

INTERNATIONAL SOFTWARE

Quality Week Europe

Brussels, BELGIUM

9 - 13 November 1998

EURO & Y2K: The Industrial Impact

Tutorials Conference

Sponsored by

SR

Software Research



2nd International Software Quality Week Europe

Conferences

Keynote Presentations

Technology Track

Tools & Solutions Track

Process & Management Track

Vendor Technical Track

Extra Presentations

Exhibitors Information

Authors Index

Tutorial Sessions

QWE' 98

SR

2nd International Software Quality Week Europe

Keynote Presentations

Plenary Session

Conference Welcome	Dr. Edward Miller, <i>Software Research</i> Conference Overview
Plenary Keynote #1 (DAY 1)	Dr. John D. Musa, <i>Consultant</i> Applying Operational Profiles to Testing: ISSRE Results
Plenary Keynote #2 (DAY 1)	Mr. Bill Eldridge, <i>Barclays' EURO Advisor</i> EMU: The Impact on Firms' Global Operations
Plenary Keynote #3 (DAY 2)	Mrs. Dorothy G. Graham, <i>Grove Consultants</i> Inspection: Myths and Misconceptions
Plenary Keynote #4 (DAY 2)	Mr. David Talbot, <i>European Commission</i> EU Commission Actions for Y2K and EURO
Plenary Keynote #5 (DAY 3)	Dr. Boris Beizer, <i>Analysis, Inc.</i> Nostradamus Redux
Conference Conclusion (DAY 3)	Dr. Edward Miller, <i>Software Research</i> Lessons Learned QWE'98 Best Paper Award

QWE' 98

SR

2nd International Software Quality Week Europe

Technology Track

Conference Day #1 (1T - 5T)

Wednesday, 11 November, 1998

Conference Day #2 (6T - 10T)

Thursday, 12 November, 1998

Conference Day #3 (11T -12T)

Friday, 13 November, 1998

QWE' 98

SR

2nd International Software Quality Week Europe

Tools & Solutions Track

Conference Day #1 (1S - 5S)

Wednesday, 11 November, 1998

Conference Day #2 (6S - 10S)

Thursday, 12 November, 1998

Conference Day #3 (11S -12S)

Friday, 13 November, 1998

QWE' 98

SR

2nd International Software Quality Week Europe

Process & Management Track

Conference Day #1 (1M - 5M)

Wednesday, 11 November, 1998

Conference Day #2 (6M - 10M)

Thursday, 12 November, 1998

Conference Day #3 (11M -12M)

Friday, 13 November, 1998

QWE' 98

SR

2nd International Software Quality Week Europe

Technology Track

Conference Day #1 (Wednesday, 11 November, 1998)

1T	Mr. Rene Weichselbaum, <i>Frequentis Nachrichtentechnik GesmbH</i> Software Test Automation
2T	Mr. James Clarke, <i>Lucent Technologies</i> Automated Test Generation From a Behavioral Model
3T	Ms. Brigid Haworth, <i>Bournemouth University</i> Adequacy Criteria for Object Testing
4T	Mr. Bill Bently & Mr. Robert V. Binder, <i>mu-Research / RBSC Corporation</i> The Dynamic Information Flow Testing of Objects: When Path Testing Meets Object-Oriented Testing
5T	Dr. Denise Voit & Prof. David Mason, <i>Ryerson Polytechnic University</i> Component Independence for Software System Reliability

QWE' 98

SR

2nd International Software Quality Week Europe

Technology Track

Conference Day #2 (Thursday, 12 November, 1998)

6T	Dr. Linda Rosenberg, Mr. Ted Hammer & L. Hoffman, <i>GSFC NASA / Unisys</i> Testing Metrics for Requirement Quality
7T	Mr. Hans Buwalda, <i>CMG Finance BV</i> Testing with Action Worlds, A Quality approach to (Automated) Software Testing
8T	Mr. Jon Huber, <i>Hewlett Packard</i> Software Defect Analysis: Real World Testing Implications & A Simple Model for Test Process Defect Analysis
9T	Prof. Antonia Bertolino & Ms. E. Marchetti, <i>CNR-IEI</i> A Simple Model to Predict How Many More Failures Will Appear in Testing
10T	Dr. Stacy J. Prowell, <i>Q-Labs, Inc.</i> Impact of Sequence-Based Specification on Statistical Software Testing

2nd International Software Quality Week Europe

Technology Track

Conference Day #3 (Friday, 13 November, 1998)

11T	Dr. Matthias Grochtmann & Mr. Joachim Wegener, <i>Daimler-Benz AG</i> Evolutionary Testing of Temporal Correctness
12T	Ms. Martina Marre, Ms. Monica Bobrowski & Mr. Daniel Yankelevich, <i>Universidad de Buenos Aires</i> A Software Engineering View of Data Quality

QWE' 98

SR

2nd International Software Quality Week Europe

Tools & Solutions Track

Conference Day #1 (Wednesday, 11 November, 1998)

1S	Mr. Manuel Gonzalez, <i>Hewlett Packard</i> System Test Server Through the Web
2S	Mr. Istvan Forgacs & Mr. Akos Hajnal, <i>Hungarian Academy of Sciences</i> Automated Test Data Generation to Solve the Y2K Problem
3S	Mr. Felix Silva, <i>Hewlett Packard</i> Product Quality Profiling: A Practical Model to Capture the Experiences of Software Users
4S	Mr. Otto Vinter, <i>Bruil & Kjaer</i> Improved Requirements Engineering Based On Defect Analysis
5S	Mr. Robert J. Poston, <i>AONIX</i> Making Test Cases From Use Cases Automatically

QWE' 98

SR

2nd International Software Quality Week Europe

Tools & Solutions Track

Conference Day #2 (Thursday, 12 November, 1998)

6S	Dr. Avi Ziv & Dr. Shmuel Ur, <i>IBM Research Lab in Haifa</i> Off-The-Shelf vs. Custom Made Coverage Models, Which Is The One For You?
7S	Mr. Howard Chorney, <i>Process Software Corporation</i> A Practical Approach to Using Software Metrics
8S	Mr. Lionel Briand, Mr. Bernd G. Freimut, Mr. Oliver Laitenberger, Dr. Gunther Ruhe & Ms. Brigitte Klein , <i>Fraunhofer IESE/Allianz Life Assurance</i> Quality Assurance Technologies for the EURO Conversion Industrial Experience at Allianz Life Assurance
9S	Mr. Jakob-Lyng Petersen, <i>ScanRail Consult</i> An Experience In Automatic Verification for Railway Interlocking Systems
10S	Mr. Tom Gilb, <i>Result Planning Limited</i> Risk Management Technology: A rich practical toolkit for identifying, documenting analyzing and coping with project risks.

QWE' 98

SR

2nd International Software Quality Week Europe

Tools & Solutions Track

Conference Day #3 (Friday, 13 November, 1998)

11S	Dr. Peter Liggesmeyer, Mr. Michael Rettelbach & Michael Greiner, Siemens AG Prediction of Project Quality by Applying Stochastical Techniques to Metrics Based on Accounting Data: An Industrial Case Study
12S	Mr. John Corden, <i>CYRANO</i> Year 2000 - Hidden Dangers

QWE' 98

SR

2nd International Software Quality Week Europe

Process & Management Track

Conference Day #1 (Wednesday, 11 November, 1998)

1M	Mr. Leslie A. Little, <i>Aztek Engineering</i> Requirements Management-Simple Tools-Simple Processes
2M	Mr. Nathan Petschenik, <i>Bellcore</i> Year 2000: Catalyst for Better Ongoing Testing
3M	Mr. Juan Jaliff, Mr. Wolfgang Eixelsberger, Mr. Arne Iversen & Mr. Roland Revesjf, <i>ABB</i> Making Industrial Plants Y2K-Ready: Concept and Experience at ABB
4M	Mr. Stale Amland, <i>Avenir (UK) Ltd.</i> Risk Based Testing
5M	Mr. Graham Titterington, <i>Ovum, Ltd</i> A Comparison of the IT Implications of the Y2K and the EURO Issues

QWE' 98

SR

2nd International Software Quality Week Europe

Process & Management Track

Conference Day #2 (Thursday, 12 November, 1998)

6M	Mr. L. Daniel Crowley, <i>IDX Corporation</i> Cost of Quality - The Bottom Line of Quality
7M	Dr. Erik P. Van Veenendaal, <i>Improve Quality Services</i> Questionnaire Based Usability Testing
8M	Mr. Gorka Benguria, Ms. Luisa Escalante, Ms. Elisa Gallo, Ms. Elizabete Ostolaza & Mr. Mikel Vergasa, <i>European Software Institute</i> Staged Model for SPICE: How to Reduce Time to Market - TTM
9M	Dr. Antonio Cicu, Mr. Domenico Tappero Merlo, Mr. Francesco Bonelli, Mr. Fabrizio Conicella & Mr. Fabio Valle, <i>QualityLab Consortium/MetriQs</i> Managing Customer's Requirements in a SME: A Process Improvement Initiative Using a IT-Based Methodoloty and Tool.
10 M	Mr. Thomas Drake, <i>Coastal Research & Technology, Inc.</i> The EURO Conversion - Myth versus Reality? Panelists: Mr. John Corden, Mr. Patrick O'Beirne Mr. Jens Pas, and Mr. Graham Titterington

2nd International Software Quality Week Europe

Process & Management Track

Conference Day #3 (Friday, 13 November, 1998)

11M	Mr. Mark Buenen, GiTek Software n.v. Introducing Structured Testing in a Dynamic, Low-Mature Organisation
12M	Ms. Elisa Gallo, Mr. Pablo Ferrer, Mr. Mikel Vergara & Mr. Chema Sanz, <i>European Software Institute</i> SW CMM Level2: The Hidden Structure

QWE' 98

SR

2nd International Software Quality Week Europe

Vendor Technical Track

1V Day#1	Mr. Charles J. Crawford, <i>Blackstone & Cullen</i> Year 2000 and the EURO: Compliance Testing and Data Management
3V Day#1	Dr. Edward Miller, <i>Software Research, Inc.</i> Remote Testing Technology
5V Day#1	Mr. Gordon Tredgold, <i>The Testing Consultancy.</i> Year 2000 Functional Testing
6V Day#2	Mr. Luc Van Hamme, <i>OM Partners n.v.</i> Results of the ESSI PIE Project OMP/CAST
7V Day#2	Dr. Boudewijn Schokker, <i>VAC Software Engineering</i> Visions and Tools
8V Day#2	Dr. Edward Miller, <i>Software Research, Inc.</i> WebSite Validation Technology: Assuring E-Commerce Quality
9V Day#2	Mr. Bob Bartlett, <i>SIM Group Ltd.</i> Building Re-usable Test Environments for Y2K and EMU / EURO Testing
10V Day#2	Mr. Ido Sarig, <i>Mercury Interactive</i> EMU Conversion - Test Reality Before Reality Tests You...

2nd International Software Quality Week Europe

Extra Presentations

Conference Day #1 and Day #2

1E	Mr. Bogdan Bereza-Jarocinski, <i>ENEA Data AB</i> Is Software Testing Scientific?
2E	Mr. Patrick O'Beirne, <i>Modelling Ltd.</i> Managing Risk in EURO Currency Conversion
3E	Mr. Jens Pas, Ms. Ethel Verbiest, Mr. Wim Blommaert & Mr. Steven Patry, <i>ps_testware</i> Testing the Year 2000
4E	Mr. Richard Tinker & Mr. Ron Walters, <i>BT Labs</i> System Integration and VV&T Strategies

QWE' 98

SR

2nd International Software Quality Week Europe

Exhibitors

Sponsors

Software Research, Inc.

Co-Sponsors:

Gold Sponsors:

CMG Information Technology

SIM Group Ltd.

Silver Sponsors:

IQUIP Informatica B.V.

Mercury Interactive

The Testing Consultancy

Golden Leaf Sponsor:

GiTek Software n.v.

Vendors

Blackstone & Cullen, Inc.

CMG Information Technology

CYRANO (UK) Ltd.

IQUIP Informatica B.V.

McCabe & Associates

McGraw-Hill Publishing Company

Mercury Interactive

OM Partners

SIM Group Ltd.

Software Research, Inc.

VAC Software Engineering

John Wiley & Sons

QWE' 98

SR

Speakers Index

Amland, Staale (4M)
Bartlett, Bob (9V)
Bazzana, Gualtiero (C1)
Beizer, Boris (A1A2)
Beizer, Boris (P3)
Bently, Bill (4T)
Bereza-Jarocinski, B. (1E)
Bertolino, Antonia (9T)
Binder, Robert (F1F2)
Bobrowski, Monica (12T)
Broekman, Bart (E2)
Buenen, Mark (11M)
Buwalda, Hans (7T)
Chorney, Howard (7S)
Cicu, Antonio (9M)
Clarke, James (2T)
Corden, John (12S)
Corden, John (10M)
Crawford, Charles (1V)
Crowley, L. Daniel (6M)
Drake, Thomas (C2)
Drake, Thomas (10M)
Eldrige, Bill (P1)
Forgacs, Istvan (2S)
Freimut, Bernd (8S)
Gilb, Tom (10S)
Gonzalez, Manuel (1S)
Graham, Dorothy (E1)
Graham, Dorothy (P2)
Grochtmann, Matthias (11T)
Hammer, Ted (G1)
Haworth, Brigid (3T)
Huber, Jon (8T)
Jaliff, Juan (3M)
Kent, John (1RT)
Kit, Ed (H1H2)
Liggesmeyer, Peter (11S)
Little, Leslie A. (1M)
Miller, Edward (P1)
Miller, Edward (3V)
Miller, Edward (8V)
Miller, Edward (P3)
Musa, John (G2)
Musa, John (P1)
O'Beirne, Patrick (10M)
O'Beirne, Patrick (2E)
Ostolaza, Elixabete (8M)
Pas, Jens (10M)
Pas, Jens, (3E)
Petersen, Jakob-Lyng (9S)
Petschenik, Nathan (2M)
Pol, Martin (D1D2)
Poston, Robert (5S)
Prowell, Stacy (10T)
Robertson, Suzanne (B1B2)
Rosenberg, Linda (6T)
Sanz, Chema (12M)
Sarig, Ido (10V)
Schokker, Boudewijn (7V)
Shaham-Gafni, Yael (6S)
Silva, Felix (3S)
Talbot, David (P2)
Tinker, Richard (4E)
Titterington, Graham (5M)
Titterington, Graham (10M)
Tredgold, Gordon (5V)
Van hamme, Luc (6V)
VanVeenendaal, Erik (7M)
Vinter, Otto (4S)
Weichselbaum, Rene (1T)
Woit, Denise (5T)

Quality Week Europe

TUTORIAL DAY #1

A1: Dr. Boris Beizer,
Analysis, Inc.

A2: Dr. Boris Beizer,
Analysis, Inc.

B1: Dr. Suzanne Robertson,
The Atlantic Systems Guild

B2: Dr. Suzanne Robertson,
The Atlantic Systems Guild

C1: Dr. Gualtiero Bazzana,
ONION s.r.l.

C2: Mr. Thomas Drake,
Coastal Research & Technology, Inc.

D1: Dr. Martin Pol,
GITEK Software N.V.

D2: Dr. Martin Pol,
GITEK Software N.V.

TUTORIAL DAY #2

E1: Mrs. Dorothy G. Graham,
Software Inspection

E2: Mr. Bart Broekman,
IQUIP Informatica B.V.

F1: Dr. Robert V. Binder,
RBSC Corporation

F2: Dr. Robert V. Binder,
RBSC Corporation

G1: Dr. Linda Rosenberg &
Ted Hammer, *GSFC NASA/Unisys*

G2: Dr. John D. Musa,
Consultant

H1: Mr. Ed Kit,
Software Development Technologies

H2: Mr. Ed Kit,
Software Development Technologies

**The Tutorial Notes and prior Conference Publications are available
from Software Research Institute**

Click here for an order form

**For more information about the Quality Week Conferences series,
please visit our web site at: <http://www.soft.com/QualWeek/>**

2nd International Software Quality Week Europe

QWE'98

**Theme: EURO & Y2K:
The Industrial Impact**

Organized by SR/Institute



Software Research Institute



Advisory Board Members

Dr. Boris Beizer, *Analysis*, USA

Mr. William Bently, *u-Research*, USA

Dr. Antonia Bertolino, *CNR-IEI*, Italy

Mr. Robert Binder, *RBSC Corp.*, USA

Dr. Juris Borzovs, *Univ. of Riga*, Latvia

Ms. Rita Bral, *SR Institute*, USA

Mr. Bart Broekman, *IQUIP*, Netherlands

Mr. Adrian Burr, *tMSc*, England

Mr. Gunther Chrobok, *Siemens*, Germany

Ms. Ann Combelles, *Objectif*, France

Mr. Dirk Craeynest, *OFFIS nv.* Belgium

Mr. Tom Drake, *CRTI*, USA

Mr. Franz Engelmann, *Synlogic*, Switzerland

Mr. John Favaro, *Intecs*, Italy

Prof. Mario Fusani, *IEI/CNR*, Italy

Prof. Marie-Claude Gaudel, *LRI*, France

Dr. Guenter Koch, *ARCS*, Austria

Dr. Peter Liggesmeyer, *Siemens*, Germany

Dr. Edward Miller, *Software Research, Inc.* USA

Prof. Leon Osterweil, *Univ. of Massachusetts*, USA

Dr. Martin Pol, *GITEK*, Belgium

Dr. Suzane Robertson, *Atlantic Systems Guild*, Eng.

Mr. Giuseppe Satriani, *ESI*, Spain

Dr. Torbjorn Skramstad, *MUST*, Norway

Prof. Andreas Spillner, *Hochs.-Bremen*, Germany

Dr. Tor Staalhane, *SINTEF.*, Norway

Dr. Erik V. Veenendaal, *Impr. Qual. Serv. & TUE*. Ne.

Dr. Otto Vinter, *Bruel & Kjaer*, Denmark

Dr. Tony Wasserman, *Software Methods & Tools*, USA



Track Chairs

Boris Beizer, *Analysis, Inc.* USA
Technology Track Chair

Juris Borzovs, *Riga*, Latvia
Tools and Solutions Track Chair

Tobjorn Skramsta, *NUST*, Norway
Process/Management Track Chair

Otto Vinter, *Brel & Kjaer*, Denmark
Vendor Technical Track Chair



Sponsor:



Software Research, Inc. (SR)

Co-sponsors:

Gold Sponsors:



CMG Information Technology



SIM Group Ltd.

Silver Sponsors:



IQIP Informatica B.V.



Mercury Interactive

Leaf Sponsors:



GiTek Software n.v.



Cooperating Organizations



Association for Computing Machinery (ACM)



European Software Institute (ESI)



European Systems and Software Initiative (ESSI)



De Koninklijke Vlaamse Ingenieursvereniging (KVIV)



Studiecentrum voor Automatische Informatieverwerking (SAI)



Applying Operational Profiles to Testing,
with Updates from ISSRE

JOHN D. MUSA

Software Reliability Engineering and Testing Courses

39 Hamilton Road
Morristown, NJ 07960-5341
j.musa@ieee.org

Copyright © 1998 by John D. Musa

ALL RIGHTS RESERVED. No part of this document may be reproduced in any form, hard copy or electronic, without written permission of John D. Musa. No part of any accompanying presentation may be recorded (audio or video) without similar written permission.

Slide 1



‘The Euro and its Impact on Firms’ Global Operations’

Bill Eldridge
E. U. Adviser’s Office, Director.
Barclays Bank PLC

Slide 2



Agenda

- The Details of EMU
- Business Issues

Slide 3



What is EMU?

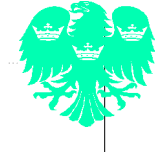
- Single Currency
- One Politically Independent Bank - ECB
- One Monetary Policy

Slide 4



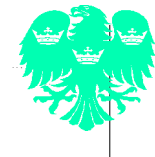
Why?

