hourly, Hourly, Quarter-Hourly Checking Intervals
Automatic Alert Modes
Emailed Alert Reports
Archival of Data

Assuring Web Site Quality, Reliability and Performance

Web users demand a lot! If your web site is too slow, has errors, is missing content, or fails to work correctly, your online customers may click away in an instant -- with likely unhappy results for your revenue stream.

While you can't affect very much how fast the Web itself works, you can control *what* your web site does and *how well* it does it. And at least you need to be aware of small problems so they don't become big problems!

eValid **Test Services** enable companies and web site managers to monitor and optimize E-commerce service validity, availability, internal quality, and performance -- as perceived *by typical users* with realistic test scripts. This is the Quality of Service (QoS) that really counts! You get test results that reflect what *your* customers see with simple and easy-to-understand reports.

eValid **Test Services** are customized applications of TestWorks Test Tools that validate that web sites meet key needs of performance, content, behavior, and security. Our unique "user perspective" testing assures your sites quality in the *context in which your customers use it* -- from a web browser running under real-world conditions (e.g. on a mid-size PC with a 28.8 kpbs or 56.6 kbps dialup modem).

Key eValid Capabilities

With pre-programmed eValid **Test Services** you can test your web site exactly the way a user uses your web site. You can make sure your web site meets quality criteria you set:

- **Benchmark how fast your site runs** from hour to hour and day to day.
- **Evaluate your E-Commerce transactions** to confirm they are working advanced *validate selected text* capabilities .

- **Monitor your site security** by performing with tests that behave just like a real user.
- **SpotCheck key areas** of your web site to make sure it hasn't been hacked!

Overall you gain **confidence** that your web site is working the way you want it to. And, you get email and/or pager warnings immediately when a problem is found.

How eValid Works

The foundation of all eValid **Test Services** is application of custom programmed tests on a regular schedule with automated responses to the web site manager. These series of tests are designed to confirm critical properties of the web sites they exercise, including:

- Geographic availability in multiple time zones;
- Confirmation of content of selected pages;
- Simulation of user interaction including type-ins and FORM submittals;
- Checking security access to restricted areas on your web site;
- Measuring typical-user retrieval times for one or more URLs.

In short, all the things that your own testers would do if they knew how and had the tools to do the work.

eValid Features and Benefits

Customized outsourced automated testing. You get the advantage of the best, most efficient scripting, with all of the test suite development and execution done for you on a service-level basis at very attractive pricing.

Powerful test tools "at your service". eValid **Test Services** is based on use of released versions of TestWorks products such as the CAPBAK/Web browser-based capture/replay system and the SMARTS test management system, combined with custom-developed test scripts and custom-developed test support utilities.

Realistic simulated-user tests. Tests are run the same way your users run them, at mid-speed dialup connections. Results are based on realistic response times and download rates and are completely isolated from the effects of caching, etc.

Multiple time zone coverage. eValid **Test Services** are available from up to six time zones (four USA plus Europe and Pacific Rim) so you know how your web site appears to users in all of the major markets.

Automated notification of test failures. When something goes wrong you are automatically sent email (or, you're paged) telling you of the problem.

Superior Return on Your Investment. Just imagine if your web site accounts for $100K/month in business. If your site is "off the air" even for a few hours, your lost-income costs could exceed the **monthly** cost of an eValid Test Services subscription.

eValid -- An Integrated Set Of Test Service Offerings

eValid **Test Services** are integrated to offer you a range of standardized test capabilities adapted to the most common web site managers' concerns. All eValid **Test Services** are oriented to measuring and testing how real users perceive your web site. Tests are executed from typical dial-up web site connections 24x7 from every 4 hours (lowest intensity) to every 15 minutes (highest intensity).

eValid Test Services			
BenchMark Testing	**Transaction Testing**	**SpotCheck Testing**	**Security Testing**
Tests confirm that selected web site pages (URL's) are present and are retrieved within customer preset maximum time limits. Either 1-5 URLs or 6-12 URLs are downloaded in a non-cached environment and carefully timed. Error messages are issued when the actual retrieval times exceed customer specified limits. There is complete logging of all test data for later analysis.	Tests simulate a "shopper" selecting items and purchasing them in a realistic manner using E-commerce transactions on your web site. Up to 2 "picks" of products are included, plus user inputs that simulate the buying process. Error messages are issued if the testing finds selected items are unavailable or if the purchase transaction fails.	Tests simulate a customer looking at various pages in your web site for certain text items or passages. Tests download a selected set of pages from your web site and examine the content of up to 6 text areas from each page. Error messages are issued if the testing finds the text you expect to be present on the page is absent or has moved to a new location.	Tests simulate a customer accessing information in a login-protected, restricted area of your web site and confirm that access is granted only with proper ID and password. Tests go to selected page on your web site and simulate user qualification (login) process. At least two tests are performed: a legal login (success) and an illegal login (failure). Error messages are issued if the testing finds that authorized users cannot get access when they are supposed to, or that unauthorized users gain access when they are not supposed to.
Complete Description	Complete Description	Complete Description	Complete Description

eValid Pricing and Customization Options

Typical web sites in E-commerce focused companies often represent a very large investment, and in many companies this investment is not protected against failure. eValid **Test Services** are priced according to the service type and level of service chosen, and according to the level of validity of the web site that the service helps assure. eValid **Test Services** are a one-stop solution to questions of web site effectiveness and quality.