- Consistency in Single Market
- Reduces Cross Frontier Transaction Costs
- Eliminates Exchange Risk
- Lower Interest Rates
- Political Cohesion



Joining Criteria


- Inflation - 2.7%
- Exchange Rate Stability
- Government Budget Deficit - 3% GDP
- Government Debt - 60% of GDP
- Convergent Interest Rates



Staying In

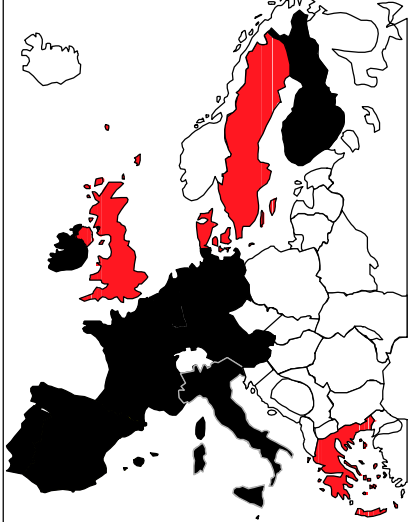
- Stability and Growth Pact
 - Preventative
 - Dissuasive

Candidates



“1st Wave” Countries


- Austria
- Belgium
- Finland
- France
- Germany
- Ireland
- Italy
- Luxembourg
- Netherlands
- Portugal
- Spain



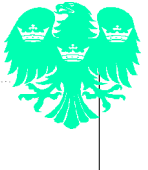
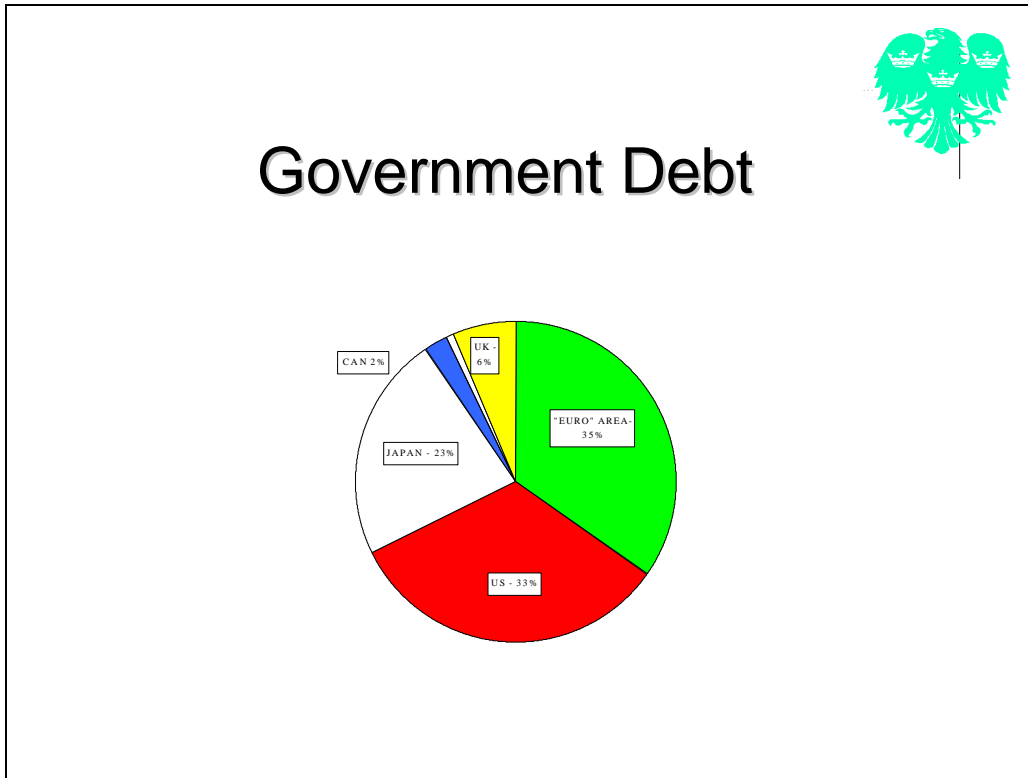
Later countries

- Denmark - opt out
- Greece-convergence
- Sweden-convergence
- UK - choice

Timetable



- 1-3 May Decision Announced
- 1st January 1999 Day One
 - No compulsion, No prohibition
- 1st January 2002 Notes and Coin
- 1st July 2002 End of Legacy Currencies



E. U. Globally

Comparable to USA - GDP, Trade

	<u>% OECD GDP</u>	<u>% Global Trade</u>
EU 15	38	20
USA	31	18
Japan	23	8



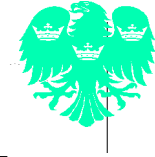
What will it mean for Firms' Global Operations?

- IT an Essential Building Block
- Business Strategy
 - Price
 - Markets
 - Economics
 - New Currency
 - Financing



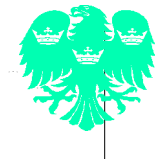
What Does it Mean for EU Banking?

- FX trading
- Systems
- Pan European Benchmarks
- Loss of Domestic Franchise
- Catalyst for Change



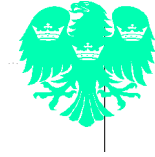
What Does it Mean for UK Banking?

- Ready for 1/1/99
 - Financial Markets
 - Asset Management
 - Corporates
 - European Retail
- UK Retail next wave?



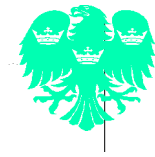
UK Banks and UK Entry

- Lead Time - 3 Years
- Short Transition
- Timetable



What Has Barclays Been Doing?

- 1991 Maastricht Treaty
- 1995 Task Force
- 1997 Customer Road Shows
- 1998 Final Preparations



Conclusions

- Good Objectives
- There is a Price to Pay
- Huge Experiment
- Preparation is Key

Slide 1

Inspection: myths and misconceptions

Dorothy R. Graham

Grove Consultants
Grove House
40 Ryles Park Road
Macclesfield, Cheshire
SK11 8AH U.K.

Tel: +44 1625 616279
Fax: +44 1625 619979
www.grove.co.uk

email: dorothy@grove.co.uk

© Grove Consultants, 1998

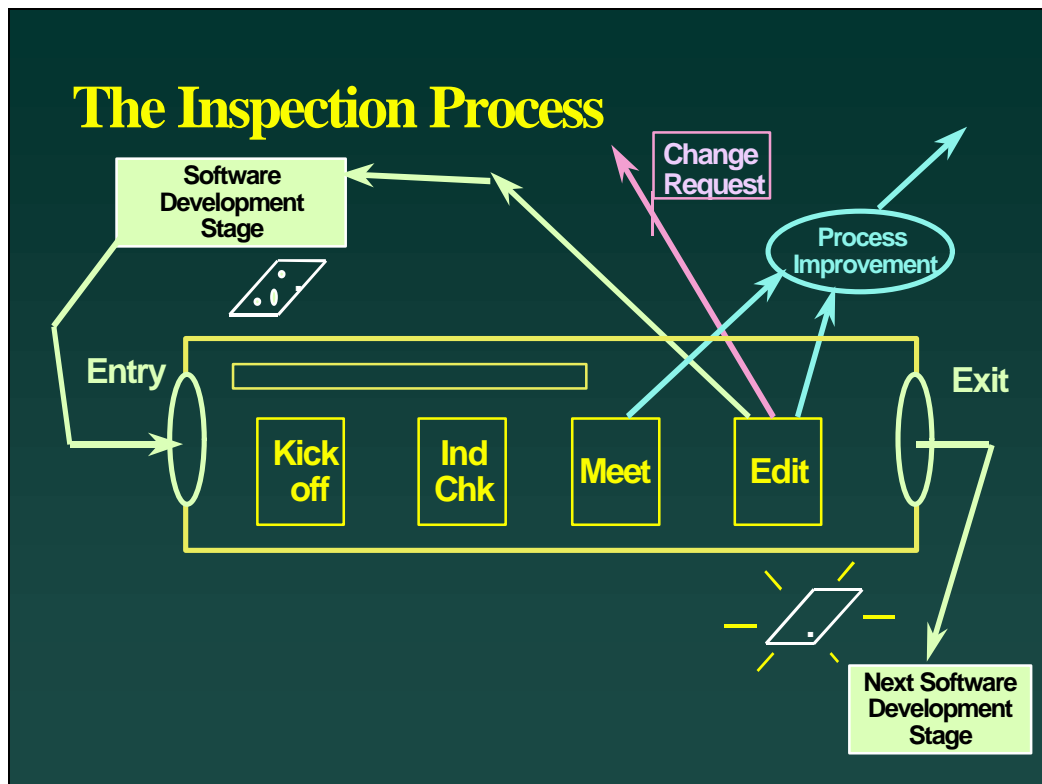
Slide 2

Myths and misconceptions

- **Myth**
 - A plausible story about a supernatural phenomenon
- **Misconception**
 - a false or mistaken view, opinion or attitude

Contents

- What Inspection is
- How Inspection is misunderstood
- What Inspection can & cannot do



Contents

- What Inspection is
- **How Inspection is misunderstood**
 - Inspection is time-consuming & expensive
 - Every page should be Inspected - with limited time, look at more pages / hour
 - Inspection is subjective and detailed (“nit-picking”)
 - The main part is the meeting
 - The main focus is on finding defects
- What Inspection can & cannot do

Benefits of Inspections

- Development productivity improvement
- Reduced development timescales
- Reduced testing time and cost
- Lifetime cost reductions
- Reduced fault levels
- Improved customer relations
- etc.

Expensive?

- **“high-priced, costly”**
 - How long is actually spent doing reviews / Inspections?
- **Compared to what?**
 - What value do they achieve? (quantified)
 - What is the cost of defects NOT found?
- **Are they value for money?**
 - “Expensive” can be much cheaper

Inspections are cost-effective

- 25% reduction in schedules
- remove 80% - 95% of errors at each stage
- 28 times reduction in maintenance cost

- Major conversion project recovered a 4 times slippage (another 3 wks early in 1 yr project)
- A software warranty offered to customers

Contents

- What Inspection is

- **How Inspection is misunderstood**

- Inspection is time-consuming & expensive

- - Every page should be Inspected - with limited time, look at more pages / hour

- Inspection is subjective and detailed (“nit-picking”)

- The main part is the meeting

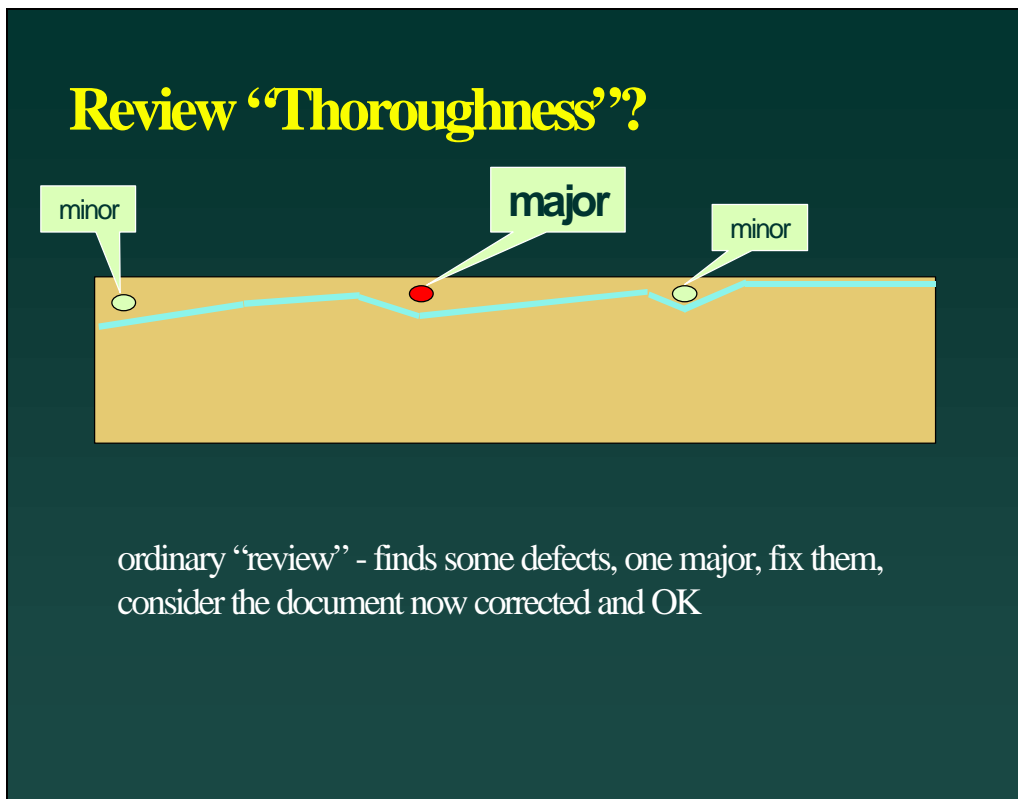
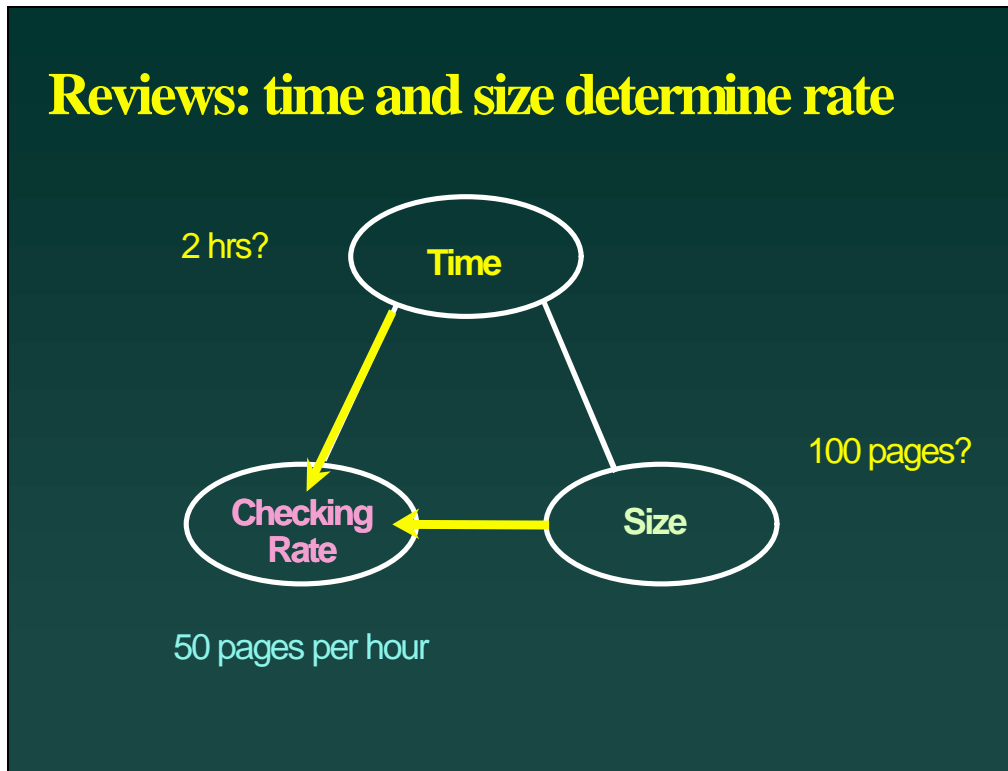
- The main focus is on finding defects

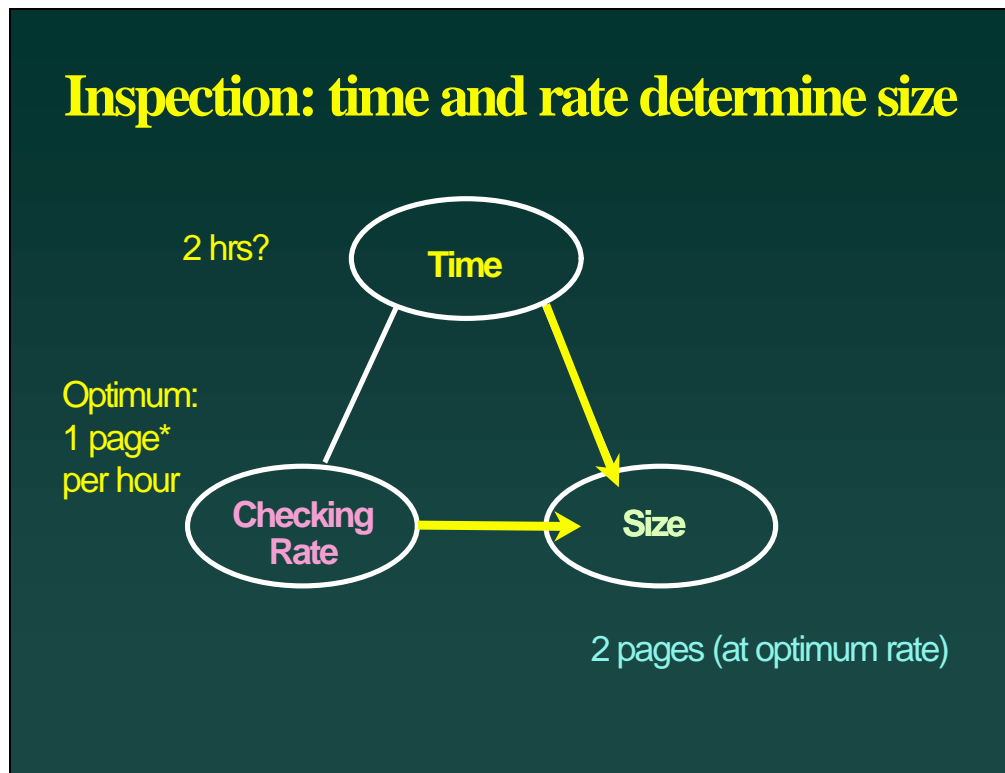
- What Inspection can & cannot do

At first glance ..



Here's a document: review this (or Inspect it)





Inspection Thoroughness

Inspection can find deep-seated defects:
all of that type can be corrected, but needs optimum checking rate

Contents

- What Inspection is

- **How Inspection is misunderstood**

- Inspection is time-consuming & expensive
- Every page should be Inspected - with limited time, look at more pages / hour
- - Inspection is subjective and detailed (“nit-picking”)
- The main part is the meeting
- The main focus is on finding defects

- What Inspection can & cannot do

What are the important defects?

- Defects which

- cause the most severe problems
- cost the most money
- cause greatest embarrassment

- What often gets checked?

- mis-spelings, indentation
- “standards”

Rules: the foundation of Inspection

- **A defect is a potential violation of a Rule**
 - Rules are aimed at major defects
 - Rules are accepted by author
 - Rules make Inspection objective, not subjective
- **Good Rulesets are critical to success**
 - ambiguity, clarity, sources, risks, versions, structure, generality

Contents

- **What Inspection is**
- **How Inspection is misunderstood**
 - Inspection is time-consuming & expensive
 - Every page should be Inspected - with limited time, look at more pages / hour
 - Inspection is subjective and detailed (“nit-picking”)
 - - The main part is the meeting
 - The main focus is on finding defects
- **What Inspection can & cannot do**

“Inspection is a meeting”

- Preparation is probably good to do
- The meeting is 2 hours
- “Discussion meeting” - discuss defects, agree which are real defects, discuss how to fix them

The meeting

- **Minor part of the process**
 - 80% of defects found in checking
- **Only held if economic**
 - value determines duration (may be 0)
- **Highly efficient if held**
 - “no discussion” rule
- **Raise issues, not agree or solve them**
 - power to the editor / author

Contents

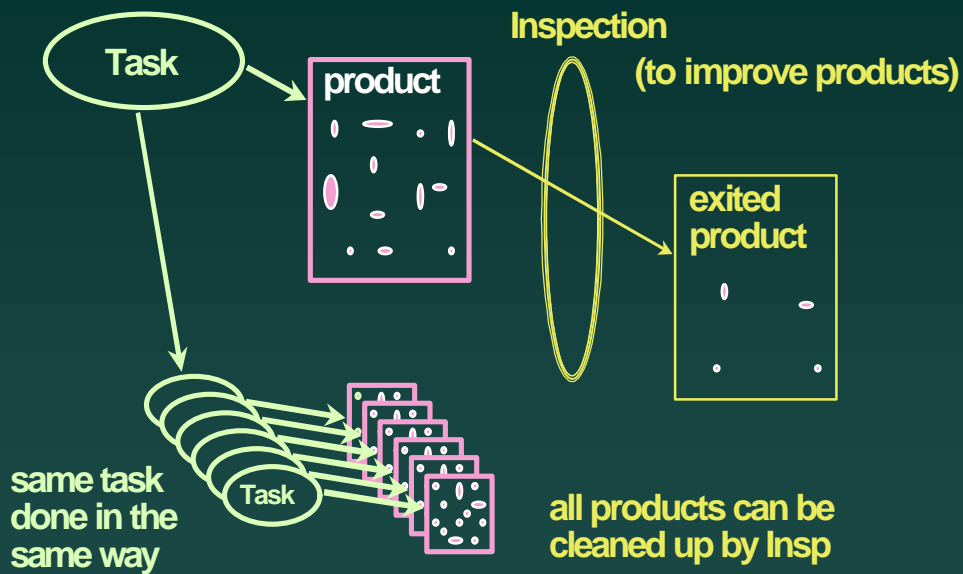
- What Inspection is

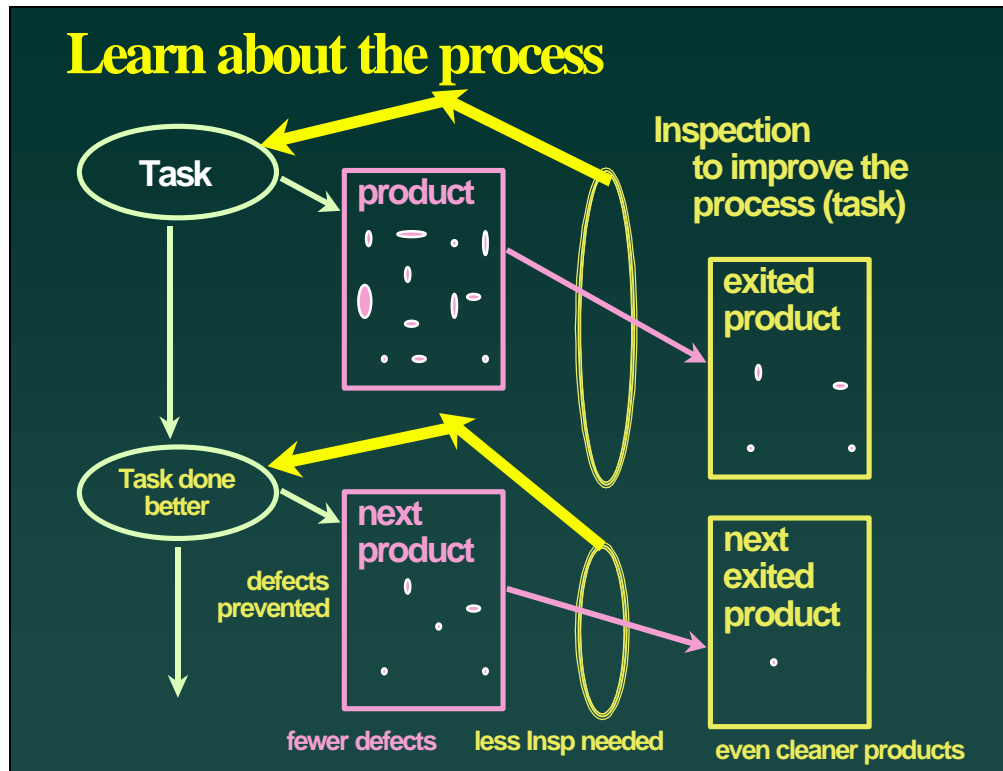
- **How Inspection is misunderstood**

- Inspection is time-consuming & expensive
- Every page should be Inspected - with limited time, look at more pages / hour
- Inspection is subjective and detailed (“nit-picking”)
- The main part is the meeting
- ➔ - The main focus is on finding defects

- What Inspection can & cannot do

Learn about products





Contents

- **What Inspection is**
- **How Inspection is misunderstood**
 - Inspection is time-consuming & expensive
 - Every page should be Inspected - with limited time, look at more pages / hour
 - Inspection is subjective and detailed (“nit-picking”)
 - The main part is the meeting
 - The main focus is on finding defects
- **What Inspection can & cannot do**
 - Limitations of Inspection
 - A myth?

When Inspection will not work

- in a “blame culture”
- manager wants to use Inspection metrics for individual performance evaluation (or rumour)
- deadlines always rewarded, poor quality never penalised

When Inspection will work

- management wants to know the real truth about quality, and really wants to improve
- quality is important to the business
- software development is a defined process, based on written documents

What Inspection can do

- Find deep-seated important defects
- Teach people how to perform their work better
- Kick-start and invigorate a process improvement initiative
- Improve quality and productivity
- Shorten delivery schedules
- Make testing easier to estimate and plan

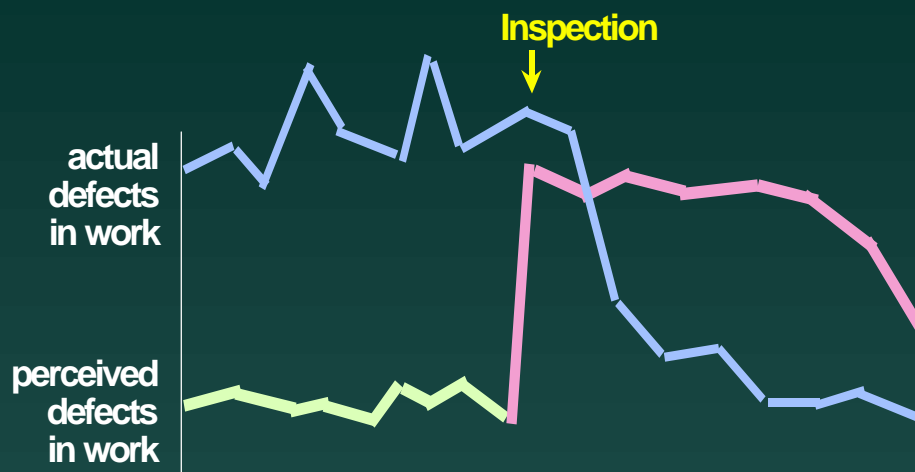
What Inspection cannot do

- Find all defects
 - not economic to be 100% effective
- Replace all other forms of review
 - reviews for decision-making, discussion, walkthroughs for education
- Decide whether this is the right system
 - Inspection can verify, only partially validate (against written sources)
- Inspection is “document-bound”
 - limitation of the technique

A myth?

- **Supernatural influence**
 - it was OK before
 - Inspection disturbed the powers
 - defects suddenly appear
- **Not a myth, another misconception**

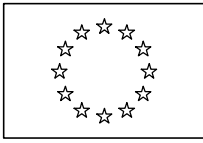
Perception versus reality



Summary: Key Points

- **Inspection is a well-defined, proven technique**
 - to identify major defects in written documents
- **Inspection has many misconceptions**
 - expensive, rates, rules, meeting, defects/process improvement, makes it worse
- **Inspection is document-bound**
 - but is the most cost-effective quality technique
 - (if carried out correctly!)





Software Quality Week Conference – Brussels 12 November 1998

EU Commission actions for Y2K and euro

By: David Talbot

- The two issues have both similarities (at the technical level) but also major differences.
- Amongst the similarities both are:
 - Major challenges for IT management
 - Significant users of scarce human resources
 - World wide in their impact (the euro is not just a European issue, for example 1 trillion ecu's worth of trade is conducted between the EU and US alone)
- However, a recognition of the differences is essential.
 - Y2K is substantially a technical IT matter with possibly profound business impacts in terms of the consequences of any failure to correct the problem; it is essentially a “distress purchase” with limited/little added value.

The euro is essentially a business matter with the potential to transform the business landscape and the way in which an enterprise will operate; this has a clear and significant impact on the enterprises' IT systems; in this regard the adaptation of IT systems is not a “technical” matter, it must be driven by business considerations.

- The Commission position and actions reflect these important differences. The speaker will aim to develop the above points in more detail and outline the “political”, practical and technical steps that have been taken to address these issues.

NOSTRADAMUS REDUX

Dr. Boris Beizer

**Presentation at Quality Week Europe, 1998
Brussels, Belgium, November 13, 1998**

Prepared By:

Boris Beizer, PhD
1232 Glenbrook Road
Huntingdon Valley, PA 19006
PHONE: 215-572-5580
FAX : 215-886-0144
Email: bbeizer@sprintmail.com, bbeizer@acm.org

Prepared For: File

Copyright 1998, Boris Beizer

This is a notice of copyright. No part of this document may be reproduced or converted to any other form by any electronic, manual, and/or mechanical means, including but not limited to: photocopy, recording, taping, facsimile transmission, scanning, storage in a computer and/or memory and/or any other storage media--without the written permission of the author. The material therein remains the sole intellectual property of the author who retains all beneficial rights thereto.

1. Nostradamus Redux

We live in confusing and troubled times: who better to guide us than Nostradamus? Some newly discovered predictions from the missing parts of his section VII, and some new translation of previous ones are provided here for your entertainment and amusement. My interjections for the sake of clarification are in brackets. Nostradamus' predictions take the form of four-line stanzas called "quatrains." He often bundles several predictions within one quatrain and sometimes a prediction is given in parts of several different quatrains.

Where appropriate, I have edited out the parts that do not concern us and merged parts to make the predictions more sensible. Nostradamus is inconsistent when it comes to dates: sometimes he refers to years after his birth, sometimes to years after publication of his predictions, and sometimes to absolute years in the common calendar. I have adjusted all dates to the common calendar.

Nostradamus, viewing our times through the eyes of the 16th century, did not have words for ideas such as "president" or "software bug." So we must interpret "king" to mean any leader, "country" or "kingdom" to also apply to corporations, and any kind of insect to mean software bug.

2. Y2K

The Internet newsgroup, comp.software.year-2000 is (in)famous for its gloomy Y2K predictions. Nostradamus foresaw the Y2K bug, the associated social problems, the resolution, and the aftermath. Nostradamus is certainly as credible as most of the Y2K predictions we hear these days.

III/34. Then when the eclipse of the sun
Will in broad daylight the monster [Y2K bugs] be seen.
It [year representation] will be interpreted quite differently [*that's the crux of the problem, isn't it?*];
They will not care about expense, none will have provided for it.

Is there any doubt that this refers to the Y2K problem? And how about...

I/22 A thing existing without any senses [*obviously he means computer programs*]
Will cause its own end to happen through artifice [*good description of an ABEND*].
I/44. In a short time sacrifices [*taxes*] will be resumed,
Those opposed will be put to death like martyrs.

The remediation effort of the United States Internal Revenue Service does not succeed at first—plagued by ABENDs. The resulting chaos prompts a short-lived tax revolt. But the software is repaired and tax collection resumed. Those who did not pay their taxes are dealt with severely. Nostradamus isn't always to be taken literally; e.g., "put to death..."; but then considering how tax departments often behave, I'm not sure that the literal interpretation isn't correct.

VII-14 He will come to expose the false topography,
The urns of the tomb will be opened.
Sect and holy philosophy to thrive,
Black for White and the new for the old.

This is a very detailed predictions that refers to my talk on Y2K at the 1998 Quality Week Conference. "He" refers to Boris Beizer. The key point is in the last line which clearly refers to remediation of legacy code and the fact that for Y2K testing, we must rely mainly on behavioral (i.e., Black Box testing) and forego detailed coverage testing.

I/47 The speeches of Lake Lemman will become angered,
The days will drag out into weeks,
Then months, then years, then all will fail [*Sounds like Y2K, doesn't it?*].
The authorities will condemn their useless powers.

IV/13 New of the great loss is brought;
The report will astonish the camp.
IV/9 When Geneva in trouble and distress

V/85 Is betrayed by the Swiss.
Through the Swiss and surrounding areas
They will war because of the clouds [*of bugs*].
A swarm of locusts and gnats [*Nostradamus for "bug"*],
The faults of Geneva will be laid quite bare.

I/61 The wretched, republic will be ruined
By a new authority.
The great amount of ill will accumulated in exile
Will make the Swiss break their important agreement.

An emergency global Y2K political conference is held in Geneva, Switzerland. It is a rancorous conference, in part as a result of all the delegates being unable to get their credit cards accepted—which is a special hardship in Switzerland. Meanwhile, remediation drags out for weeks, months, and even years — but there are still massive software failures at the end. The politicians are frustrated by their inability to find political solutions to the problem. The collapse of the global monetary system is attributed to Y2K bugs in Swiss banking software. Switzerland closes their international banks and cancels all agreements.

X/72 In the year 1999 and seven months,
From the sky will come the great King of Terror.
I/80 Then a monster will be born of a very hideous beast:
In March, April, May and June great wounding and worrying.

Dire predictions for July 1999, which is when we can expect many of the first Y2K bugs to strike. Expect the Y2K problems to peak between March-June 2000, with even greater impact than previously expected — the use of “then” makes it clear that he’s talking about the following year (2000).

VI/8 Those who were in the kingdom for knowledge
Will become impoverished by a royal change.
Some exiled without support, having no gold,
Neither learning nor the learned will be held of much value.

Nostradamus tells us that there will be a new administration in the US, following the 1999 election but he doesn’t tell us which party will win. However, whatever the party, the new president puts the blame on the data processing community and initiates harsh measures in punishment. We can certainly expect this kind of behavior on the part of politicians who have to have someone to blame and who better to blame than programmers and SQA people. A big drop in programmer salaries after Y2K. A note of caution by the seer for Y2K consultants—Don’t expect the current billing levels to hold forever.

VI/2 In the year one-thousand, nine-hundred and eighty more or less
One will await a very strange century.
In the year two-thousand and three,
The skies as witness that several kingdoms (one to five) will make a change.

Although Y2k remediation began in the early 1980's he predicts that by 2003 only 1 in 5 countries will have completed the task.

VII/44 After a generation and a bit, from a few bits
Anew a plague of midges, mites, and locusts rises from Xanthus
To plague the unwary and uncaring
Who heeded not the previous holocaust.

The timing makes this right for the UNIX 2028 rollover bug. “Xanthus” is probably an anagram for “UNIX.” There it is all over again. Just like Y2K, the problem will be ignored.

3. Euro

Conversion to the Euro, coming as it does simultaneously with Y2K is another problem that was much on Nostradamus’s mind.

I/40 The false trumpet concealing madness [*apparently his opinions of the Euro*]
Will cause Byzantium to change its laws.
From Egypt there will go forth a man
Who wants the edict withdrawn, changing money and standards.

Turkey and Egypt are conditionally accepted into the European Economic Union. Turkey willingly changes monetary policies in order to achieve compliance, but Egypt objects to the schedule because it cannot change its software in time and in accordance to the specified standards.

IV/48 The plains of Europe, rich and wide,
Will produce so many gadflies and grasshoppers [*bugs*]
That the light of the sun will be clouded over.
Devouring everything, a great pestilence will come from them.

Oh-oh! Looks like there's going to be a lot of turmoil from the Euro conversion.

I/73 France shall be accused of neglect by her five partners ... [*What else is new?*].
The Outer Three will adamant remain, [*Norway, Denmark, and Sweden, obviously*]
IV-21 The change will be very difficult.
Both city and province will gain by it.

Business as usual in EEU. Conversion will not be easy, but all's well that ends well.

4. The Computer Industry, Microsoft, Gates

Nostradamus is credited with predicting Napoleon and Hitler. You didn't think he would leave out Microsoft and Bill Gates, did you?

IV/31 The moon, in the middle of the night of the high mountain
The young wise man alone with brains has seen it.
Invited by his disciples to become immortal,
His eyes to the south, his hands on his breast, his body in the fire.

Who else but Bill Gates? But what about that "body in the fire?" Perhaps the following quatrain sheds light on this.

VII/45 The Newest Testament [*Anagram for NT?*] again delayed
Held closeted in secrets deep.
The multitudes play into his wily hands
By their intemperate impatience.

By once again delaying the release of WIN NT5, Gates assures total saturation of WIN98, capturing the last holdouts under DOS, WIN 3.1, and WIN 95 who are forced into WIN 98 as the only Y2K compliant operating system. But Gates takes a lot of heat over this ploy ("his body in the fire").

V/75 He will rise high over his wealth, move to the right,
He will remain seated on the square stone;
Towards the south placed at the Window,
A crooked staff in his hands, his mouth sealed.

Gates will get even richer and more powerful. As he does, he will increasingly adopt conservative politics. The use of Windows will increase significantly in Africa and South America as the next big market for personal computers. More trouble from the US Justice Department because of predatory business practices by his staff (a staff which he completely controls). Gates will refuse to testify before a grand jury.

VII/61 From Gates through the Window, air and wheels combine,
The aging sage foresees the union.
Timid judges rise in righteous anger,

But soon to sleep they go again.

An aging Bill Gates argues before the U.S. supreme court that the absorption of United Airlines by Microsoft (“air and wheels combine”) is a natural evolution of the operating system (WIN NT8.0) that began with virtual flight in Flight Simulator XXIII. Similarly, the proposed absorption of General Motors is argued as an inevitable evolution from virtual travel on the Internet to physical travel. The US Supreme Court judges make politically correct verbal opposition but let the case die when they refuse to provide an opinion on the question.

VII/52 The antipodal names inverted
The king, his colors true revealed.
Leviathan, his mighty gorge extended,
Sweeps the smaller sea.

Microsoft’s name is changed to “Megahard.” “We’re no longer “micro” and we were never “soft.”, Gates explains. “The new name is in keeping with what we have always been.” Megahard buys the entire list of the NASDAQ stock exchange.

II/89 One day the two great leaders will be friends;
Their great power will be seen to grow.
The new land will be at the height of its power,
To the man of blood the number is reported.

Microsoft and Oracle merge as Gates and Ellison shake hands. “The man of blood”(Gates) is the new CEO of the combined companies who asks for an immediate financial statement.

IV/75 He who was ready to fight will desert,
The chief adversary will win the victory.
The rear guard will make a defense,
But will falter and die.

Lou Gerstner (IBM’s CEO) fights a hostile takeover bid by Microsoft, but eventually bails out with his golden parachute. The Lotus Notes loyalists attempt to take their product private, but do not succeed.

II/11 The following son the elder will succeed,
Very greatly raised to a kingdom of privilege.
His bitter renown will be feared by all,
But his children will be thrown out of the kingdom.

Bill Gates IV, succeeds Bill Gates III as chairman of Macrohard/IBM/Oracle/UNITED/GM (known in the industry as “MACROMUG”). He botches the job by changing the company into a repressive hell more regimented than IBM at its worst. Eventually, the next generation (Bill Gates V) loses control of the conglomerate.

5. Clinton, American politics, and L’affair Lewinski

As of the time of this writing (August 4, 1998) the Clinton/Lewinski affair is still unresolved. Apparently, like many American politicians (in the opposition party) Nostradamus believed it to be one of the great scandals of history. We have already seen some of these, and others have been hinted about.

VIII/23 Letters are found in the queen’s chests,
No signature and no name of the author.
The ruse will conceal the offers;
So that they do not know who the lover is.

Sounds familiar? Anonymous letters that possibly are an invitation by Clinton to an intern for a very private meeting are found in Hillary Clinton’s files; but key sections are missing so that it’s impossible to prove to whom the letters refer or who wrote them.

IV/57 Ignorant envy supported by the great king,
He will propose forbidding the writings.
His wife, not his wife, [Hillary?] tempted by another,

No longer will the double-dealing couple protest against it.
Clinton proposes a cover up by censorship. Hillary is tempted to have her own affair in retribution — but, finally, the letters are turned over to the grand jury as the Clintons agree to their release.

VIII/14 The offence of the adulterer will become known
Which will occur to his great dishonor.
VIII/95 The seducer will be placed in a ditch
And will be tied up for some time.

A clear reference to the final resolution of the Clinton/Lewinski affair. Clinton certainly seems to be “tied-up and in a ditch” over this affair.

VI/59 The lady, furious in an adulterous rage,
Will come to conspire to not speak to her Prince.
But the culprit will soon be known,
So that seventeen will be martyred.

This happened a while ago. Clinton white house. Hillary is furious and won't speak to Clinton. The news gets out. Nostradamus has Lewinski's age wrong: She was older than seventeen.

VI/72 Through feigned fury of a divine emotion
The wife of the great one will be badly violated.
The judges wishing to condemn such a doctrine,
The victim is sacrificed to the ignorant people.

Trouble for Hillary. She acts furious about Clinton's disclosures. It's an act, because she knew about it all along. Nevertheless, she has to take a lot of abuse for speaking out. The supreme court wants to act against the president. Clinton turns it about by getting Lewinski indicted. Special prosecutor Starr backs off from the immunity pledge and Lewinski is charged with perjury.

VI/13 A doubtful one will not come far from the kingdom,
The greater part will wish to support him.
A Capitol will not want him to reign:
He will not be able to bear his great burden.

Despite everything, Clinton's popularity in the polls continues. Congress wants to impeach him. He considers resigning.

X/76 The Senate will see the parade for one
Who afterwards will be driven out, vanquished.
His adherents will be there at the sound of a trumpet,
Their possessions for sale, the enemies driven out.

Oh, oh! It looks like the US senate will successfully impeach Clinton, after all. His whole administration will go out with him.

6. Software Quality, QA, and Testing

The sage foresaw not only the software industry and bugs, but also that we would be holding conferences about the subject.

IV/26 The great swarm of bees [*bugs*] will arise
But no one will know whence they have come, [*that's pretty typical*]
The ambush by night, the sentinel under the vines,
A city handed over by tongues not naked.

This is all about bugs striking a municipal government's software—or is Nostradamus giving warning for all municipal software? He says that QA has failed in its task — “the sentinel under the vines” means that QA was drunk.

V/93 When Mercury is at the height of his powers,
VII/72 Masters of the heavenly arts in Brussels gather
V/37 Three hundred will be of one agreement
And accord to the execution of their ends.

This is obviously about the QWE98 conference. There is consensus over how things should be done to improve software quality.

IV/18 Some of the most learned men in the heavenly arts
Will be reprimanded by ignorant Princes;
Punished by edicts, driven out as scoundrels,
And put to death wherever they are found.

We all know this scenario. How many of us in SQA have been there?

IV/53 The fugitives and the banished are recalled.
IV/69 The exiles will hold the great city.
They will promise to show them the entrance
By untrodden paths.

But eventually, the powers that be come to their senses and turn control over to people who know quality assurance. The QA leaders and doers will show them the way to get things done. Note here that the key (the entrance) is to assure 100% coverage (untrodden paths). But Nostradamus has it wrong. He's calling for 100% path cover instead of 100% branch cover. But hey, that's a common mistake among people today who should know better.

VI/17 After the penances are burned the ass drivers
Will be forced to change into different clothing.
Those of Saturn burnt by the Millers,
Except the greater part which will not be covered.

Eventually, 100% branch cover as fervently espoused by Edward Miller for so many years will be officially adopted but in practice, most of the code will still not be tested (covered).

III/67 A new set of philosopher, despising gold and riches
Will not be limited, even by mountains;
In their following will be crowds and support.
IV/16 From hundreds they will become thousands.

This is obviously about QW Europe. We must "despise gold and riches" because we're not getting any — anyhow, it speaks well for the results of the message. Great supporting crowds going from hundreds to thousands.