eValid **Test Services** are priced from $89/month to $579/month. A minimum 3-month service commitment applies; there are discounts for multiple service combinations and yearly subscriptions. Complete details and a free 3-day evaluation of eValid **Test Services** are available on request from webtest@soft.com.

In addition to standard eValid **Test Services** we can customize the tests we run to handle very large web sites and to handle web sites that present special or unusual types of operation. Please explain your needs to us at

webtest@soft.com and we'll respond promptly.

Software Research, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Phone: +1 (415) 550-3020
TollFree: +1 (800) 942-SOFT [USA Only]
Fax: +1 (415) 550-3030
E-Mail: info@soft.com

IEEE COMPUTER SOCIETY
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314
Phone: 714-821-8380
Fax: 714-821-4010
email: membership@computer.org
Web Site: http://computer.org
contact: Marian Anderson

IEEE Computer Society, one of the most prestigious professional associations in the world, serves its members through numerous publications, conferences, and workshops. Membership information, magazines, and books are on display.

Please visit us at BOOTH #409

IIST is an educational and professional development organization that has been founded to *promote a disciplined approach to software testing and to caution against ad hoc testing by non-qualified individuals and groups*. IIST operates according to the following charter:

- To promote and provide education and certification of software testing professionals around the world.

- To create a pool of qualified software testing professionals to meet the needs of testing organizations.

- To provide assistance and guidance to members performing testing of all types of software systems.

- To partner with different software quality groups around the world to assure continuity of information flow among testing professionals.

- To serve as a clearinghouse of services, methods, and tools supporting the discipline of software testing.

IIST has the following offerings:

- **Certification of Software Testing Professionals (CSTP)**
- **Comprehensive Training program to support the certification**
- The **International Conference on Practical Software Quality Techniques (PSQT)**
- **The Journal of Software Testing Professionals (JSTP)**

Contact Information:

International Institute for Software Testing
8476 Bechtel Ave
Inver Grove Heights, MN. 55076
Attn: Steve Brown, Director of Education & Marketing
Phone: (651) 306-1387
Fax: (651) 552-0791
www.softdim.com
sbrown@softdim.com

LDRA Ltd
131 Mount Pleasant
Liverpool L3 5TF
UK

Tel: +44 (0)151 708 8505
Fax: +44 (0)151 709 2027
E-mail: info@ldra.com
www.ldra.com

The Company

LDRA, the world's longest established vendor of automated code analysis testing tools, celebrates its 25th birthday this year. World renowned for its role in developing industry leading testing methodologies in this highly specialised field, LDRA has recently been described in Software Magazine as the market leader.

LDRA Testbed helps its users to enforce industry quality standards on their code, to locate software defects and to ensure that code is testable and maintainable in the long term. Whether unit, integration or regression testing, LDRA Testbed is used to maximise the effectiveness of the testers' efforts. Importantly, LDRA Testbed has been proven to improve testers' motivation.

Complementary to the company's R&D into testing and tools development, LDRA provide testing consultancy. In this area, LDRA have been responsible for software inspections in motorsport, including FIA Formula 1&3 and the BTCC.

Through LDRA's worldwide representation, LDRA's customers enjoy exceptional sales and product support.

On Show

LDRA will be presenting their latest tool enhancements, including TBrun, a revolutionary test harness generator to assist you in automating your unit testing process.

TBrun:
- Automatically generates test drivers & harnesses
- Runs tests on code units
- Detects changes in source code
- Documents changes required in tests
- Performs regression tests
- Maintains test data and results
- Runs in a host/target environment

We welcome you to visit our stand and come along to LDRA's Technical Presentation and Demonstrations.

Interim Technology Consulting – Software Quality Management
250 Word Description

The incredible growth of the Internet has introduced business change on such a massive scale that it is forcing companies to redefine their core strategies and the manner in which they conduct business. With organizations quickly responding to this rapidly changing business by developing eBusiness applications, the risks to their business is skyrocketing.

For more than 30 years, clients have trusted Interim Technology Consulting with their Software Quality Management (SQM) solutions for both Information Technology (IT) and user departments. Our world-class expertise along with our unique approach to service delivery, provides an unbeatable combination in exceeding our client's expectations with each and every engagement.