Slide 1

Genetic Algorithms

Perfectly suited for Software Test Automation

Dipl. Ing. René Weichselbaum

COPYRIGHT FREQUENTIS 1998 Rev. 1.0
File: qwe98v10.PPT Page: 1 Author: René Weichselbaum
QFM 01140, Rev. 3

Slide 2

Outline

- Testing activities
- Goals/concept
- The tool: GATester
- The algorithm and its performance
- Software reliability considerations
- Benefits
- Future work

COPYRIGHT FREQUENTIS 1998 Rev. 1.0
File: qwe98v10.PPT Page: 2 Author: René Weichselbaum
QFM 01140, Rev. 3

Testing Activities

- Plan
- Determine
- Refine

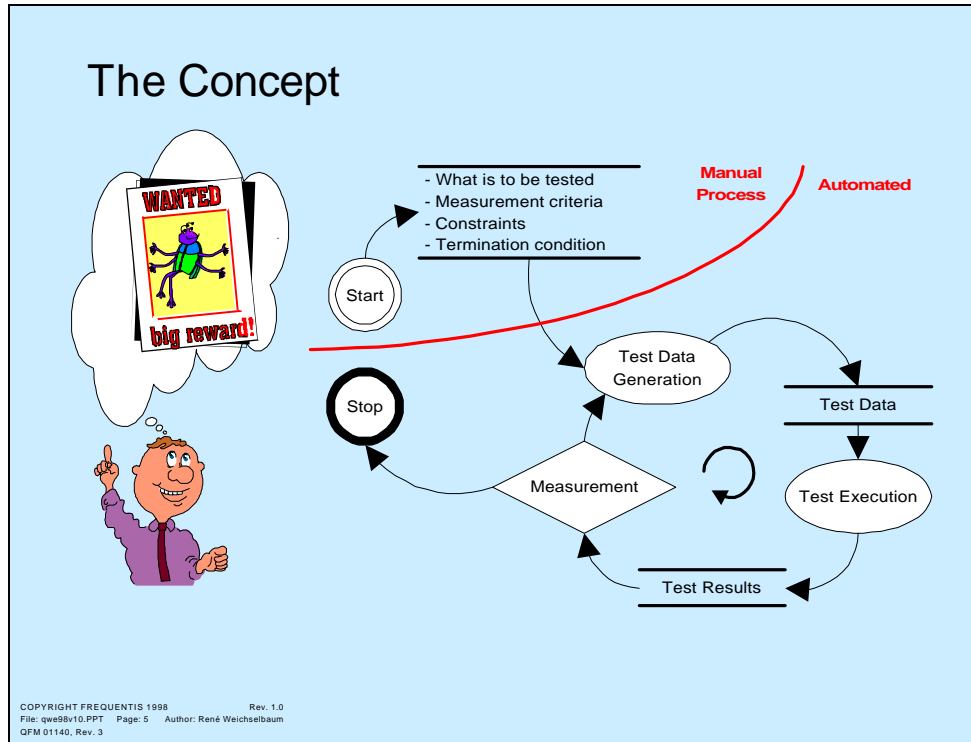
- Design
- Implement

- Execute
- Check
- Evaluate

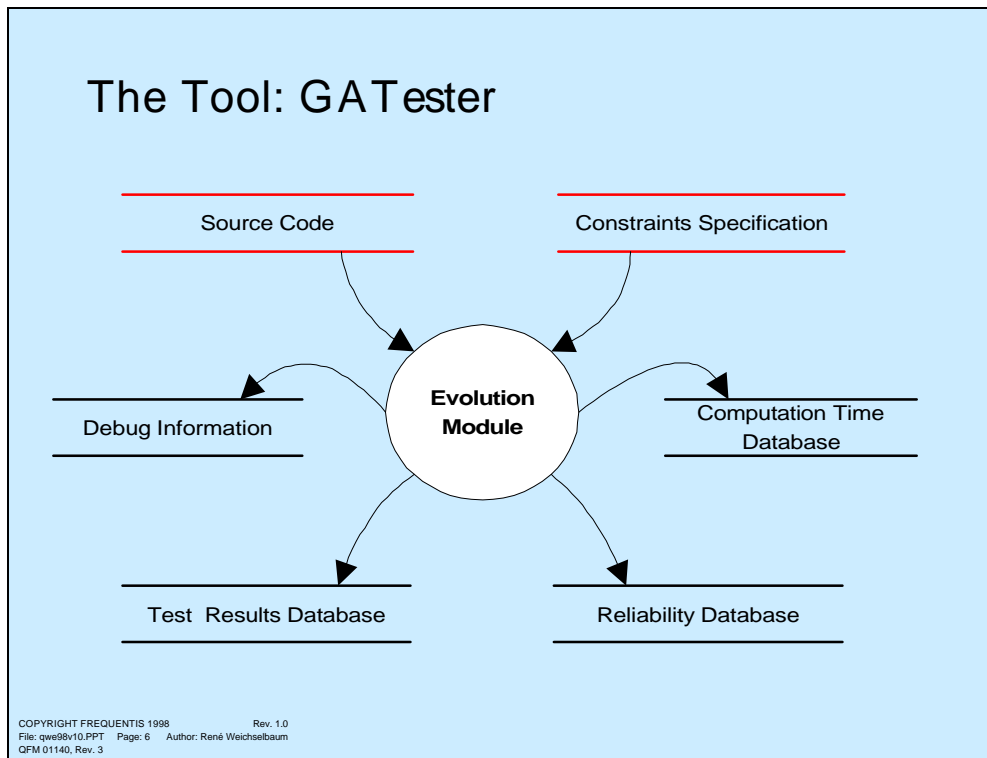
Main Goals

- Automate test data generation
- Sidestep problems encountered by traditional approaches
- Make it repeatable
- Implement detailed reporting procedures
- FAT & SAT forecast

Slide 5



Slide 6



GATester's Workflow

- Identify software units
- Choose a coverage criterion (opt.)
- Check the termination condition (opt.)
- Specify constraints (opt.)
- Provide user defined reliability requirements (opt.)
- "Touch the button"

Performance Considerations

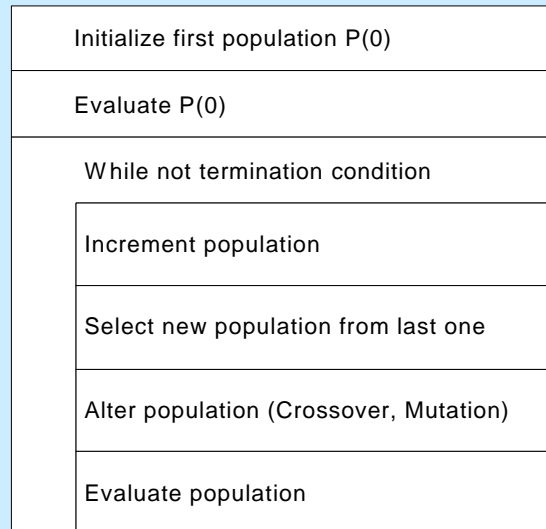
- 586, 166 MHz, 32 MB RAM, Linux
- 100% statement coverage

Computation Time in clocks	Statement feasibility
149	1,53E-05
245	5,96E-08
9546	2,33E-10
379394	9,09E-13

Terminology

- **Gene**
A gene controls the inheritance of one or several characters. We use it as the basic unit storing a variable's test data.
- **Chromosome**
A Chromosome is made of genes, arranged in linear succession.
- **Population**
A population is a set of chromosomes.
- **GA = Genetic Algorithm**

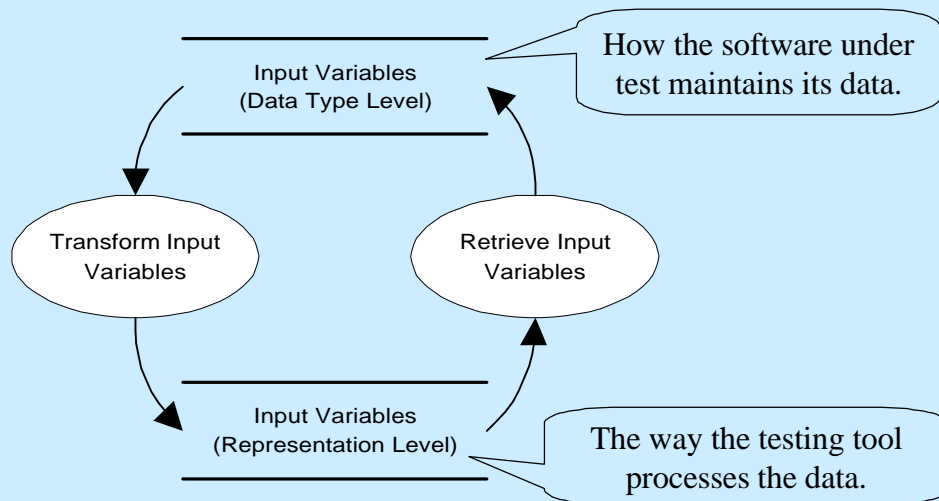
The Genetic Algorithm

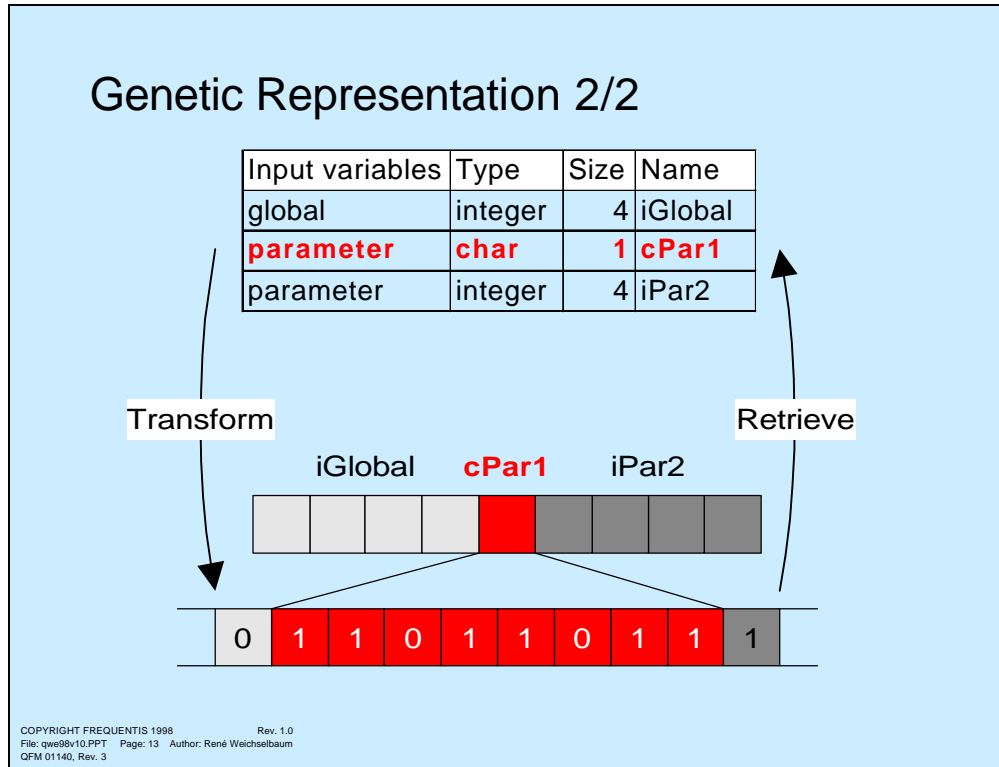


GA 's Components

- Genetic representation for potential solutions
- A way to create an initial population of potential solutions
- An evaluation function, rating solutions in terms of their "fitness"
- Genetic operators that alter the composition of children
- Values for various parameters

Genetic Representation 1/2



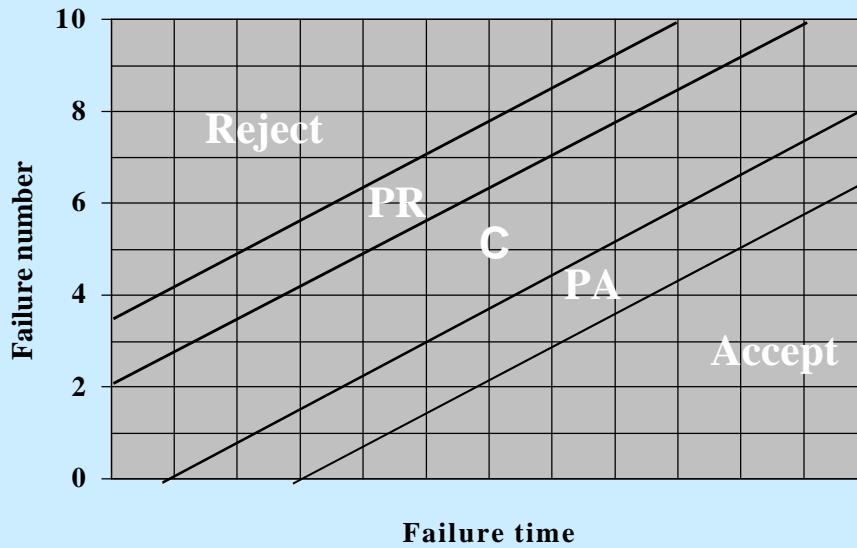


- ## Why do GAs work?
- Building blocks
 - Schema
 - Order of a schema
 - Defining length of a schema
 - Schema theorem
- COPYRIGHT FREQUENTIS 1998 Rev. 1.0
 File: qwe98v10.PPT Page: 14 Author: René Weichselbaum
 QFM 01140, Rev. 3

Software Reliability

- Definition
- Qualification testing vs. reliability testing
- Operational profile
- Saturation point
- System reliability

Unit Reliability Measurement Chart



Unit Run Reliability

- Definition
 - Assumption: uniform probability distribution
 - $R_k = nc/NR$
- Primitives
 - nc...number of correct runs
 - NR...number of total runs
- Input space handling
 - Evolution strategy

Benefits

- Automated test data generation
- Repeatable testing process
 - Automated regression tests
 - No test data maintenance effort needed
- Thorough testing
- Achieving a level of coverage automatically
- Software reliability assessment

Future Work

- GA configuration
- Self checking test cases
- Software reliability estimation/prediction

Genetic Algorithms

Perfectly suited for Software Test Automation

René Weichselbaum

FREQUENTIS GesmbH
Software Quality Management
Spittelbreitengasse 34, A-1120 Vienna, Austria
Tel: +43 1 81150 1210
Fax: +43 1 81150 1299
e-mail: rweichse@frequentis.com

Copyright 1998 FREQUENTIS. All rights reserved.

Abstract

One of the main problems with automating software testing is its complexity. Genetic algorithms aim at such complex problems. For example, they address the problem of test data generation without instructing them, step by step, on how to do it. Instead of this, their learning algorithm is inspired by the theory of evolution. Using this approach neatly sidesteps many of the problems encountered by other systems in attempting to automate the test process. This paper describes a test tool performing automatic coverage testing by means of genetic algorithms including some key issues in software reliability assessment.

Keywords:

Software testing, test data generation, coverage testing, software reliability, genetic algorithms

1. Introduction

Software testing is an expensive component of software development. In safety related applications it can take up to 80% of the costs of the software development.

Automation seems to be an essential ingredient for both a cost-effective approach to testing and a thorough approach. Manual software testing is very time consuming. The same procedures have to be repeated again and again for every release and every regression test. So it is a good idea to automate these procedures. There are activities being relatively easy to automate, especially test execution and the final check of the results against given rules. Identifying test conditions, designing test cases and implementing the tests are the more intellectual parts of software testing. It is difficult to automate them at all.

This paper describes a test tool performing automatic coverage testing by means of genetic algorithms. Without qualification, coverage usually means branch or statement coverage [1]. Statement coverage, also called C0 coverage, is a metric of the number of source language statements executed under test. Branch coverage, also called C1 coverage, measures the number of branch alternatives executed. Usually these two coverage criteria are accepted as the minimum mandatory testing requirement. However, for some applications condition combination coverage is required. This strategy is stronger than C1 because it requires that all combinations of conditions in every decision statement must take both true and false values. As an example, Figure 1-1 shows the test cases needed for a given decision statement.

```
/* Comment: T...true, F...false
 *   Test cases with the following constraints must be found:
 *   TTT, TTF, TFT, FTT, TFF, FTF, FFT, FFF
 */
IF (a < b) AND (a < c) AND (b < c)
    a = b
```

Figure 1-1: Test Cases required for condition combination coverage.

The tool was developed by the author himself within the scope of his Ph.D. work and is called GATester. GATester is implemented on Linux operating system and primarily intended to support unit and module testing including regression testing.

The main goals of the project are as follows:

- Find a new approach for automating the test process in order to sidestep many of the problems encountered by traditional approaches
- Automate test data generation in order to decrease costs of building new tests
- Make the test data generation task repeatable in order to compare actual results with previous ones
- Make the automated test process understandable by implementing a detailed reporting procedure

In section 2, the testing process is divided into its generic tasks. Section 2 then discusses which of these tasks can be partly or fully automated and which cannot.

In section 3, the steps involved in the proposed testing process are described.

Section 4 presents first performance measures and gives an idea of the effectiveness of this new approach.

In section 5 and 6, the author discusses the basic algorithms performing test data generation and test data evaluation, respectively. These sections should reveal why genetic algorithms are perfectly suited for software test automation.

Section 7 investigates how the test tool can contribute to software reliability assessment calculations.

Finally, the author's conclusions are presented and further research is outlined.

2. The Testing Process

According to [37], a test unit is a set of one or more computer program modules together with associated control data, usage procedures, and operating procedures that satisfy the following conditions:

- All modules are taken from a single computer program
- At least one of the new or changed modules in the set has not completed the unit test
- The set of modules together with its associated data and procedures are the sole object of a testing process

In Figure 2-1 the activities involved in a standard unit testing process are presented [37]. The last column indicates if an activity must be carried out by hand or if it can be automated by GATester. "Partially Automated" means that there exist default values for this activity in order to proceed automatically. However, the user may perform additional configuration steps within this activity, e.g. choosing a particular coverage criterion as a termination condition. Note that only activities concerning coverage testing are considered here. Tasks that add unnecessary costs or do not add value are eliminated.

1	Perform Test Planning Phase	
1A	Plan general approach, resources, and schedule	Manual process
1B	Determine features to be tested	Partially Automated
1C	Refine the general plan	Deleted
2	Acquire test set phase	
2A	Design the set of tests	Automated
2B	Implement the refined plan and design	Automated
3	Measure test unit phase	
3A	Execute the test procedures	Automated
3B	Check for termination	Automated
3C	Evaluate the test effort and unit	Partially Automated

Figure 2-1: The unit test activities according to ANSI/IEEE Std 1008-1987 applied to coverage testing using GATester

Under normal conditions, these activities are sequentially initiated except for an Execute and Check cycle [37], no matter if they are automated or not.

Apart from the fact that some configuration steps must be taken, the process may be fully automated if the goal is to satisfy some level of coverage. For example, if the test should stop after reaching 100% statement coverage, the only manual activity is to start GATester and provide the tool with the program, i.e. the unit that is to be tested, and the information telling the tool which coverage is to be met. The tool returns the test cases needed and some additional output like the computation time. More details about the workflow are presented in the next section.

If also the dynamic nature of software behaviour is of interest the user can provide GATester with some additional constraints. These constraints are treated as software requirements that must be satisfied during test execution. Every constraint must be assigned to one or more statements. GATester will report any violations of given constraints. The next section addresses this issue in more detail.

3. How to use the Test Tool

This chapter describes the GATester's basic functionality from the user's point of view. Although it is far away from being a user guide, it illustrates how to use the tool. In Figure 3-1 the user's tasks are presented. Please refer to Figure 2-1 for the relationship between the ID-column and the corresponding test activity. Deleted or fully automated activities are eliminated here.

ID	The user's tasks
1A	Identify units that are to be tested, choose a coverage criterion, specify a new termination condition if the default one is not appropriate, and optionally provide GATester also with user defined reliability requirements
1B	Specify constraints if the dynamic nature of the software under test should also be evaluated
3C	Complete the provided test summary report if necessary, ensure that the testing products are collected, organized, and stored for reference and reuse

Figure 3-1: The GATester's workflow

Identify units that are to be tested and choose a coverage criterion:

The user provides a unit that is to be tested and selects a coverage criterion. This may be one of the criteria presented in the first chapter, namely statement coverage, branch coverage, or condition combination coverage. The default criterion is statement coverage. If the primary goal is to meet the coverage criterion by some test data the user's tasks are completed. However, it is recommended to perform also the next step, namely checking the termination condition. After a while, usually within a few seconds (see also chapter 4), the tool will return the test cases needed to meet the desired coverage and additional documentation, for example the paths traversed, for every statement the number it has been executed under some test, and all the time intervals used for finding "a better test case" than the best so far. For details about how to interpret these time intervals please refer to section 5.

Check the termination condition:

After choosing an appropriate coverage criterion the user should check the termination condition. The termination condition is a function of the maximum number of test cases specified and the chosen coverage criterion. If either the maximum number of test cases reached or the specified coverage criterion is met the test execution will stop. For example, if the chosen coverage criterion is very hard to meet the user must allow the testing tool to generate "very much" test data.

Provide user defined reliability requirements:

Since software reliability assessment is a very advanced topic that cannot be introduced by a few words it is left out here and discussed in chapter 7.

Specify constraints:

The tool allows to assign any expression to a set of statements. Precisely speaking, the constraints are composed of variables, parentheses, and programming language operators including user defined functions. Constraints evaluate to one of the Boolean values TRUE or FALSE. During test execution, GATester reports all violations of one of the given constraints. Figure 3-2 gives some examples of constraints (representation: C programming language):

```
(a<b)&&(c == d) "all"  
a==3 1,2,3  
a=2 6
```

Figure 3-2: Some examples of constraints

Basically, a constraint specification consists of the constraint itself and a definition telling the tool for which statements the constraint is relevant. For example, `a==3` is the constraint, and `1,2,3` is the definition.

The first line assigns a constraint to all statements of the unit under test. The following line just assigns the constraint `a==3` to the first three statements. An interesting variant is the last line. Since the expression is an assignment rather than a boolean compare, variable 'a' gets the new value 2 at the sixth statement. So the user is able to force the traversal of some paths or test some unintended side effects. As we will see in chapter 7, constraints can also be used for specifying an operational profile.

3C-activities:

If the desired coverage could not be reached the user has to check manually if

- the chosen coverage criterion involves the traversal of infeasible statements or paths
- the termination condition is set appropriately

The other activities associated with the identifier '3C' are up to the overall Verification & Validation (V&V) organisation. Since they do not serve as an input for GATester, they are not discussed here.

4. First Performance Measures

The author's algorithm for software test data generation was tested on a number of software units. The units' cyclomatic complexity ranges from one to 70, their lines of code vary between three and 213, and they are all written in the C programming language.

Objectively measuring the performance of a genetic algorithm is not a trivial matter. Factors to be taken into account are at least speed (how quickly the algorithm completes) and success rate (what proportion of runs converge to an optimal solution).

Test data generation in software testing is the process of identifying program test data which satisfy selected testing criteria [17]. The testing criterion chosen for the experiments discussed in this chapter is 100% statement coverage. Assuming that the goal of running GATester is to solve the test data generation problem, the algorithm completes after satisfying the given testing criterion.

The experiments clearly show that both factors speed and success rate are determined by statement feasibility and decision statement complexity. Statement feasibility may be defined as the execution probability of a statement. Assuming random input variables of the software unit under test, a statement with a high statement feasibility will be executed more often than a statement with a lower one. The lower its feasibility the more iterations are necessary in order to find an optimal solution, i.e. appropriate test data. Splitting complex decisions into more and simpler ones is another way of contributing to a better performance. The reason why GATester behaves this way is its learning algorithm. The search direction within the set of all possible solutions is driven by a function (usually called fitness function) evaluating current potential solutions. This evaluation is based on current statement coverage. Since statement coverage increases faster if more and simple instead of less and complex statements are provided, software engineers can contribute to a better performance of GATester by avoiding very complex decision statements. Figure 4-1 shows current performance values (586 CPU, 166 MHz, 32 MB RAM, Linux).

Computation time in clocks	Statement feasibility
149	1,5E-05
245	6,0E-08
9546	2,3E-10
379394	9,1E-13
---	3,6E-15

Figure 4-1: Computation time against statement feasibility

For software units having statement feasibilities of up to 9,1E-13 the success rate was 100%. For statement feasibilities of about 3,6E-15 the algorithm didn't find an optimal solution. But note that the runs were limited to 1000000 generations. Without this constraint GATester is able to generate test data for such cases as well. Current experiments focus on optimizing the evaluation function in order to speed up the search process.

A more detailed analysis of GATester's performance including some charts can be found in [34].

5. Genetic Algorithms

One of the major problems with automating software testing is its complexity. Genetic algorithms aim at such complex problems and have already been applied quite successfully to optimization problems such as scheduling, transportation problems, etc. [2][23].

This chapter is devoted to a discussion of genetic algorithms (GAs) in general. The author answers the following questions:

- What are GAs?
- How do GAs work?
- Why do GAs work?

5.1. What are GAs?

As stated in [23], there is a large class of interesting problems for which no reasonably fast algorithms have been developed. Many of these problems are optimization problems that arise frequently in applications.

Basically, GAs maintain a population of individuals. These individuals, also called chromosomes, represent potential solutions to a given problem. In order to find the best solution, they undergo an evolution process by applying rules of selection, mutation, and reproduction, similar but far less complex than known from natural genetics. GAs use fixed-length binary strings and only two basic genetic operators, that are *mutation* and *crossover*.

The evolution process run on a population of chromosomes corresponds to a search through a space of potential solutions. Such a search requires balancing two (apparently conflicting) objectives: exploiting the best solutions and exploring the search space [23]. Since GAs are a class of general purpose search methods, the strength of GAs is to balance these two objectives stated above.

Applying genetic algorithms to a problem entails finding the proper representation of the problem and a fitness function. The author's representation is a bit stream storing the actual test data. In other words, a chromosome contains the test data needed for one test case. Since a chromosome is made of genes, arranged in linear succession, one gene is a concrete variable of the software under test. The fitness function determines the relative quality of the solutions of every chromosome, i.e. their fitness. Chromosomes with a high fitness are the most likely candidates for further reproduction.

The application we discuss is a software testing tool, and the main problem we face is the generation of adequate test data. We use a GA to generate test data that are able to test the software under test according to a given test strategy.

Going through the structure of the genetic algorithm in more detail, the basic approach of the testing tool is revealed. First of all, the population containing the first set of test cases must be initialized appropriately. Then, little by little all test cases of the first generation are executed during the execution phase. Although being unlikely after the first generation, the testing tool checks if the given

termination condition is already true. If so, the testing process will stop successfully. Otherwise, the next generation must be initialized. This task is the most critical one. The new generation is basically a new version of the last one. More details about this selection process are presented in chapter 6. Afterwards some changes are introduced that make the test cases doing a better job than the prior ones. These changes are discussed in detail in the following chapter. After executing the test cases of the actual generation again an evaluation step decides whether or not the termination condition is true. After some loops (up to several millions) the test process will stop according to the user defined termination condition. The number of iterations depends on the termination condition being specified prior to the tests. This may be a requirement like “stop if 90% statement coverage reached” or “in any case, do not generate more than 100000 generations”.

The next paragraphs illustrate the GAs' effectiveness by analyzing how and why they really work.

5.2. How do GAs work?

First of all the initial population must be defined. This may be done randomly in a bitwise fashion, since the chosen representation is a bit stream. If there is already some knowledge about potential solutions or optima available, the initialization process may be changed appropriately.

A large number of strategies exists for determining the contents of a new generation. Mostly they only differ in some details. Basically, for each generation the fitness function calculates the fitness of each chromosome. The detailed algorithm is presented in chapter 6.

After the selection process the recombination operator, crossover, is applied to the chromosomes. Crossover combines the features of two parent chromosomes to form two offsprings by swapping corresponding chunks of the parents. The position of the crossing point determining the size of the chunks is assigned randomly.

There also exists a certain amount of mutation, where one bit of a chromosome at random is replaced by a random value. More precisely speaking, a bit changes from zero to one or vice versa, since we have a binary representation of the data. Mutation introduces some extra variability by randomly changing a single position of a selected chromosome and is performed on a bit by bit basis.

After selection, crossover, and mutation, again an evaluation procedure follows, determining the fitness values of the chromosomes. The cyclic repetition will stop after a defined termination condition turns out to be true. As already mentioned above, the testing tool's termination condition can be configured in two ways. There may be specified a defined maximum of generations, or a given problem related termination condition like 100% statement coverage, or a combination of both.

5.3. Why do GAs work?

GAs provide robust and powerful adaptive search mechanisms. More precisely, they maintain a population of chromosomes that evolve according to the rules presented above in order to find a solution to a given problem.

The structure of the information stored in a chromosome does not change from one problem to the other. A chromosome always consists of several genes storing information in a binary string.

But the representation of a problem in terms of parameters may be unique for each problem to be solved by GAs. A potential solution to a problem is usually represented as a set of parameters, known as genes. This transformation step from the problem space into a set of parameters or genes is the most critical task for the GA's performance. Results will be most successful if the coding strategy applied in the transformation step forces the creation of "tight" building blocks that will not be destroyed by the genetic operators, i.e. crossover and mutation. In other words, the genes should be as small as possible.

Here we have to deal with a new term called "schema". A schema is built by introducing a don't care symbol (x) into the alphabet of genes. Now we can create strings or schemata over the ternary alphabet {0,1,x}. For example, the string 101x matches two strings, namely 1010 and 1011. Whereas both the zero and the one retain their normal meaning, the don't care symbol can be interpreted as either a zero or a one. Notice that the don't care symbol is just a meta-symbol that is not explicitly processed by the genetic algorithm. There are two important schema properties, *order* and *defining length*.

The order of a schema S is the number of 0 and 1 positions, i.e. fixed positions (non-don't care positions), present in the schema. In other words, it is the length of the template minus the number of don't care symbols. The order defines the speciality of a schema [23].

For instance, the order of the schema $S=(x0x111x0x)$ is five.

The defining length of a schema S is the distance between the first and the last fixed string positions. It defines the compactness of information contained in a schema. [23]

For example, the defining length of the schema $S=(x0x111x0x)$ is $8 - 2$ or six because the last fixed position is the 8th, and the first is the 2nd.

A schema is a similarity template describing a subset of potential solutions with similarities at certain positions. Schemata greatly simplify the analysis of the performance of GAs. The key point here is that short, low order schemata having an above average fitness receive exponentially increasing trials in subsequent generations of a GA. This statement, known as the **Schema Theorem** gives an immediate result, that is GAs explore the search space by short, low-order schemata which, subsequently, are used for information exchange during crossover [23]. Such "tight" building blocks having high fitness will most likely stay alive and join the next generation. In other words, the best get more copies in the next generation, the even stay even, and the worst die off.

The combined effect of selection, crossover and mutation on a particular schema of course increases the probability that this schema will be disrupted. But it still receives an exponentially increasing number of strings in the next generations.

6. The Evaluation Procedure

Genetic algorithms do not search only one path through the search space. On the other hand, they do not conduct an exhaustive search of the space of all possible solutions. Rather, they perform a type of beam search where the population, i.e. the set of current potential solutions, is the beam [2]. An important task is to decide which members of the population will be subject to the genetic operators presented in chapter 5.

The evaluation procedure decides if the test cases performed were “good ones”. Technically speaking, their fitness is evaluated. Good test cases are test cases that meet the testing criterion, for example statement coverage. If so, the termination condition is fulfilled and the test will stop. Otherwise, the test data will be refined and the next test case executed. Recall that one test case is represented by one chromosome. Basically, the following steps are performed by the evaluation procedure [23]:

First part:

- Calculate the fitness value for each chromosome
- Find the total fitness of the population
- Calculate the probability of a selection for each chromosome
- Calculate a cumulative probability for each chromosome

Second part (loop x times, x is the population size in terms of chromosomes):

- Generate a random number from the range [0..1]
- Select the first chromosome whose cumulative probability is greater than the random number

The evaluation procedure selects a new population with respect to the probability distribution based on fitness values of chromosomes of the current population. It also ensures that the best chromosomes get more copies, the average stay even, and the worst die off. But note that the next generation consists of the same number of individuals as the former one.

The chromosomes forming the new generation then are subjected to the genetic operators crossover and mutation, respectively. Afterwards the test cases are executed one by one. Their contribution to the overall testing goal, for example statement coverage, determines their fitness value. And again a new iteration of the evaluation procedure has started.

7. Software Reliability Considerations

Reliability is the probability that a product or system will perform some specified end user functions under specified operating conditions for a stated period of time.

Making good reliability estimations or predictions depends on testing the product as if it were in the field [24]. The operational profile is a set of end user functions the product can perform with their probabilities of occurrence. It is clear that estimating the operational profile is a non-trivial task. However, it is essential in reliability engineering. Once a satisfactory operational profile is available, the testing can begin, and reliability growth can be monitored.

To apply a reliability measurement approach to a system, it is important to understand that both hardware components and software units have to be considered. Only if both software and hardware reliability calculations are combined reliability can serve as a high-level indicator of the operational readiness of a system. Note that there is a fundamental difference in the dynamic behaviour of hardware and software. Whereas initially failure-free hardware components may show some defects later on because of wearing out, every software unit already contains all of its faults at the time the software engineer performs the last file save operation. In other words, all errors concerning software engineering are made in time, and the testing can start. Clearly, neither the customer nor the organization responsible for a product wishes to have defective products. Nevertheless we can be sure that a system will fail some time. Even if defect minimization strategies would have been able to establish a failure free development process, the wear out phase introduces defects. But the complexity of software systems combined with budget and schedule constraints makes it practically impossible to ship zero-defect-software.

Today we are confronted with a plethora of models, techniques, and measures for software reliability engineering in the literature. Nevertheless, it is still a matter of fact that the user must decide which model is the most appropriate for a given application. This decision is by no means an easy one because there does not exist one single model that is able to produce reliable results in all contexts. Further more, the outcomes of the models may also vary considerably. But probably the main problem is that it does not seem possible to analyze the particular context in order to decide a priori which model is likely to be trustworthy [1]. But the author also believes that you can obtain reasonably accurate reliability measures for relatively modest reliability levels. As stated in [5], techniques that depend on reliability growth cannot assure very high reliability without infeasibly large observation periods.

This section discusses only issues concerning software reliability. More precisely, the author presents his approach of automatically producing a forecast for acceptance testing results by means of a unit reliability measurement chart and a unit run reliability calculation. The testing tool GATester calculates all the primitives needed for the unit's reliability measurement chart or for its run reliability calculation without manual assistance. The results may serve as a helping hand for managerial decisions or as a parameter for software system reliability assessment. Note that this kind of measurement information supports

both technical and managerial activities. Note also that the tool calculates reliability figures for the particular unit under test. Therefore, the results do not represent software system reliability but just single software unit run reliabilities. Actual research activities include the issue of combining all the unit's results to one big number called software system reliability.

The unit reliability measurement chart introduced above gives an idea of the maturity of the software unit. For example, it can be used to determine software readiness for acceptance testing. It plots the failure time against the failure number and identifies five regions within the chart, telling you if the testing shall proceed or not. Two areas, namely reject-area (R-area) and accept-area (A-area), recommend to stop the testing because the test results are not acceptable or the desired maturity has already been reached, respectively. The three remaining regions recommend to continue testing. These are called probable-reject-area (PR-area), probable-accept-area (PA-area), and continue-area (C-area). If the required reliability rating is low the project management staff may decide to perform an acceptance test although some of the software units have not reached the A-area so far, but are still in the PA-area. On the other hand, if a unit is in the PR-area and the required reliability rating is very high, it is recommended to step back one or more life-cycle-phases and try to refine the unit until the unit reliability measurement chart shows acceptable results.

The run reliability R_k is the probability that k randomly selected runs (corresponding to a specified period of time) will produce correct results [39]. The author adapted this measure and created a so-called unit run reliability. Assuming a uniform probability distribution, $R_k = nc/NR$, where nc is the number of correct runs in a given test sample and NR is the number of total runs made in a given test sample. The unit's input space is viewed as the set of all possible combinations of inputs into the unit, for example global variables, parameter, or user inputs. Generally speaking, you cannot test all theoretically possible combinations of input variables because it would simply last too long, sometimes up to several thousands of years. That's why the problem of minimizing the number of test cases has to be addressed by a test tool. During runtime, the genetic algorithm generates more and more test cases according to the evolution strategy. GATester generates and executes a subset of all theoretically possible test cases. These test cases form the sample space that is relevant for the unit run reliability that is calculated at the end of an automatic unit test.

If the user provides GATester with some constraints as shown in chapter 3, the corresponding unit run reliability is denoted by R_{kCx} where Cx is the x^{th} constraint provided by the user. As shown below, constraints are a necessity for reliability calculations. This is a powerful enhancement of run reliability calculation because it can be used to assess the reliability of an arbitrary subset of the unit's functionality.

It appears clear that the operational profile cannot be calculated automatically by GATester. How should the tool know about the end-user's habits? Once again, the constraints specification is used in order to solve this problem. As already outlined in chapter 3, a constraint is basically an expression of whatever complexity the user chooses. A set of constraints can also represent an operational profile by controlling the input variables appropriately. Without specifying any constraints, the user still gets a rough idea of the unit's maturity,

but only in terms of coverage testing results. Constraints allow GATester to take the dynamic behaviour into consideration. In other words, the testing tool itself is able to decide if a failure occurred or not. Without this information GATester's calculated reliability figures would be meaningless. Reliability assessment without any kind of reasonable failure data will not work per definition.

Every testing technique is limited in its ability to detect failures. Starting with technique A you usually will detect some failures. But after a while the detection rate will decrease significantly. One might think that at this point almost all failures are detected, therefore it is difficult to find any more. But more likely the so-called saturation point of testing technique A has been reached. In other words, there are still some failures waiting to be detected, but they cannot be found by technique A. If you start using technique B afterwards, the scenario might be very much the same. First, your detection rate is quite good, although technique A was unable to find failures, but then it gets worse and worse, similar to A because again the saturation point of B has been reached. In [21] the potential danger of reliability overestimation is pointed out. Assuming that the software is more reliable if failures are identified and fixed, it may be concluded that it must be sufficiently reliable if there cannot be detected any more failures. The danger now is that the reliability estimation is based on the saturation point, rather than on the real failure data revealing when several testing techniques are being used. That is one of the reasons why the reliability figures calculated by GATester may serve as a parameter for system reliability assessment but in fact cannot be the system reliability assessment. By the way, the operational profile is the second reason. Further research is necessary in order to clarify how to transform a given operational profile at system level into one at unit level without loss of important information.

8. Conclusions and Future Work

In this paper the author has presented an approach for software test automation including some key issues in software reliability assessment. We have seen how to use genetic algorithms for software testing, particularly for test data generation and studied its performance and effectiveness.

The tool presented in this paper, GATester, supports the effort of reducing the costs of software testing. All the test cases can be generated and executed without user interaction at a rate of up to several hundreds per minute. In the end, the software unit under test has passed sufficient test cases for a given coverage requirement if it is feasible at all. The number of test cases is a significant parameter for the overall testing effort. By automatically generating test data you are able to apply a quantitative approach to software testing detecting also failures that would not be found otherwise by applying conventional testing techniques, for example EP (equivalence partitioning) or BVA (boundary value analysis). If the user provides GATester optionally with a so-called constraint specification the tool is able to detect failures during test execution and calculate some reliability figures.

At the moment, GATester accepts only units written in the C programming language. It is intended to support also C++ and maybe some other languages as well.

The tester should also be able to choose data flow criteria as a coverage criterion. If the result of some computation has never been used, one has no reason to believe that the correct computation has been performed [30].

But in the long run, the most challenging research project is the combination of the approach presented in this paper with another approach that tries to generate test cases for system testing automatically from a formal specification. Then GATester could derive the constraints specification automatically and would also be able to include self checking procedures for the generated test cases.

9. References

- 1 Abdel-Ghaly A. A., Chan P. Y., Littlewood B., "Evaluation of Competing Software Reliability Predictions", IEEE Trans Software Engineering, vol. SE-12, no. 9, September 1986, pp. 950-967
- 2 Banzhaf W., Nordin P., Keller R. E., Francone F. D., "Genetic Programming", Morgan Kaufmann Publishers, Inc. and dpunkt-Verlag für digitale Technologie GmbH, 1998
- 3 Beizer B., Software Testing Techniques, 2nd ed., Chapman & Hall, 1990
- 4 Bird D. L., Munoz C. U., "Automatic generation of random self-checking test cases", IBM Systems Journal, vol 22, no. 3, 1983, pp. 229-245
- 5 Brocklehurst S., Chan P. Y., Littlewood B., Snell J., "Recalibrating Software Reliability Models", IEEE Trans Software Engineering, vol 16, no 4, April 1990, pp. 458-470
- 6 Brocklehurst S., Littlewood B., "New Ways to Get Accurate Reliability Measures", IEEE Software: Special issue on Software Reliability Modelling, July 1992, pp. 34-42
- 7 Davey S., Huxford D., Liddiard J., Powley M., Smith A., "Metrics Collection in Code and Unit Test as Part of Continuous Quality Improvement", Software Testing, Verification and Reliability, vol. 3, 1993, 125-148 (1993)
- 8 DeMillo R. A., Offutt A. J., "Constraint-Based Automatic Test Data Generation", IEEE Trans Software Engineering, vol. 17, no. 9, September 1991, pp. 900-910
- 9 Ehrlich W. K., Lee S. K., Molisani R. H., "Applying Reliability Measurement: A Case Study", IEEE Software, pp. 56-64, March 1990
- 10 Ehrlich W., Prasanna B., Stampfel J., Wu J., "Determining the Cost of a Stop-Test Decision", IEEE Software, March 1993, pp. 33-42
- 11 Everett W. W., "Reliability and Safety of Real-Time Systems", IEEE Software, Guest Editors' Instruction, May 1995, pp. 13-16
- 12 Fewster M., "The Managing Director Wants 100% Automated Testing. A Case History", Journal of Software Testing, Verification and Reliability, vol 1, no 2, 1991, pp. 43-55
- 13 Graham D., "Software Testing Tools: A New Classification Scheme", Journal of Software Testing, Verification and Reliability, vol. 1, no. 2, ???, pp. 17-34
- 14 Hamlet D., "Are We Testing for True Reliability?", IEEE Software, July 1992, pp. 21-27
- 15 Hamlet D., "Partition Testing Does Not Inspire Confidence", IEEE Trans Software Engineering, vol. 16, no. 12, December 1990, pp. 1402-1411
- 16 Harrold M. J., Soffa M. L., "Selecting and Using Data for Integration Testing", IEEE Software, March 1991, pp. 58-65
- 17 Korel B., "Automated Software Test Data Generation", IEEE Trans Software Engineering, vol. 16, no. 8, August 1990, pp. 870-879
- 18 Korel B., "Dynamic Method for Software Test Data Generation", Software Testing, Verification and Reliability, vol. 2, 203-213 (1992)
- 19 Littlewood B., "Software reliability modelling", Software engineer's reference book, chapter 31, Butterworth-Heinemann, 1991
- 20 Littlewood B., "Stochastic Reliability-Growth: A Model for Fault-Removal in Computer-Programs and Hardware-Designs", IEEE Trans Reliability, Vol. R-30, no. 4, October 1981, pp. 313-320
- 21 Lyu M. R., Handbook of Software Reliability Engineering, McGraw-Hill/IEEE Computer Society Press, 1996
- 22 Lyu M. R., Nikora A., "Applying Reliability Models More Effectively", IEEE Software, July 1992, pp. 43-52
- 23 Michalewicz Z., Genetic Algorithms + Data Structures = Evolution Programs, 2nd ed., Springer-Verlag, 1994