Our services include Strategic Quality Consulting, eSQM sm through eBusiness Testing Strategy and Planning as well as Independent eBusiness Integration Testing, Software Testing and Validation, Tool Automation Evaluation and Implementation, Project Quality Services, Metric Services, User Services, Environment Readiness Testing, Performance/Recovery Testing, Security Testing and Usability Testing Services.

Our services are supported by a proprietary set of tools and methodologies including our state-of-the-art hypertext VALI/TEST Pro® (Validation & Testing methodology) and the Interim Technology/SQM Tool Suite sm.

The future of Software Quality Management -- eSQM sm
Delivering quality applications that meet business' needs the day they are implemented has become a critical challenge for IT managers. As business critical applications have become more complex, so has the validation and testing process. Due to the complexity of the architecture and rapid application development, testing requires a defined, disciplined strategy, a revised approach, tools and trained/experienced resources. Therefore, we've introduced a service offering wrapped around eSQM sm – enhanced eBusiness solutions for our clients encompassing long-established Software Quality Management success.

Interim Technology Consulting
823 Commerce Drive
Oak Brook, IL 60523
630-645-8800
630-645-8801 – Fax
www.interimtechnology.com

McCabe & Associates, Inc.
9861 Broken Land Parkway
Fourth Floor
Columbia, Maryland 21046
1-800-638-6316
Balt. 410-381-3710
Fax 410-995-1528
http://www.mccabe.com

McCabe & Associates…because some things just have to work right. In today's competitive business environment software is no longer a productivity tool, it has become the business and it must work right. McCabe & Associates enables IT to deliver better applications, by providing products and process that implement a relevant, repeatable, and measurable approach to managing software changes and their effects on the testing and quality of applications. McCabe Integrated Quality (McCabe IQ), our proven methodology and product suite, tightly integrates change management with software quality analysis and increased test effectiveness to promote and assure the delivery of high quality software systems. McCabe IQ has been implemented by global corporations and government agencies in their efforts to:

- Manage and thoroughly test software changes
- Quickly and easily merge software changes into released code
- Objectively assure the quality of released software
- Lower the maintenance time and costs of strategic systems
- Assure the stability of enterprise software

About ParaSoft

ParaSoft Corporation develops and markets error prevention and error detection tools for UNIX, Windows, and Internet development environments. ParaSoft's award winning products and patented technologies assist developers and managers in improving software quality, accelerating time to market, and reducing development costs. ParaSoft's headquarters is located at 2031 S. Myrtle Ave., Monrovia, Ca 91016. Telephone (888) 305-0041. Fax (626) 305-3036. Email to: info@parasoft.com. URL:http://www.parasoft.com

Product Description

Insure++is a tool that automatically detects large classes of programming and run-time errors, including algorithmic anomalies, bugs, and deficiencies.

CodeWizaed is a unique coding standars enforcement tool that uses patented Source Code Analysis technology (patent #5,860,011) to help deveopers prevent errors and standardize C++ code automatically. CodeWizard spontaneously enforces C++ coding standards, saving hours of labor-intensive analysis.

Jtest is a fully integrated, easy-to-use, automatic class testing tool for Java. Jtest integrates every essential type of Java testing in one intuitive tool, automatically performing white box testing, regression testing and static analysis, and helping to automate black-box testing.

SiteRuler is a comprehensive Web site testing and maintenance tool that checks for broken links, non standars HTML, and spelling errors. In addition, SiteRuler can publish a site and test it before it goes online, preventing any WEB site glitches from reaching the public. SiteRuler's open architecture allows Web developers to plug in their favorite editors, authoring tools, graphic tools, and source control.

C++Test, a soon-to-be released tool from ParaSoft, automates the unit testing process for C and C++ developers. C++Test is an easy-to-use tool taht can easily fit into any existing C/C++ development cycle. C++ Test automatically tests any number of C functions or C++ classes from any program that can compile with the Microsoft Visual C++6.0 compiler, including embedded systems and device drivers.

A Major Technological Leap Forward

PolySpace Technologies offers a static verification product: **PolySpace Verifier**. An innovative and efficient tool used to automatically and exhaustively detect run-time errors in Ada and C applications.

PolySpace Verifier discovers all run-time errors including division by zero, overflow, out of bounds array access, arithmetic exceptions, illegal type conversion, concurrent access to shared data without execution nor intrusion into the software (source code is enough).