- 24 Musa J. D., "Operational Profiles in Software-Reliability Engineering", IEEE Software, March 1993, pp. 14-32
- 25 Musa J. D., "Software-Reliability-Engineered Testing", Computer, November 1996, pp. 61-68
- 26 Musa J. D., Ackermann A. F., "Quantifying Software Validation: When to Stop Testing?", IEEE Software, May 1989, pp. 19-27
- 27 Musa J. D., Everett W. W., "Software-Reliability Engineering: Technology for the 1990s", IEEE Software, Nov 1990, pp.36-43
- 28 Paradker A., Tai K. C., Vouk M. A., "Automatic Test-Generation for Predicates", IEEE Trans Reliability, vol. 45, no. 4, December 1996, pp. 515-530
- 29 Ramamoorthy C. V., "On the Automated Generation of Program Test Data", IEEE Trans Software Engineering, vol. SE-2, no. 4, December 1976, pp. 293-300
- 30 Rapps S., Weyuker E. J., "Selecting Software Test Data Using Data Flow Information", IEEE Trans Software Engineering, vol. SE-11, no. 4, April 1985, pp. 367-375
- 31 Roper M., "Automatic test data generation", in Software TESTING '96, June 1996, Paris
- 32 Veevers A., "Some Issues in Software Reliability Assessment", Journal of Software Testing, Verification and Reliability, vol. 1, no. 1, 1991, pp. 17-22
- 33 Voas J., Morell L., Miller K., "Predicting Where Faults Can Hide from Testing", IEEE Software, March 1991, pp. 41-47
- 34 Weichselbaum R., "Software Test Automation - By Means of Genetic Algorithms", EuroSTAR98 Proceedings, 6th European International Conference, Munich, 1998
- 35 Weyuker E., Goradia T., Singh A., "Automatically Generating Test Data from a Boolean Specification", IEEE Trans Software Engineering, vol. 20, no. 5, May 1994, pp. 353-363
- 36 Wood A., "Predicting Software Reliability", Computer, November 1996, pp. 69-77
- 37 ANSI/IEEE Std 1008-1987, IEEE Standard for Software Unit Testing
- 38 IEEE Std 1012-1998, IEEE Standard for Software Verification and Validation
- 39 IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software



Automated Test Generation From a Behavioral Model

Jim Clarke

Member of Technical Staff

Lucent Technologies

jmclarke@lucent.com



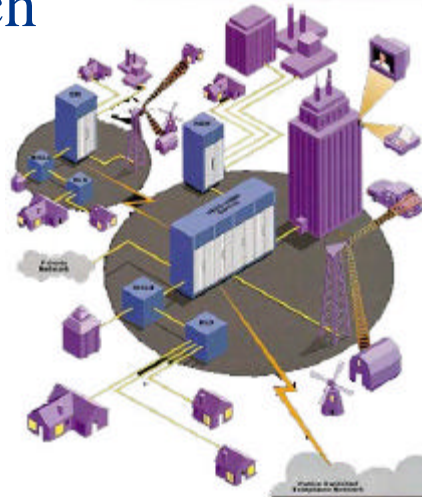
Outline

- 5ESS[®]-2000 Switch
- Challenges for Testers
- Requirements Behavioral Modeling
- Automated Test Generation
- Case Studies
- Conclusions



5ESS[®]-2000 Switch

- Digital Exchange
- Single System, Multiple Applications
 - ISDN voice & data
 - Local & Long Distance calls
 - Intelligent Network
- Distributed Architecture



Challenges For Testers

- Growing complexity of feature specifications
- Business needs require reduced development intervals
- Find new test design approach



Specific Problems Addressed

- Problem 1:
Effectively testing software with complex requirements
- Problem 2:
Efficiently testing software with complex requirements



Behavioral Modeling

- Control Flow Testing - a technique based on a *structural* model
- Transaction Flow Testing - a technique based on a *functional* model
- Extended Finite State Testing - control and transaction flow in a *Mealy* model

Behavioral Modeling Advantages

Lucent Technologies
Bell Labs Innovations



- Quickly identifies redundant and invalid requirements
- Quickly identifies discrepancies between requirements and design
- Quickly determines the impact of new and modified requirements
- Improves requirements coverage

7

Quality Week Europe '98

© 1998 Lucent Technologies

Behavioral Modeling Caveats

Lucent Technologies
Bell Labs Innovations



- Does not explicitly identify missing requirements
- It's unlikely to find problems if modeled from the code itself
- Flowgraphs can contain hundreds of states or nodes
- Tests are only as good as the model

8

Quality Week Europe '98

© 1998 Lucent Technologies

Behavioral Modeling Difficulties

Lucent Technologies
Bell Labs Innovations



- Manual model validation required
- Manual mapping of scenarios and requirements to tests
- Manual update of tests and tables to reflect changes in requirements
- Models quickly becomes cumbersome
- Effective technique is inefficient

9

Quality Week Europe '98

© 1998 Lucent Technologies

In Search of Automation.....

Lucent Technologies
Bell Labs Innovations



- Graphical User Interface (GUI)
- Independent of execution environment
- Supports behavioral modeling
- Automatic generation of test cases
- Supports constraints to limit test output

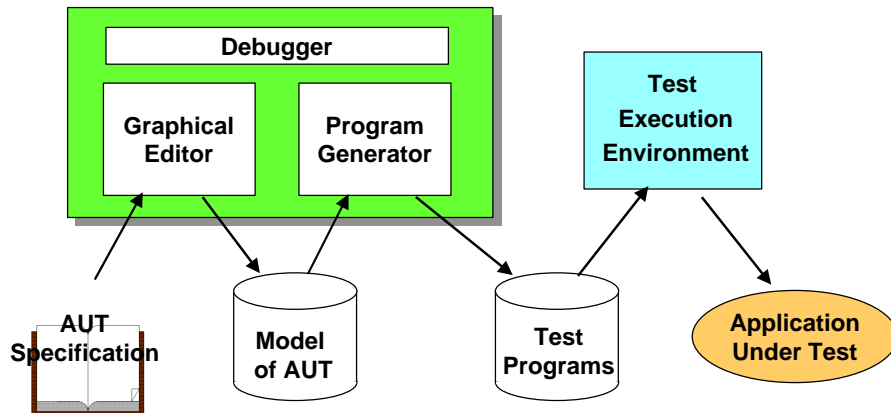
10

Quality Week Europe '98

© 1998 Lucent Technologies

TestMaster Subsystems

Lucent Technologies
Bell Labs Innovations



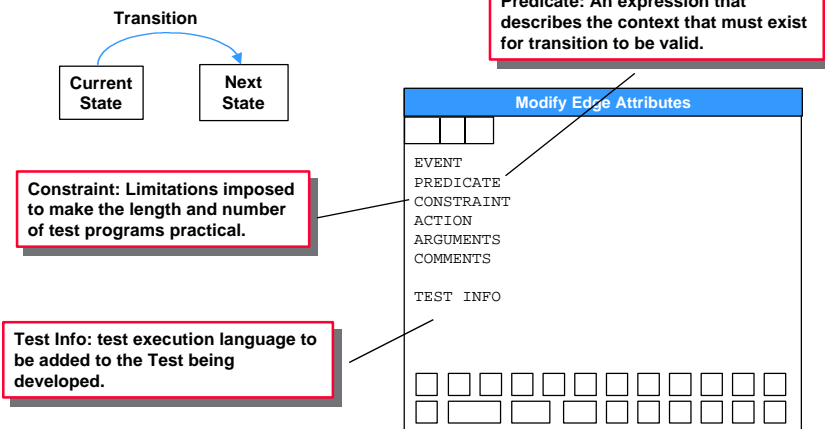
11

Quality Week Europe '98

© 1998 Lucent Technologies

Model Reference Technology

Lucent Technologies
Bell Labs Innovations



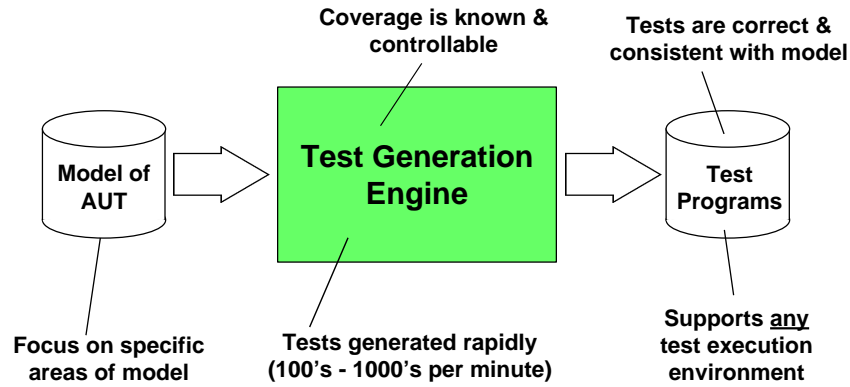
12

Quality Week Europe '98

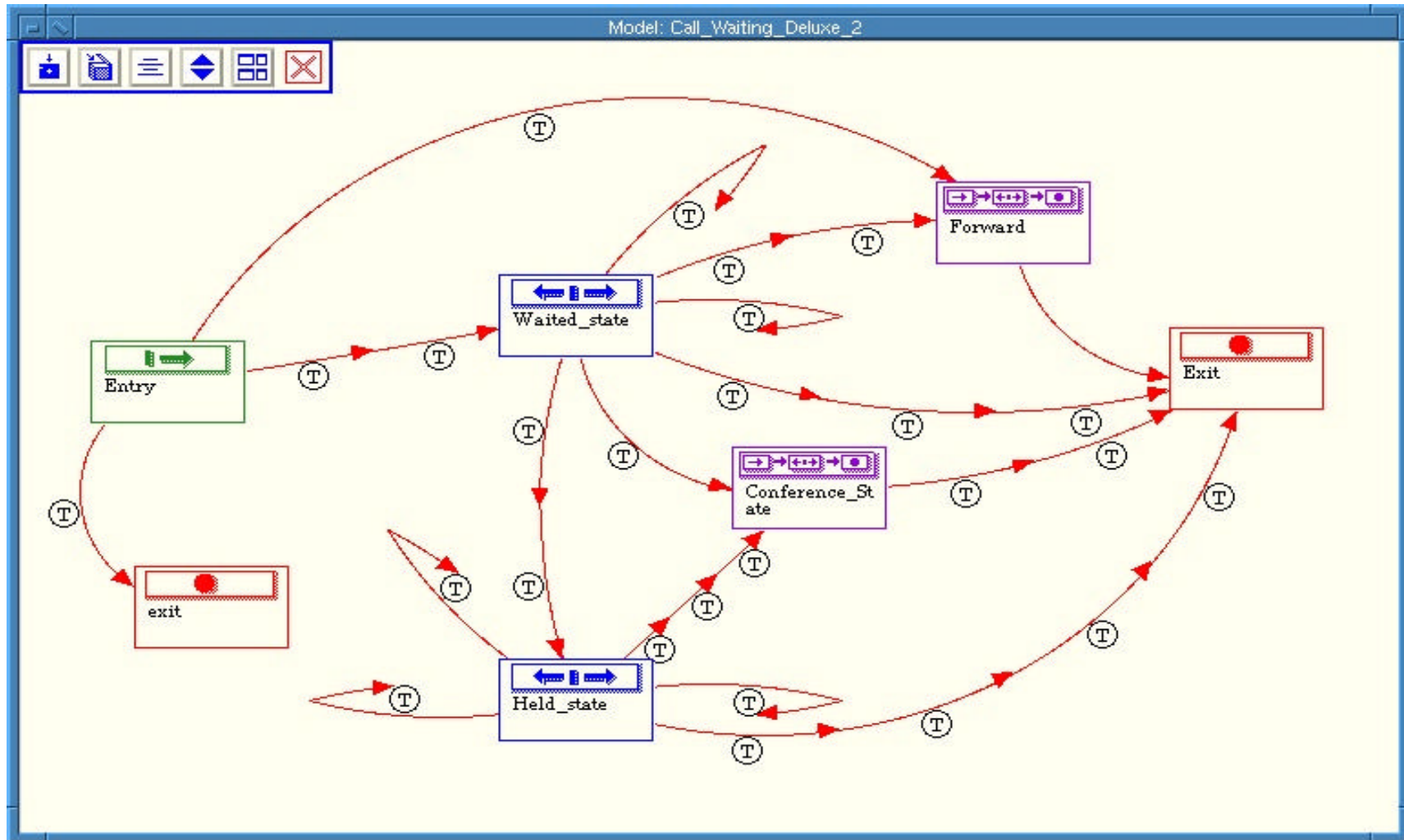
© 1998 Lucent Technologies

Model Reference Technology

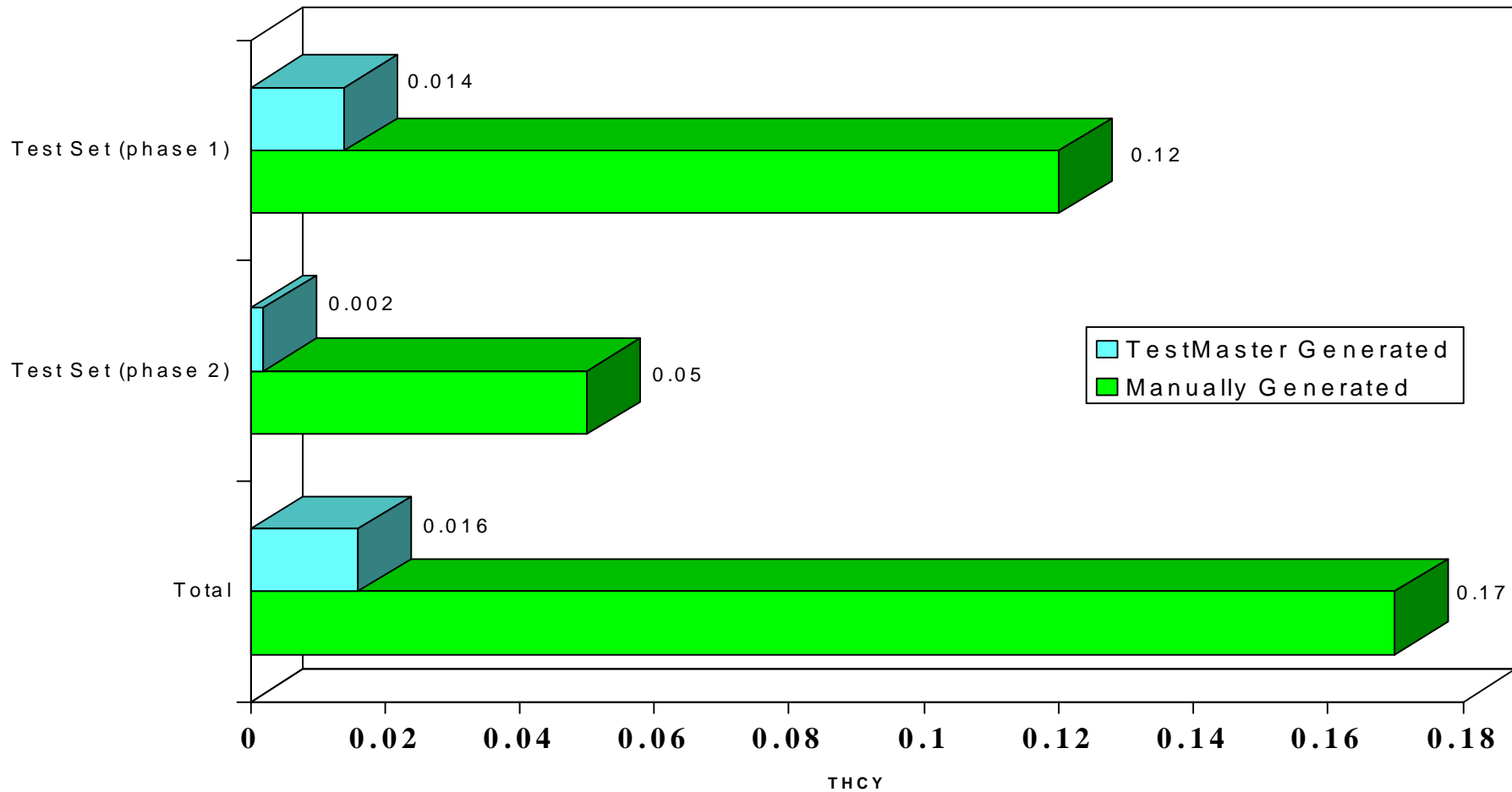
Lucent Technologies
Bell Labs Innovations



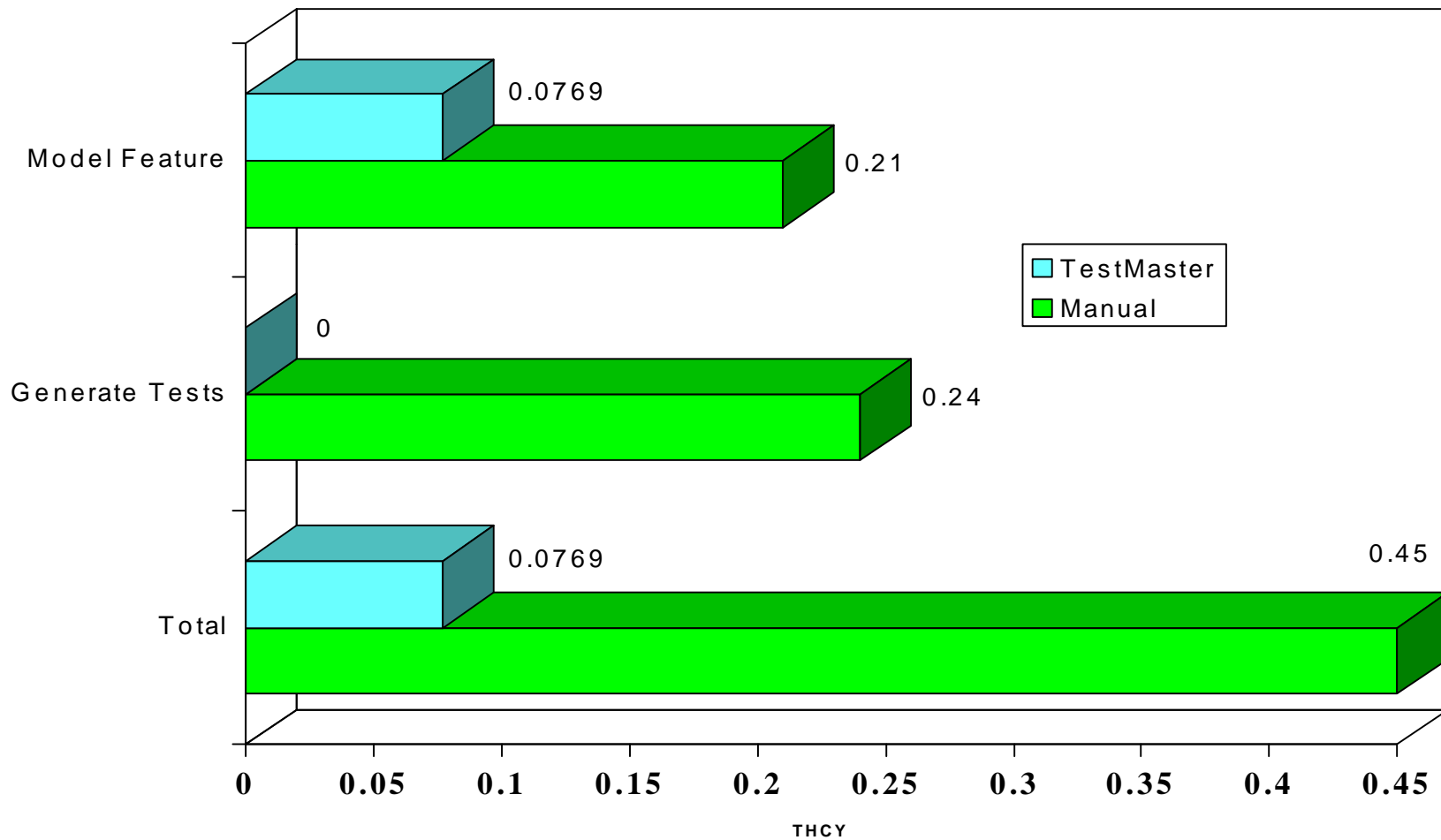
TestMaster Model Example



Case Study 1



Case Study 2



Manual vs. TestMaster

Lucent Technologies
Bell Labs Innovations



TEST GENERATION METHOD	MANUAL	TESTMASTER
Inputs	Knowledge of Application Knowledge of Test Lab Schedule Resources	Knowledge of Application Knowledge of Test Lab Schedule Resources
Outputs	Static Test Strategy Static Set of Tests	Model of Application's Behavior Dynamic Test Set
Comparison	Higher Cost per Test Higher Test Maintenance Cost No Dependency on Technology	Lower Cost per Test Lower Test Maintenance Cost Improved Coverage Improved Fault Prevention

17

Quality Week Europe '98

© 1998 Lucent Technologies

What's Next ??

Lucent Technologies
Bell Labs Innovations



- Formalize modeling standards
- Determine how to formally review models
- Integrate with automatic execution
- Develop new coverage metrics

18

Quality Week Europe '98

© 1998 Lucent Technologies

Automated Test Generation from a Behavioral Model

James M. Clarke

Lucent Technologies
2000 Naperville Road
Naperville, IL 60666-7033
(630) 979-1861
jmclarke@lucent.com

Abstract

The challenge for testers: reduce the testing interval without reducing quality. One answer: find a new way to approach test design and test generation. This paper will discuss an ongoing Lucent Technologies experiment in automated test generation from a behavioral model of the software product under test. Results indicate that our new approach can increase the effectiveness of our testing while reducing the cost of test design and generation.