The unparalleled benefits of our technology

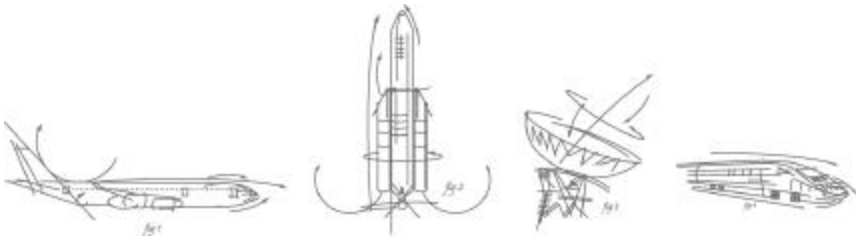
1. Efficiency and ROI: a significant level of software process automation, providing quickly measurable productivity, reliability, time to market and reproducibility gains; Both automation and early detection of errors is a well-known source of time and money saving. PolySpace helps in detecting all run-time errors before you run integration tests.
2. Easy-to-use: a remarkably low level of disruption of established development methodologies and tools, achieved in particular by compatibility with existing programming languages; Use PolySpace Verifier just as source code compiles without changing your development process.
3. Quality: a level of software safety that simply was impossible to attain before. PolySpace Verifier is the unique product solution able to exhaustively detect run-time errors in software.

Markets

PolySpace Technologies addresses the needs of companies building mission-critical, embedded and real-time software systems in many sectors including avionics and space, automotive and ground transportation, consumer electronics, medical equipment, energy and telecommunications.

… … …

PolySpace Technologies *is a start-up high-technology company whose* ***mission*** *is to provide our customers with highly innovative, unintrusive and quickly deployable software verification and validation tools for real time systems.*



PolySpace Technologies
c/o INRIA - 655, avenue de l'Europe - 38330 Montbonnot Saint-Martin – France -
Tel. : +33 476 61 52 60 - Fax : +33 476 61 54 09
email: contact@polyspace.com

www.polyspace.com

MuTek Solutions, Inc. targets the world's leading software developers and commercial organizations. MuTek's flag-ship product- BugTrapper, is used by independent software vendors (ISVs) that develop software for resale, information systems (IS) departments that develop software to support internal operations, and engineering departments that develop software as a component of a manufactured product. The company mission is to become a market leader in providing support solutions for service desks and personal computer users.

Aimee Potter
Raleigh Group International
5440 Atlantic Springs Rd./ Suite 115
Raleigh, NC 27616
(919) 878-3717
http://www.ralgi.com

RATIONAL SOFTWARE

Rational Software, the e-development company, helps organizations develop and deploy software for the Internet through a combination of tools, services and software engineering best practices. Rational's e-development solution helps organizations overcome the e-software paradox by accelerating time to market while improving quality. In 1999, International Data Corporation recognized Rational as the leader in multiple segments of the software development life cycle management market.
www.rational.com

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Web: www.rational.com

SaleView Systems is an application service provider (ASP) offering Internet performance and content quality monitoring with real-time alert notifications. Flexible, and non-invasive, multi-protocol monitoring is performed on a 24x7 basis from various locations around the world. SaleView emulates end-user web activities to determine the actual performance and availability experienced by your web visitors. SaleView can monitor standard protocols (i.e. FTP, HTTP, etc) as well as complex web-based applications. Unlike most ASP services, the customer, via a web interface, can control monitoring and notification configurations. Contact24 technology provides sophisticated alert notification management, including flexible escalation policies that notify support personnel as problems are observed. Notifications can be placed to email, directly to two-way pagers, and to voice phones using text to voice technology.

Segue Software, Inc.
201 Spring Street
Lexington, MA 02421

Phone: 781-402-1000
Fax: 781-402-1099
Email: info@segue.com
Web: www.segue.com

Segue Software, Inc. (NASDAQ: SEGU), a leader in providing e-business reliability software products. Segue develops and markets solutions that help companies deliver more reliable e-business systems. Segue partners with clients deploying business-to-business and business-to-consumer sites to help them achieve dial-tone level reliability and a rapid, integrated quality assurance methodology. Segue Software is the first and only company to deliver a comprehensive solution for ensuring the reliability of application-level transactions within complex middleware environments. Segue's Silk product family includes:

- SilkTest for automated functional and regression testing;
- SilkPerformer for load and performance testing;
- SilkPilot for functional and regression testing of Java programs through CORBA/IIOP, RMI and EJB interface specifications
- SilkObserver for diagnostics and monitoring of distributed Java and non-Java objects over IIOP
- SilkMeter for tracking of how Java server objects are used, enabling proper billing in e-commerce, collecting usage metrics and enforcing access rules
- SilkRadar for automated defect tracking and results management and
- SilkMonitor for real-time monitoring of Web applications and data base servers.

Representative customers from around the world include AT&T, Citibank, Ericsson, Hewlett-Packard, Liberty Mutual, Lockheed Martin, Lucent, Merrill Lynch, Pratt & Whitney, Sony Pictures, Sprint, T.Rowe Price, Telcordia and Warburg Dillon Read.

Headquartered in Lexington, Massachusetts, with offices across North America and Europe, Segue can be reached at 800.287.1329, or at www.segue.com

3/00

Company/Product Profile

Company Profile:

SilverMark, Inc. provides object test automation tools and services. SilverMark's object testing solutions and expertise have a proven track record throughout the world, in various industries such as insurance, banking, finance and manufacturing.

Product Description:

SilverMark's Test Mentor for Java(tm) - SilverMark is presenting their automated, component testing tool for Java. Integrated with the VisualAge(r) Java development environment, Test Mentor for Java takes component test automation to new levels by providing a specialized test development and code generation environment. Test Mentor for Java strongly promotes well structured, reusable tests through a component-oriented test architecture. Test Mentor for Java speeds test development and reduces maintenance costs, making the ideal of parallel code and test development practical, economical, and enjoyable.

SilverMark's Test Mentor for Smalltalk - The most effective and complete automated testing solution for Smalltalk. Test Mentor for Smalltalk has been developed for IBM's VisualAge Smalltalk, Cincom's VisualWorks Smalltalk, as well as IBM's VisualAge Generator. Test Mentor for Smalltalk automates test creation by generating tests from recorded user interface interactions, as well as Rational Rose or IBM UML Designer design models. Test Mentor for Smalltalk creates and executes tests in development or deployment images.

SilverMark's Connection Detangler for VisualAge Smalltalk & Generator - is a "Powerful connection debugger that puts to shame all previous attempts to make a debugging tool for links and connections" - product review

SOFFRONT SOFTWARE, INC.

Since 1992, more than 45,000 users and over 2,500 corporations worldwide, including many Fortune 500 companies have thoroughly tested and ranked TRACKWeb as their number one choice to satisfy their Defect Tracking, HelpDesk and Sales Automation needs.

TRACKWeb is fully web-based and allows full database administration, including customization, report generation, data submission and access to information through a web browser. TRACKWeb offers fully customizable and easy-to-use solution for organizations of all sizes.

TRACKWeb Defects application is the pioneer defect tracking application for Windows environments. It sets the software standard for managing product Defects / Bugs, Code Changes, Test Cycles, Product Releases, System Configurations and more.

The product integrates with TRACKRules component for defining business rules, such as automatic escalation of issues, field updates and email notifications. TRACKKB component helps build defect Knowledge Base that automates location of possible solutions to recurring defects.

To satisfy your helpdesk need Soffront offer TRACK HelpDesk application. The application is designed to offer multi-level helpdesk, customer service, and customer support structure. This fully web-based and fully-customizable enterprise-level application is designed to track all aspects of a helpdesk operation including problem management, problem resolution, asset inventory management, and change request management. TRACK HelpDesk is designed to handle more calls in less time and achieve lower costs of operation while assisting in the construction of a global helpdesk operation across multiple campuses. The application integrates with a knowledge base to further automate and expedite helpdesk operation.

Soffront Software Inc.
830 Hillview Court, Suite 140
Milpitas, CA 95035
Web: www.soffront.com

- Software Development Technologies (SDT) is committed to supporting organizations throughout the world with improving their software quality and testing efforts. Assistance may take the form of training and consulting — Software Testing, Software Test Automation, Test Planning and Design, Integrated Test Design and Automation and Technical Review and Inspection — software test automation products and/or outsourcing your test efforts. SDT offers the following solutions for solving your testing needs:
    - Technology Assessments
    - Test Design and Automation Solution (TestFrame™)
    - Review and Inspection Solution (ReviewPro™ and TRIP™)
    - Test Curriculum -- onsite and public courses as well as curriculum licensing
    - Outsourcing Test Services

  SDT clients include Cisco Systems, Sun Microsystems, Norwest Financial, Fidelity Investments, NASA and GoTo.com.

- Software Development Technologies (SDT)
  125 South Market St., Suite 700, San Jose, CA 95113
  ph (408) 297-1911
  fx (408) 297-1993
  www.sdtcorp.com

SOFTWARE QUALITY ENGINEERING

Founded in 1986, Software Quality Engineering (SQE) assists software professionals and organizations throughout the world with improving their software testing and quality engineering practices. The company's hands-on experience and training expertise help companies - large and small - to improve testing practices, gain measurable control over software projects, and ultimately deliver better software.

SQE can assist your software organization through the following services: Professional Services - SQE understands that a one-size-fits-all approach to software improvement is rarely effective. Our professional support process begins with assessment and analysis of the unique needs of your organization, followed by hands-on, targeted action. From on-site training to in-depth project implementation support, SQE has the technical experience and resources to address the special needs of your organization. Training Seminars - SQE delivers specialized training seminars on systematic testing, test automation, Web/eBusiness testing, software metrics, requirements, software management, and more. These high-leverage courses are presented publicly throughout the year and provide verifiable results in improving productivity and software quality. International Conferences - SQE organizes international conferences, including ASM Software Measurement Conference, SM Software Management Conference, and the STAR Software Testing Conferences. Publications & Research - In an effort to make good software testing and development information more readily available, SQE publishes Software Testing & Quality Engineering magazine. Each issue focuses on nuts-and-bolts information and practices you can apply in your projects and daily work.