Outline

1. Introduction
2. 5ESS[®]-2000 Testing Background
3. Major Challenges for Testers
4. New Test Design Strategy - Behavioral Modeling
5. Automatic Test Generation
6. Case Studies
 - Case 1: Call Management Feature
 - Case 2: Number Portability Feature
7. Observations and Conclusions

1. Introduction

At Lucent Technologies, TestMaster™ automates the generation of tests for call processing features developed for the 5ESS[®]-2000 Switch. The 5ESS-2000 Switch, a digital exchange for use in the global switching network, allows service providers, such as telephone companies, to route ISDN voice and data, local voice calls, long distance calls, Internet access, wireless PCS, Advanced Intelligent Network services, interactive video and multimedia services in a high-speed, reliable public network. During the test development phase, a call processing feature's specification document (FSD) serves as the basis for a TestMaster state-based model. The model describes the behavior of the switch/network when a call uses the associated feature. TestMaster then performs a path analysis on the model, generating a comprehensive set of tests that are formatted and executed in the Lucent 5ESS-2000 testing environment.

In this paper I will review the testing problems we faced, the solutions we found and the results of implementing those solutions.

2. 5ESS[®] - 2000 Testing Background

The 5ESS-2000 Switch is a flexible digital exchange for use in the global switching network. Digital switches replaced earlier electromechanical and analog switching systems. A digital switch is a single system with multiple applications such as local, toll, and operator services. The 5ESS equipment switches ISDN voice and data, local voice calls, long distance calls, Advanced Intelligent Network services as well as other media on the public switched network. The switch architecture is a modular, distributed architecture that allows developers to implement enhancements easily and allows service providers to change their communication network quickly.

The modular design of the 5ESS-2000 Switch also carries through to its software architecture. The software, primarily written in the C programming language, extends the many advantages of a distributed processing environment. Lucent Technologies Bell Laboratories develops and tests the software for the 5ESS-2000 Switches that FCC-required quality monitoring has shown to be four times more reliable than its nearest competitor.

At one time Lucent Technologies (at the time a business unit of AT&T) viewed testing as a standalone phase in the traditional waterfall process. System testing was done by a separate organization, and the testers became involved in a project only after the specifications, design, and the majority of the coding was complete. This made for expensive and time consuming test plans. In fact, at one time it required almost 22 months to deliver a major software release for the 5ESS-2000 Switch. Process and organizational changes have reduced that figure to approximately 10 months, but as new features become more complex, it has become increasingly difficult to maintain both an aggressive delivery schedule and the high level of software quality that our customers have come to expect.

3. Major Challenges for Testers

The challenge now for test plan designers is to continue to achieve the high degree of testing coverage required to ensure that these increasingly complex features maintain quality standards. This requires the use of test development methods that are more effective in managing the coverage of complex functionality. Traditional methods, such as analyzing each requirement and developing test cases to verify correct implementation, are not effective in understanding the software's overall complex behavior. Also the cost pressures in a competitive industry add the constant of cost reduction. This adds the need for efficiency in using more effective test development methods. While initially these two goals, reduced testing costs and maintaining product quality, appeared to be mutually exclusive our automation test generation initiatives have indicated that this is not necessarily the case.

A Feature Specification Document, written by the systems engineering organization, details the requirements for the behavior of call processing features of the 5ESS. The behavior of the feature depends on inputs from the parties on the call and the configuration and signaling input from the 5ESS network. Complex interactions arise between the calling parties, other features on the switch, and the network, and these must be understood to adequately test the new feature. To date, test generation has relied on manual methods to interpret the Feature Specification Document, state diagrams, and call processing behavior of the switch. For a given call, the switch waits for input, e.g., a set of DTMF tones. The switch processes the input and changes the state of the call in progress. (Different inputs from the caller and network configurations cause the 5ESS switch to process calls differently.) For example, if the user enters a valid telephone number, the call will be processed; if not, an announcement will play asking for a valid input. Advanced features in the 5ESS switch have so many variables that it is difficult for the test engineer to

identify them all, let alone generate a set of tests to verify that the feature works in all cases.

4. New Test Design Strategy – Behavioral Modeling

The traditional test design methods used to generate test cases became too expensive and labor intensive when applied to these highly complex features. We had to employ a different strategy to adequately test new and existing functionality, while keeping the testing interval from growing with the software complexity.

For the last year our strategy has been to use requirements behavioral modeling on a number of features to determine the effectiveness of this approach as a test design and generation strategy. The behavioral modeling we use combines transaction-flow, control-flow and finite state machine (FSM) testing techniques in an extended finite state (EFSM) model. EFSM modeling employs a technique known as *predicate notation* to simplify models of complex systems, and reduce the state explosion problem commonly encountered with pure FSM modeling.

The goal was to create a EFSM model that would capture the functional behavior of the requirements for a new 5ESS-2000 software feature. The models, if created during the requirements definition phase of the development cycle, would prevent different interpretations of the requirements by the developers and testers. Minimizing these differences will in itself prevent some faults from ever reaching the test execution phase, helping to further reduce testing costs.

Our initial results indicate that while behavioral modeling is very effective in ensuring adequate coverage during the test design phase and in providing the entire development team with a common view of the requirements, it quickly becomes labor intensive during the test generation phase. On larger features the process of manually modeling also quickly becomes too difficult and expensive. An obvious answer was to find a tool that could automate some or all of this process. Which automation tool to use was not as obvious.

5. Automating Test Generation Using Model Reference Technology

To help decide which tool to use, we developed a checklist of the characteristics and functionality to rate automation tools- characteristics such as execution environment independence, support for EFSM, flexible output format and the ability to automatically generate unique paths (tests) from the behavioral model.

The tool with the best score based on our checklist is a product called TestMaster, and Lucent Technologies started a trial program with to evaluate its ability to allow test engineers to create and maintain behavioral models of our products.

TestMaster (produced by Teradyne, Inc.) uses model reference technology (MRT) to provide automatic test generation driven from an EFSM model of the application under test. TestMaster comprises three major components: a graphical editing tool, a test program generator, and a model debugger.

Using the same inputs used to manually generate test scripts or manually create an EFSM, test engineers use the State Transition Editor to build a model of the applications behavior. The model is a series of states connected by transitions. Each transition defines a state change based on inputs from user or switch. Each transition in the model contains the following associated programmable fields: the predicate and constraint fields, which evaluate context in the model, and the test information field that contains procedures or test code that will be included in any test case that includes the transition as part of its path. Predicates are boolean expressions that must evaluate *true* in order for the transition to be a valid path within the behavioral model. The constraint field allows the user to limit the number

of paths produced during test generation. A set of interactive debugging tools is available to the test engineer as well.

The test program generator uses the model to automatically find valid paths through the model. These paths consist of transitions that represent the behavior of the application that has been modeled. Each valid path through the model is converted into a test case by replacing each transition in the path with its test information. Thus a complete test case is concatenation of all the test information field for some valid path. These test cases can be produced in any target language.

6. Case Studies

This section will briefly discuss two cases in which we can compare generating tests with TestMaster to manually writing tests. In both cases the two methods were used to create comparable type and number of test cases. Both of these cases are products that are currently available on the 5ESS-2000 Switch.

Case 1: Call Management Feature

Background

This feature expands the capabilities of basic Call Waiting to include a number of call management features. If you subscribe to Call Waiting on your analog phone line, and a third party calls you while you are on a phone call, you receive tones indicating that another call has arrived. At this point you only have two choices: press the phone switch-hook and answer the new call or ignore the new call.

Call Management provides the ability to see the new call's telephone number¹ and the name of the caller². At this point you can conference the two calls together, place either call on hold (music optional), or forward the new call manually or automatically to another telephone number.

Test Generation: Manual vs. TestMaster

This product was delivered in two phases. Phase two testing required, modifying some of phase one's tests, using some of phase one's tests as is, and writing new tests. Test generation is measured by the Technical Head Count Year (THCY) effort required to produce the test cases required. For example, if the test generation took an engineer one month to complete, it would equal a 0.0833 THCY effort.

To generate the tests manually, we used traditional 5ESS-2000 call processing test design and generation methods. Then, using TestMaster, we created an EFSM for the product and automatically generated test cases. Table 1 compares the THCY effort required by these two methods for this feature.

	Manual Generation	TestMaster Generation
Phase One	0.120	0.014
Phase Two	0.050	0.002
Total	0.170	0.016

Table 1

The use of TestMaster in this case provided a test generation productivity improvement of just over 90%. At this level of test generation productivity

¹ Caller ID Feature

² Calling Name Feature

improvement one test engineer using TestMaster can be as productive as ten test engineers using manual test generation.

Case 2: Number Portability Feature

Background

The competition to provide local phone service is increasing every year. But most people would probably decline to change their local service providers if changing companies meant changing phone numbers. The Number Portability (NP) feature, mandated by the FCC to overcome this barrier, allows you to switch service providers without changing your telephone number.

Test Generation: Manual vs. TestMaster

For this feature we manually created an EFSM behavioral model of the requirements and manually generated test cases using the model. Then, we created an EFSM of the product in TestMaster and automatically generated test cases. Table 2 compares the THCY effort required by these two methods for this feature.

Test generation is again measured by the Technical Head Count Year (THCY) effort required to produce the test cases required.

	Manual	TestMaster
Create Model	0.21	0.05
Generate Tests	0.24	0.00 ³
Total	0.45	0.05

Table 2

In this case TestMaster provided a test generation productivity improvement of just over 88%. Additional functionality was added to this feature after the original feature was released. Editing the TestMaster model to create the new tests case took half a day⁴ compared to the estimate of two and a half weeks⁵ for manual generation.

7. Observations and Conclusions

TestMaster provides a single environment to capture the behavior, input variables, configuration, and 5ESS state information in the form of a model. TestMaster can then automatically process the model to quickly generate a complete set of tests. This technology provides the 5ESS-2000 call processing test team an efficient and effective method of generating feature tests for 5ESS development projects.

Since starting with TestMaster in September of 1996, we have successfully modeled, generated, and executed test cases for a number of advanced call processing features in the 5ESS switch. To increase the reusability of the models, test engineers are developing standardized methods for analyzing the Feature Specification Document and creating TestMaster models. We are also investigating ways to formally review the models for completeness. The test cases generated are in a standard format so they can be used by both manual and automated test executors who have no knowledge of the TestMaster model and who run the tests as if they were generated manually. We can

³ Automated test generation, no THCY effort required.

⁴ 0.00192 THCY effort

⁵ 0.0288 THCY effort

easily incorporate changes into the models to keep pace with changing feature requirements.

Preliminary data indicates that using TestMaster to automate our test generation process can increase our productivity by over 80%, while providing a more effective way to analyze complex requirements.

Adequacy Criteria for Object Testing

Brigid Haworth

brigid@bournemouth.ac.uk

Adequacy Criteria - Overview

- **Test Data Generation**
- **Test Adequacy Measurement**
 - **Specification Based**
 - **Program Structure Based**

Test Adequacy Measurement

- **Management and control of software testing**
 - **Structural Coverage analysis techniques**
 - **Static Analysis/Dynamic Analysis**
 - **Supported by test tools**

Criteria for OO Software

- **Consider**
 - **choice of component for "unit" test**
 - **underlying structure of component**
 - **structural coverage criteria development**

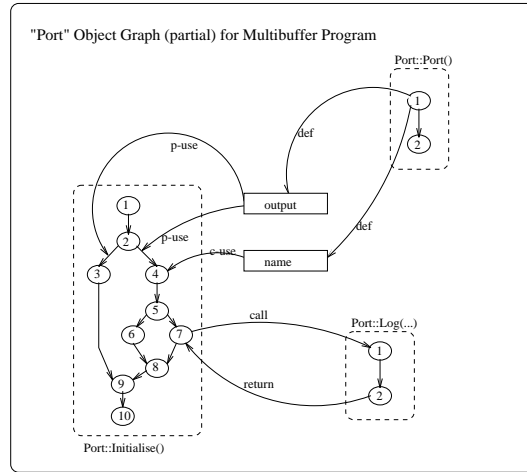
OO Unit Testing

- **an "object" as basic unit of test**
- **structure based on methods and data**
- **new and inherited object components**
- **new criteria required for object structure coverage**

Object Structure

- **dynamic view**
- **control flow based within methods**
- **control flow AND data flow based between methods**
- **unified "flow" based model developed**

Graphical Representation



Test Model Object Flows

- **triple format** $\langle M1, C, M2 \rangle$
- **M1, M2 represent method points**
- **C represents flow connection**

Object Level Grammar

```
object_flow --> <m_point,connection,m_point>
m_point --> ancestor_m_point | local_m_point
ancestor_m_point --> 'inh_m_point' | 'inh_virtual_m_point'
local_m_point --> 'new_m_point' | 'overriding_m_point'

connection --> data | 'direct'

data --> ancestor_d | local_d
ancestor_d --> 'inh_d'
local_d --> 'overriding_d' | 'new_d'
```

Object Level Flow Types

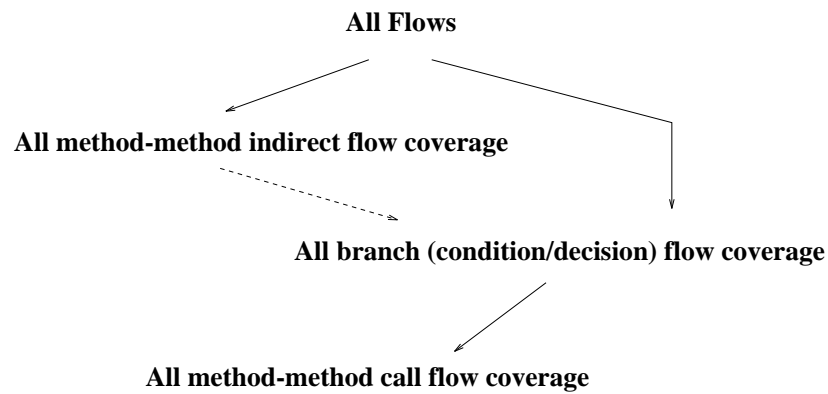
```
ancestor_m_point 'direct' ancestor_m_point
ancestor_m_point 'direct' local_m_point
local_m_point 'direct' ancestor_m_point
local_m_point 'direct' local_m_point

ancestor_m_point ancestor_d ancestor_m_point
ancestor_m_point ancestor_d local_m_point
local_m_point ancestor_d ancestor_m_point
local_m_point ancestor_d local_m_point
local_m_point local_d local_m_point
```

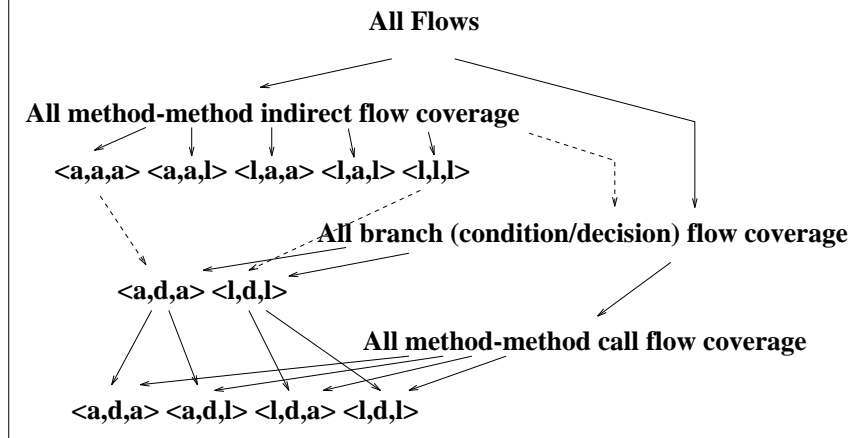
Object Coverage Criteria

- **intra-object method-method call flow coverage**
- **intra-object branch (decision/condition) flow coverage**
- **intra-object method-method indirect flow coverage**
- **intra-object all flows coverage**

Criteria Hierarchy



Criteria Hierarchy - refined



Examples

Refer to Appendix A

Appendix A

```
// (c) 1997 Tiger Communications plc
//=====
// TigStreamProcess constructor
//=====
TigStreamProcess::TigStreamProcess
( char *conf_name          // Name of config file.
)
{
    .                               // Nodes
    .                               // 1 SNode
    .                               //

    conf_file = new CONF_FILE(conf_name);          //

    input = TigIOPort::CreatePort(conf_file, "Input");    //
    output = TigIOPort::CreatePort(conf_file, "Output");  //
    control = TigIOPort::CreatePort(conf_file, "Control"); // 1, def(control)
    monitor = TigIOPort::CreatePort(conf_file, "Monitor");
    rawdata = TigIOPort::CreatePort(conf_file, "RawData");
    alarmport = TigIOPort::CreatePort(conf_file, "Alarm");

    .
    .
    .
}

//=====
// ProcessTick - idle process while nothing is happening.
// Override this to provide checks when no data received.
// But don't forget to call it!
//=====
void TigStreamProcess::ProcessTick(void)
{
    // Call the idle method for all existing I/O ports.
    input->Tick();          // Nodes
                          // 1 SNode

    if ( output != NULL ) // 2 PNode
        output->Tick();   // 3 SNode
                          // 4 EndPNode

    if ( control != NULL ) // 5 PNode with pp-use(control)
        control->Tick();
    if ( monitor != NULL )
        monitor->Tick();
    if ( rawdata != NULL )
        rawdata->Tick();
    if ( alarmport != NULL )
        alarmport->Tick();
}
}
```

Figure 1: Indirect flow example for <a,a,a> type

```

// (c) 1997 Tiger Communications plc
//=====
// TigStreamProcess constructor
//=====
TigStreamProcess::TigStreamProcess
( char *conf_name          // Name of config file.
)
{
    .                      // Nodes
    .                      // 1 SNode
    .                      //

    conf_file = new CONF_FILE(conf_name);          // 1, def(conf_file)

    input = TigIOPort::CreatePort(conf_file, "Input");
    output = TigIOPort::CreatePort(conf_file, "Output");
    control = TigIOPort::CreatePort(conf_file, "Control");
    monitor = TigIOPort::CreatePort(conf_file, "Monitor");
    rawdata = TigIOPort::CreatePort(conf_file, "RawData");
    alarmport = TigIOPort::CreatePort(conf_file, "Alarm");

    .
    .
    .
}

//=====
// Multibuffer constructor
//=====
Multibuffer::Multibuffer(char *conf_name) : TigStreamProcess(conf_name),ports()
{
    char *pname;
    int l;

    action_on_unknown_port = conf_file->GetInt("Input", "OnUnknownPort", 0); //Nodes //1, c-use(conf_file)
    port_change_string = UnescapeString(conf_file->GetString("Input",
        "PortChangeString"));

    port_str_ptr = port_change_string;

    .
    .
    .
}

```

Figure 2: Indirect flow example for <a,a,l> type

```

// (c) 1998 Tiger Communications plc
//=====
// ProcessStream - Open the input stream, then process it.
//=====
void Multibuffer::ProcessStream(void)
{
    if ( input == NULL || outbuf == NULL || ident_buf == NULL )           // Nodes
        return;                                                            // 1,2,3
                                                                            // 4
                                                                            // 5

    if ( input->Open() )                                                    // 6
    {
        terminate = FALSE;                                                // 7 def(terminate)
        for ( current_port = (MultibufferPort *)ports.Head();
              current_port != NULL;
              current_port = (MultibufferPort *)current_port->Next() )
        {
            if ( current_port->output == NULL || !current_port->output->Open() )
            {
                terminate = TRUE;
            }
        }
        current_port = NULL;

        while ( !terminate )                                               // 16 pp-use(terminate)
        {
            inbuf_count = input->Read(inbuf, inbuf_size);

            ProcessAnyControlMessages();

            .
            .
            .
        }
    }
}

```

Figure 3: Indirect flow example for <l,a,l> type

```

// (c) 1998 Tiger Communications plc
//=====
// Multibuffer constructor
//=====
Multibuffer::Multibuffer(char *conf_name) : TigStreamProcess(conf_name), ports()
{
    char *pname;
    int l;

                                                                    // Nodes
    .
    .
    .

    outbuf_len = conf_file->GetInt("Output", "BufferSize", 1024);
    if ( outbuf_len < 20 )
        outbuf_len = 20;
    outbuf = new char[outbuf_len];                                                                    // 10, def(outbuf)
    if ( outbuf == NULL )
    {
        logprintf("Failed to allocate memory for output buffer.");
    }
    .
    .
    .
}

//=====
// FlushOutput - write the buffered output to the port.
//=====
{
                                                                    // Nodes
    .
    .
    .

        output->Write("\n", 1);

        written_to_def = TRUE;
    }
    output->Write(outbuf, outbuf_count);                                                                    // 11, c-use(outbuf)
    output->Close();

    .
    .
    .
}

```

Figure 4: Indirect flow example for <1,1,1> type

Adequacy Criteria for Object Testing

Brigid Haworth

Department of Computing, Bournemouth University,
Talbot Campus, Fern Barrow,
Poole. BH12 5BB.
brigid@bournemouth.ac.uk

Abstract

In this report, criteria for adequacy measurement for objects in OO software are presented. Coverage criteria that address internal object structure have been developed. These criteria include coverage of direct and indirect inter-method flows in addition to internal method control flows. Results from the analysis of a commercial system developed in C++ support the theory that coverage analysis based on methods alone is insufficient for object level coverage measurement.

keywords: object coverage analysis, adequacy criteria, structural testing, object oriented

1 Introduction

Coverage analysis techniques are commonly advocated as a useful approach to assess test adequacy. These techniques can be used to aid the management and control of software testing for projects [6]. Adequacy criteria can provide test managers with measures to use as indicators of the thoroughness of the testing performed. These criteria may be specified in terms of coverage for the component that is subject to test. The coverage analysis techniques may

apply to specification of the test component or to the internal structure of the component [9]. Such techniques are still being explored for OO systems [3][8]. In [8], graph representations for the four types of classes defined in [1] are used as a basis for the definition of specification based criteria. In our earlier work [3], a 3-level model upon which to base structural coverage criteria was explored. In this earlier study, the motivation for the development of coverage techniques based on method interactions in the form of flows was confirmed. The model has since been refined in order to improve the granularity at which the flows are detected (by static analysis) and measured (by dynamic analysis). We now consider the flow from the point in a method where it occurs to a destination point in another (or the same) method. This improves the earlier approach where only the originating method and destination method were recorded. Based on the extended model, a hierarchy of criteria for coverage of objects has been defined. These coverage criteria are defined in terms of flows that are modelled as triples; this includes both control flow style flows and data flow style flows in a single form. Using this common basis the criteria may be compared using the “subsumes” relation-

ship as for example in [9]. Section 2 describes the object level test model that is used as a basis for the criteria. These criteria are defined together with the relationship between them in section 3. A code sample from a commercial development in C++ is used in order to illustrate the types of the criteria in section 4. Finally, results of the static analysis given in section 4 are used to present some conclusions.