Software Quality Engineering
330 Corporate Way
Suite 300
Orange Park, FL 32073(904) 278-0707
(800) 423-8378
sqeinfo@sqe.com
www.sqe.com

TestWorks

Software Research, Inc.'s TestWorks, an integrated suite of software test tools, is the broadest test tool suite available. TestWorks tools help automate and streamline the software development and testing process with product lines that work independently or as an integrated toolsuite. TestWorks is the only tool suite that offers Regression Testing, Test Suite Management, and Test Coverage support for Web **and** Windows **and** UNIX Platforms.

TestWorks Overview

You can get most of your basic questions about TestWorks products answered from the TestWorks Frequently Asked Questions (FAQs).

Send Email with your question to `info@soft.com` and we're respond quickly.

TestWorks for Web

TestWorks/Web™ is a bundle of software test tools tailored to support complete regression testing for Web Sites, including those that exploit the advanced features of Java™.

The **TestWorks/Web** product bundle consists of three major components:

- CAPBAK/Web for Windows
  - Frequently Asked Questions(FAQs)
  - Take a Tour of CAPBAK/Web
  - Features and Benfits Summary
  - Pricing and Order Form
- SMARTS (hierarchical test controller)
- TCAT for Java/Windows (coverage analyzer for Java)

Read the new White Papers by Edward Miller on The WebSite Quality Challenge. and WebSite Testing.

Consider one or more of our new subscription-based eValid Test Service Options that analyze your site's performance, security, reliability, content quality and performance.

TestWorks For Windows

TestWorks for Windows, an integrated suite of automated testing tools, is the broadest suite of tools available to test applications running under MS/Windows (Win3.1), MS/Windows NT or MS/Windows '95/'98 (Win32).

Testworks for Windows has two main bundles of tools:

- TestWorks/Regression
  - CAPBAK/MSW
  - SMARTS/MSW
- TestWorks/Coverage
  - TCAT C/C++
  - TCAT for Java/Windows

TestWorks For UNIX

TestWorks for UNIX is designed to work independently or as an integrated tool suite to provide an efficient, automated testing environment for most UNIX-based platforms.

TestWorks for UNIX consists of three product lines:

- TestWorks/Regression
  - CAPBAK
  - SMARTS
  - EXDIFF

- TestWorks/Coverage
  - TCAT C/C++
  - TCAT for Java/UNIX
  - TCAT/S-TCAT Ada/f77
- TestWorks/Advisor
  - METRIC
  - TDGEN
  - STATIC

Your Quality Process

TestWorks products support a multi-filter ***Quality Process Architecture***™ for software development projects in C, C++, Java, plus Ada and F77.

The ***TestWorks Quality Index*** is a quantitative Quality Index (Figure Of Merit) that characterizes your Application Development Process.

Downloading Products

Download Datasheets

DOWNLOAD PRODUCTS

License Key Request

QuickStart Manuals

User Manuals

Platforms Support

Check **Available Bundles, Products, Platform/OS, and Versions**

Software Research, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Phone: +1 (415) 550-3020
TollFree: +1 (800) 942-SOFT [USA Only]
Fax: +1 (415) 550-3030
E-Mail: info@soft.com

Superior IS is a rapidly growing IT technology solutions provider to both the Global 1000 and fast-moving dot.com startups. SIS specializes in executing and delivering IT services and products that are a combination of high-end consulting services wrapped around business-focused high technology solutions.

1. Strategic and Technology eConsulting

Superior IS has always focused on leading edge strategic business and technology issues. Current services include:

- **IT and Internet Strategy & Planning**
- **IT and Internet Architecture Planning & Implementation**
- **Acting CTO's**
- **Strategic Sourcing**
- **Web Development Project Management**
- **ECommerce Development & Deployment Management**
- **LAN/WAN Technical Services**

2. Outsourced Testing Solutions at the Speed of eBusiness

Superior IS's testing solution is a factory-based, fast and comprehensive approach to testing, created specifically for the eWorld.

- **Functional Testing:** Removes most of the functional testing burden from your staff by shifting the load to our testing specialists located in our "testing factory"
- **Load Testing:** Stress testing web-sites with 100's of users, differentiating between users, accessing your site simultaneously to ensure scalability.
- **Site Integrity:** Review of the navigation, HTML syntax, graphics download speed, and multiple browser compatibility to ensure proper operation by all users.
- **Site Monitoring:** Vigilant quality assurance by providing continuous/scheduled testing and monitoring to ensure peak performance**Superior IS – Experience Companies have Trusted for over Fifteen Years**

For more information on Superior IS's eTechnology Services, please contact:

Matthew Dusanic
Superior IS
760 Market Street, Suite 306
San Francisco, CA 94102
415 398-6500
mdusanic@superioris.com

Greta Cohen
Superior IS
3361 Berry Drive
Studio City, CA 91604
213-500-8097
gcohen@superioris.com

www.superioris.com

TechExcel, Inc.
Company Profile

TechExcel is a leading provider of integrated Web and client/server solutions for tracking and workflow applications. Founded in 1995, TechExcel is dedicated to integrating the power of the Internet with robust LAN-based systems, providing comprehensive solutions for today's demanding high-technology companies.

High Profile Customers

TechExcel's products have been embraced by some of the top companies and software development teams around the world. U.S. customers include Hewlett Packard, Texas Instruments, Dolby Labs, Honeywell DMC, Eastman Kodak, EDS, Fujitsu Software, and the Gartner Group. International customers include Nortel Networks (Canada), Cochlear Ltd. (Australia), Ericsson (Sweden), Orion (Norway), LionBridge Technologies (Ireland), Software Kinetics Ltd. (Canada) and Mosaic Software (South Africa).

Feature Product

DevTrack is the premiere defect- and project-tracking tool for software development teams. DevTrack comprehensively tracks and manages all product defects, development issues, and related documentation. DevTrack also provides robust reporting and communication tools, and allows for extensive customization. Intuitive and powerful, DevTrack provides a sophisticated Web and client/server solution for software development teams.

Features Include

Comprehensive tracking and management of software defects, feature requests, and other development issues.
Definable workflow allows the defect-resolution process to be fully defined and controlled, ensuring high-quality products.
Full ODBC-compliance ensures easy scalability from one to many thousands of users.
The industry's best integration with Microsoft's Visual SourceSafe version control software.
Automatic e-mail notification for any QA or development team member when selected events occur, based on definable criteria.
DevTrack Web provides nearly all of DevTrack's powerful features, with an easy and intuitive LAN-like interface.

The Telcordia Software Process Improvement Practice offers consulting expertise that allows for rapid, common sense improvement of processes. This is based on first-hand knowledge of successful techniques and quantifiable data to support potential savings in time-to-market, testing and development costs and increased customer satisfaction. The offerings are customized to meet the specific needs of each company's software development organization.

In 1999, Telcordia Technologies became one of the few corporations in the world to have their own software development organization assessed by the most stringent industry standard – the Software Engineering Institute (SEI) Capability Maturity Model® (CMM). The 3,500 person IT Group attained Level 5 status, making it the largest in the world ever to achieve this distinction. These results enable our expert consultants to bring practical experience, as well as, cutting edge methods for rapid process improvement to each or our clients, including: strategy definition customized process framework and goal-oriented metrics programs.

Telcordia Technologies, Inc., an SAIC company, provides software, engineering, consulting and training services to optimize the performance of communication networks worldwide. Telcordia was created in 1984, employs 6000 professionals, and has revenues of more than $1.2 billion. Telcordia (www.telcordia.com) is headquartered in Morristown, NJ, with offices throughout the United States, Europe, Central and South America and Asia Pacific.

TERADYNE SOFTWARE & SYSTEMS TEST

SST is the leading supplier of solutions for automating the system definition and test generation process for software systems. Major telecom suppliers currently use SST's products across a wide range of telecom systems to achieve 80-90% improvement in test development time and productivity. TestMaster is a general-purpose solution that generates tests for virtually any application running under any test execution environment or combination of test execution environments. Hammer CallMaster is a new optimized test generation solution for computer telephony applications that are tested using the Hammer IT Test system.

Teradyne Software & Systems Test
44 Simon Street
Nashua, NH 03060
Web: www.teradyne.com

Vanteon e-Quality Solutions

Vanteon e-Quality and Test solutions are as creative as the software you write, as innovative as the web site your business depends on.

For more than fifteen years, Vanteon has developed and refined the industry's most effective QA and testing methodologies, processes, documentation and tools to meet the business, market and quality requirements of our clients. As business and technology platforms constantly change, Vanteon continuously evolves its methodologies to keep pace with the rapid deployment of new web technologies, and hardware and software designs. At Vanteon, we not only implement QA and testing best practices for our internal product development, we create them for our clients as well.

Through its diverse experience, Vanteon has built a repository of knowledge accessible to companies looking for customized Quality Assurance and Testing solutions. Vanteon specializes in QA process consulting, test planning, test documentation development and test execution, including strategies and scripts for Automated Desktop (capture/playback) and Automated Web Testing (functional, load, stress, performance, benchmark). Compatibility testing is performed in our Configuration Test Labs. Vanteon solutions span eBusiness, wireless peripherals, desktop/consumer, networking, client/server, embedded and print/imaging. Our professionals are experts in a range of technologies, including Windows, Mac, UNIX, LINUX, Windows CE, VxWorks, QNX and PSOS.

Vanteon is the premier national provider of comprehensive engineering solutions that generate revenue for clients ranging from the Fortune 500 to hot.com start-ups. In our seven centers of engineering excellence, Vanteon has assembled one of the industry's most proficient integrated teams of engineering and consulting professionals in e-business, quality assurance, commercial software development, and hardware and embedded systems. For more information, call 1.800.266.5046 or visit www.vanteon.com.

VeriTest certifies information technology for global e-business through definitive, real-world testing. As the testing arm of Lionbridge Technologies, we cut time-to-market through cost-effective, global processes. Our state-of-the art labs in North America, Europe and Asia offer advanced test architecting, scripting, and execution. Emerging dot.com's and established industry leaders alike come to us for:

Web Testing: We offer load, UI, hostability, compatibility and functional testing of web-deployed applications, using automated and manual test methodologies. Clients include developers, ASPs, site-builders, and e-commerce leaders.

Localization and Internationalization Testing (L10N & I18N) of products as they are globalized for deployment. We offer UI, compatibility and functionality test capabilities in over 27 languages. Our technical and linguistic expertise allows us to work with the industry's most complicated network management and multi-tier web-enabled products.

Functionality and compatibility testing of software and hardware products. We work within a full range of worldwide business and consumer environments, including wireless and mobile infrastructures. Our labs offer professional test planning and a huge library of products for interoperability evaluations. Clients include some of the world's largest software vendors and OEMs.

Certification Testing with leading industry standards in the Windows, Web, Linux, Unix, ASP and advanced multi-platform network infrastructure realms. We pioneered certification testing with Microsoft in 1994, and are now the authorized site for critical software testing programs including Certified for Microsoft Windows 2000. We also run compliance testing programs on behalf of Autodesk, Oracle, and BMC Software.

Contact VeriTest today by U.S. Telephone 310.450.0062, email info@veritest.com <mailto:info@veritest.com> , or see www.veritest.com <http://www.veritest.com> .

Watchfire
135 Michael Cowpland Drive
Suite 400
Kanata, ON K2M 2E9
Tel: (613)599-3888
Fax: (613)599-3826
http://www.watchfire.com

Founded in 1996 as Tetranet Software, Watchfire of Kanata, is the leading provider of quality analysis solutions for websites. Watchfire specializes in the development of best of breed solutions that aid in reaching the highest potential of the web environment as a means of communication and commerce. Automated, scalable solutions make it simple for organizations to ensure their ebusiness sites maintain the most efficient levels of content quality, application reliability, site accessibility and usability.

Watchfire's powerful portfolio of software solutions has received numerous top accolades. These include PC Computing's Most Valuable Product Award for Site Management Software, Network Computing's Well-Connected Award for eCommerce Web Content Management Software. Watchfire has also won support from such key market leaders as Microsoft, Compuware, PSINet and Allaire.

Watchfire was early in recognizing that the promise of the information highway was being seriously undermined by quality control problems. By developing the first generation of website testing solutions, their team of experts responded to the problem.

The intricacy compounded by the amount of quality control problems faced by web developers is increasing alongside the exponential growth of Internet use, the augmenting sophistication of web-based applications, and the ever-expanding complexity and quantity of website content. Watchfire confronts these new market changes with their progressive development of next generation quality analysis solutions, which will allow organizations to ensure that their mission- critical ebusiness websites provide high performance and superlative quality for the customer experience.

Ziff Davis Labs (www.zdlabs.com) is the independent, for-hire testing service of Ziff Davis Publishing. At our facilities in Silicon Valley and North Carolina's Research Triangle area, we perform two distinct but related kinds of work:

- providing top-quality independent, for-hire **testing** of Internet and technology products, and
- developing and distributing for Ziff Davis Publishing its industry-standard **benchmark software**

We provide the trusted, independent testing services you need to move at top speed in the Internet economy. Whether you want us to test your Web site, your desktop or server system, or any other Internet or technology product, we have the expertise, experience, and equipment to do the job.

We live and work at the heart of the Internet economy, testing for companies large and small, established and up-and-coming. We'll put your product--and, if you wish, your competitor's--through its paces. Our experts will give you an objective assessment and warn you about any bugs or potential problems. We'll examine your product from a fresh perspective, go as deeply into the details of its operation as necessary, and put our findings in a meaningful, action-oriented context.

If you'd like to keep our findings to yourself, we'll provide strict confidentiality. If you want to publicize the results, our marketing team can help you leverage our testing and analysis efforts to your best competitive advantage. When your customers see that Ziff Davis Labs has tested your Internet or technology product, they will know they can trust the results.