2 Object Level Test Model

The test model for OO software has been developed in order to facilitate the development of new testing techniques and adequacy criteria. These are needed in order to provide the tester with methods that are appropriate for use in an object-oriented environment. In particular, the object level test model provides a view of the internal object structure that may be used to develop structural techniques and coverage criteria for objects. For testing purposes, objects may be treated as components that can be tested in isolation and therefore such techniques and criteria need to address combinations of methods and data in a way that current component testing techniques do not. In addition, the new techniques and criteria should in some way account for inherited tested features, or at least provide the mechanism to recognise when tested features form part of the new component. This is important when the efficiency of testing object-oriented components needs to be considered. The structural view of objects and the object level test model developed are described in the following subsections.

2.1 Object Structure

An object is considered to have method components and data components. Each of these types of components and the connections between them provide the basis for a structural view of objects. The components may be related in a number of ways. Data components may be connected to other data components, method components may be connected directly to other method components, or data and method components may be interconnected. For example, data components may be linked within an object via data references, through use of aliases, pointers or arrays referring to object data. Methods may be connected by use of inter method calls. Methods and data may be linked through method definitions of, and references to, data.

For dynamic testing, the interconnections that may be traversed during program execution provide the basic structures of interest. This includes both control flow paths and data flow paths. Method control flow paths may be modelled using flow graphs in the way that these have been for functions and procedures. Between methods within an object, inter method control flows may be modelled by extending the program graphs to include links between the node where a call occurs and the corresponding start and end node from the called method. Data flow style connections may be modelled by including links between nodes in methods where data definitions occur and the object data defined and also between data object and the corresponding nodes (c-use) or edges (p-use) in a method where a use occurs. An object flow graph may be constructed to model all of these flows providing a graphical representation of the object in terms useful for testing purposes. An example of such a graph is shown in Figure 1.

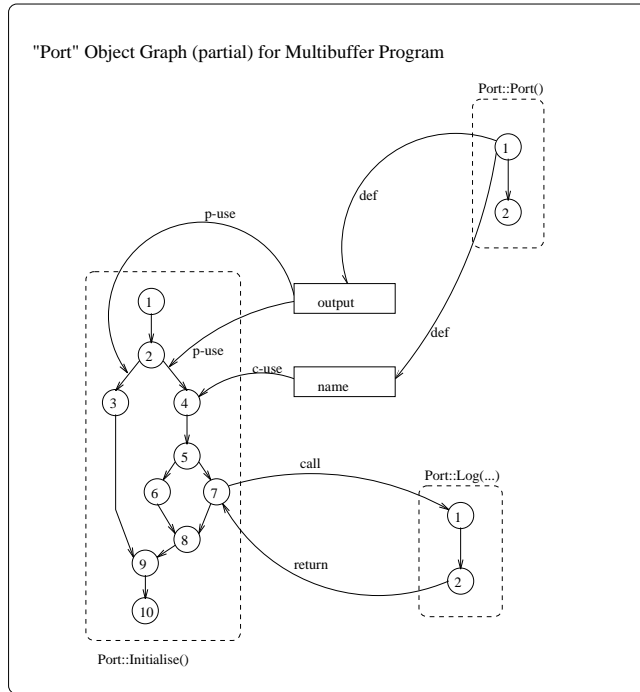


Figure 1: Intra-Object Flowpaths

The graph shows control flow graphs for three methods in a “Port” object from the “Multibuffer” program that is used later in the examples section. The nodes in these graphs represent sequences of executable code within the method and the edges represent the flow control. “Port” data elements “output” and “name” are represented by solid line rectangles. Edges occurring between the method graphs and the data elements show the different interactions that occur. The method “Port::Port()” defines both of the data elements during the object construction. This is shown by the edges labelled ‘def’ between the method and the data elements. The “Port::Initialise()” method has a predicate use of the data element “output” resulting in p-uses

on the emanating edges from its predicate node number 2. Node number 4 has a c-use of the data element “name”. Node number 7 shows a direct call to the “Port::Log(…)” method. This results in an edge labelled ‘call’ from node 7 of “Port::Initialise()” to node 1 of “Port::Log(…)” and a return edge from node 2 of “Port::Log(…)” to node 7 of “Port::Initialise()”.

Previously, class graphs have been used to model this flow information for individual classes [4] [7]. It is proposed to extend this through class hierarchies by considering the flow internal to an object. This differs from the incremental view proposed in [4] and is based on the view that a system’s objects represent the fundamental components for testing purposes. In this case it is

preferred to have a single model that addresses the testing of objects whether or not inherited classes form part of the object structure.

2.2 Test Model Object Flows

A flow is represented by a triple in the form $\langle M1, C, M2 \rangle$ where $M1$ and $M2$ are method points and C is a flow connection occurring between these method points. The flow connection may be in the form of a control flow e.g. a direct call from the point $M1$ to the point $M2$ or a branch occurring at point $M1$ that is followed by the point $M2$. Alternatively a flow may be in the form of a data flow e.g. a data element is defined at point $M1$ and used at point $M2$, and the resulting flow path from $M1$ to $M2$ is definition clear with respect to that data element.

In abstract terms the test model for an object is viewed as a set of flows. The flows may be categorised in a way that aids the tester in the determination of test requirements for an object with inherited tested features. This shows the tester which of these inherited features may require further testing due to interaction with newly defined features. The extent to which inherited object features are “preserved” i.e. are embedded as is, can also be shown by these categories. This information may be regarded as an indicator of the “testability” of the implementation of the object in the sense that it may be desirable to limit interaction between the new features and the inherited features to the inherited object’s interface. This “preservation” of the inherited object can reduce the testing effort for the new object. This style of implementation retains the generalisation/specialisation relationship that is desirable in an OO system without compromising the integrity of the inherited object.

The categories for the flows are defined by the view of the flow with respect to the current object. The method points $M1$ and $M2$ are tagged in accordance with the origin of their declaration. These are either ancestor method points i.e. declared in an inherited object or are local method points i.e. declared in the current object. The connection between the method points may be direct or indirect via data. In the latter case the connection is also categorised in accordance with the origin of the declaration. The data is either ancestor data or local data as in the case of the method points. These terms are more formally defined;

```
object_flow --> <m_point,connection,m_point>
m_point --> ancestor_m_point | local_m_point
ancestor_m_point --> 'inh_m_point' |
                    'inh_virtual_m_point'
local_m_point --> 'new_m_point' | 'overriding_m_point'
connection --> data | 'direct'
data --> ancestor_d | local_d
ancestor_d --> 'inh_d'
local_d --> 'overriding_d' | 'new_d'
```

The number of possibilities of the types of flow that may be determined from the grammar above is, however, greater than the number that can occur in practice. This may be restricted by a set of categories i.e. we can define categories of flows based on the legal options that are determined by programming language rules. The following restricted set of categories are thus defined;

```
ancestor_m_point 'direct' ancestor_m_point
ancestor_m_point 'direct' local_m_point
local_m_point 'direct' ancestor_m_point
local_m_point 'direct' local_m_point
ancestor_m_point ancestor_d ancestor_m_point
ancestor_m_point ancestor_d local_m_point
local_m_point ancestor_d ancestor_m_point
local_m_point ancestor_d local_m_point
local_m_point local_d local_m_point
```


The test model instance for a particular object consists of a set of these flows defined in the form of a set of triples. This set of triples is determined from source code using static analysis techniques. The flows may then be categorised in accordance with the restricted set described above. These categories not only serve to specify legal types of flows occurring in objects but also serve to provide type information within the test model that may be used in determining coverage requirements for an object that is subject to test. Categories that are entirely inherited i.e. with all ancestor components as for example in flow types $\langle a,d,a \rangle$ and $\langle a,a,a \rangle$, indicate an opportunity to reduce testing effort in an object. Test cases previously executed on flows of these types may be omitted entirely without loss of coverage of the object structure. Alternatively a selection of cases may be re-used, for example in the case where it is desirable to re-test inherited features in their new context e.g. when testing the impact of execution in an object with increased memory requirements. In terms of coverage of program structure nothing more is gained but there may be benefit from the re-test when other such test objectives are considered. The decision to re-test or not then becomes a matter for risk analysis where the additional costs of re-test may outweigh the possible benefits from doing so.

3 Criteria Definitions

The coverage criteria we propose for objects are;

- intra-object method-method call flow coverage
- intra-method branch (decision/condition) flow coverage

- intra-object method-method indirect flow coverage
- intra-object all flows coverage

Each of these criteria may be further refined to account for the object level categories. This shows how the impact of inheritance can be addressed in each case.

The criteria together with their refinements are defined as follows;

3.1 Intra-object method-method call flow coverage

This coverage criterion requires all internal method-method calls to be executed at least once;

Defn A set of execution paths P satisfies *intra-object method-method call flow coverage* if for each method M , for all nodes n in the method control flow graph for M that contain an intra-object method-method call, there is at least one path p in P such that p contains n .

This criterion can be refined by considering the following categories from the model:

```
ancestor_m_point 'direct' ancestor_m_point
ancestor_m_point 'direct' local_m_point
local_m_point 'direct' ancestor_m_point
local_m_point 'direct' local_m_point
```

The method-method call described in the criterion can be replaced by these four types of possible call. The method-method call in general maps to the idea of a triple in the form:

```
caller_method_point 'direct' called_method_point
```

The `caller_method_point` occurs at a node on the control flow graph where the method-method call occurs. The called method point occurs at the start node of the control flow graph for the called method.

3.2 Intra-method branch (decision/condition) flow coverage

This coverage criterion requires the method control flow graph to be constructed with separate nodes for each condition in an expression used in a decision.

Defn A set of execution paths P satisfies *intra-method branch (decision/condition) flow coverage* if for all edges e in the method control flow graph there is at least one path p in P such that p contains e .

This criterion can be refined by considering the following categories from the model:

```
ancestor_m_point 'direct' ancestor_m_point
local_m_point 'direct' local_m_point
```

In this case, a branch occurs on a direct flow between two method points from within a single method. This can be an inherited method giving rise to $\langle a,d,a \rangle$ flows or a newly defined method giving rise to $\langle l,d,l \rangle$ flows.

3.3 Intra-object method-method indirect flow coverage

This coverage criterion requires all method-data-method flows to be exercised at least once. These flows occur on execution paths between methods in a definition-use style manner. The criterion requires that the methods will be executed in such a way that an execution path from the

point of definition of the object data to the point of use of the object data will be executed. This execution path must be definition clear with respect to that object data.

Defn A set of paths P satisfies *intra-object method-method indirect flows coverage* if for all methods $M1$ and $M2$, $M1$ not necessarily distinct from $M2$, for all nodes $n1$ in the method control flow graph for $M1$ that contain an object data definition, for all nodes $n2$ in the control flow graph for $M2$ containing object data c -uses and all edges e in the control flow graph for $M2$ containing object-data p -uses, there is at least one path p in P such that p includes a subpath through which the definition of the object data reaches its use.

This criterion can be refined by considering the following model categories:

```
ancestor_m_point ancestor_d ancestor_m_point
ancestor_m_point ancestor_d local_m_point
local_m_point ancestor_d ancestor_m_point
local_m_point ancestor_d local_m_point
local_m_point local_d local_m_point
```

3.4 Intra-object all flows coverage

This coverage criterion requires that all flows of the forms identified above should be exercised at least once. The indirect flows may in some cases fail to subsume the branch (decision/condition) coverage criterion. All flows requires indirect flows and additional flows that ensure branch coverage is achieved.

Defn A set of paths P satisfies *intra-object all flows coverage* if for all methods $M1$, and $M2$, $M1$ not necessarily distinct from $M2$, for all nodes $n1$ in the method control flow graph for $M1$ that contain an object data definition and for all nodes $n2$ in the method control flow graph

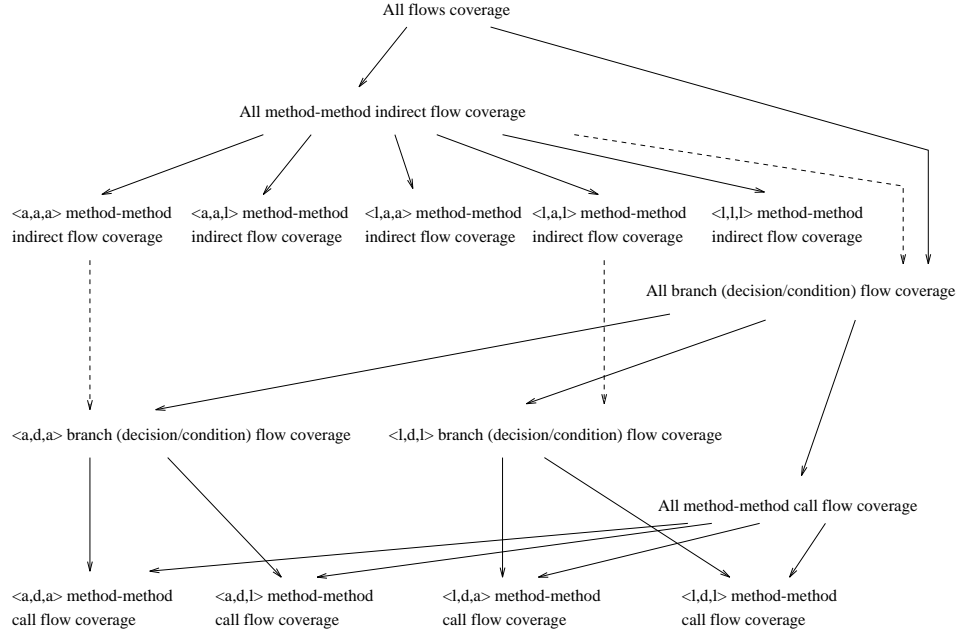


Figure 2: Object Level coverage criteria - subsumes hierarchy

for M2 that contain object data c-uses and all edges e in the graph for M2 that contain object data p-uses, there is at least one path p in P such that p includes a subpath through which the definition of the object data reaches its use; additionally, for all methods M , for all edges e in the method control flow graph that do not contain a p-use of object data, there is at least one p in P such that p contains e .

In each of the criteria above the execution paths must be feasibly executable.

3.5 Relationship between Criteria

The relationship between the criteria is defined in terms of the coverage of object structure achieved when test cases executed are adequate

with respect to given criteria. A subsumes relationship as defined in [9] is used to develop a hierarchical view of the coverage criteria proposed for objects.

The following subsumes relationship holds for object level coverage analysis;

- intra-method level branch (decision/condition) coverage subsumes direct method flows
- indirect method flow coverage subsumes intra-method level branch (decision/condition) coverage if
 - each condition of a decision in a method has an object data reference i.e. uses some object data to determine the outcome of the decision

- all flows coverage subsumes all indirect method flow coverage and also all intra-method branch (decision/condition) coverage

The subsumes hierarchy is shown in Figure 2. In this figure, the subsumes relation is depicted by a solid arrow line. Dashed arrow lines show the subsumes relation between criteria that holds when the assumptions described above are true.

4 Examples

The analysis of a class hierarchy developed in C++ is used here to illustrate the model concepts and the criteria definitions. An overview of the hierarchy is shown in Figure 3. Note that lines with arrows denote the inheritance relation while lines without arrows denote associations. This overview includes those classes that form part of the inheritance structure for the “Multibuffer” program.

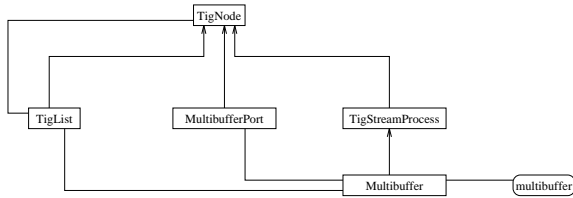


Figure 3: Multibuffer class tree overview

The process used to determine flows for a “Multibuffer” object is described and is illustrated in detail through the analysis of the “Multibuffer::FlushOutput” method. In the following description, nodes of the control flow graph may be

- SNodes i.e. representing a sequence of code up to but excluding any expression part of a predicate
- PNodes i.e. representing an expression that determines some alternative control flow sequence
- EndPNode i.e. representing a point where alternative control flow paths merge

The source for the “Multibuffer::FlushOutput” method together with the control flow graph is shown in Figures 4 and 5.

The notion of a potential p-use (pp-use) of a data element is also used in the creation of the node lists. This represents the use of a data element in a predicate node PNode and which is usually referred to as a p-use of the data but attributed to the emanating edges from the PNode. This pp-use is transformed to the possible alternative p-uses in the step determining the triples.

4.1 Process steps

1. For each class in the object hierarchy;
 - 1.1 For each method in each class;
 - 1.1.1 construct the method control flow graph with numbered nodes for each method point
 - 1.1.2 for each node of the method control flow graph construct an ordered list of data definitions (def) uses (c-use, pp-use) and calls
 - 1.1.3 for each node construct a next node list
 - 1.1.4 tag each member documented in the lists as ancestor if declared in a parent class or local if declared in the current class
 2. For each class
 - 2.1 For each local data member in the class;
 - 2.1.1 construct data flow triples from method node lists

```

// (c) 1998 Tiger Communications plc

//=====
// FlushOutput - write the buffered output to the port.
//=====
void Multibuffer::FlushOutput(void) //Nodes
{
    if ( current_port == NULL ) // 1
    {
        if ( output != NULL && output->Open() ) // 2,3
        {
            if ( !written_to_def ) // 4
            {
                // This is the first time we've written to the default
                // output port this invocation, so we'll write a
                // date stamp to separate it from any previous stuff.
                time_t now = time(NULL); // 5
                char *str;

                str = "\n\n=====" " ;
                output->Write(str, strlen(str));

                str = ctime(&now);
                output->Write(str, strlen(str));
                output->Write("\n", 1);

                written_to_def = TRUE;
            } // 6
            output->Write(outbuf, outbuf_count); // 7
            output->Close(); // 8
        }
    }
    else
    {
        current_port->output->Write(outbuf, outbuf_count); // 9
    } // 10
    outbuf_count = 0; // 11
    outbuf_ptr = outbuf;
}

```

Figure 4: Multibuffer::FlushOutput source

- | | |
|--|--|
| <ul style="list-style-type: none"> 2.2 For each inherited data member accessed in the class; 2.2.1 construct data flow triples from node lists of current class and inherited class 2.3 For each method in the class construct the method-method direct calls from the node lists 2.4 For each method in the class construct the branch list from the node lists | <ul style="list-style-type: none"> 3. Determine the triples for the object 3.1 Determine triples blocked or changed by scoping for the object <ul style="list-style-type: none"> 3.1.1 For each inherited triple representing direct method-method calls <ul style="list-style-type: none"> 3.1.1.1 Examine method point 1 for points in a virtual method that is overridden in the object and mark for deletion unless there is an explicit |
|--|--|

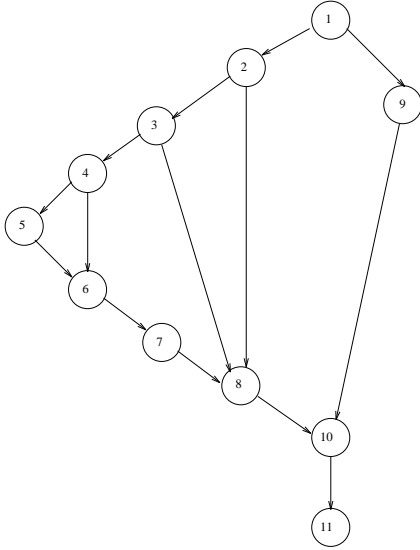


Figure 5: Multibuffer::FlushOutput control flow graph

call to the overridden method

3.1.1.2 Examine method point 2 for points in a virtual method that is overridden in the object and re-classify as a,d,l (this accounts for picking up the overriding method due to the object instance)

3.1.2 For each inherited triple representing branch flows

3.1.2.1 Examine method points for points in a virtual method that is overridden in the object and mark for deletion unless there is an explicit call to the overridden method

3.1.3 For each inherited triple with data connection

3.1.3.1 Examine method point 1 for points in virtual functions that are overridden in the object and mark for deletion unless there is an explicit call to the overridden method

3.1.3.2 Examine method point 2 for points in

virtual functions that are overridden in the object and mark for deletion unless there is an explicit call to the overridden method

3.2 Remove triples marked for deletion

4.2 Flows analysis

4.2.1 Multibuffer::FlushOutput node list

```

Node 1 : PNode, pp-use(current_port,l)
Nextnodes : 2,9
Node 2 : PNode, pp-use(output,a)
Nextnodes : 3,8
Node 3 : PNode, pp-use(output,a)
Nextnodes : 4,8
Node 4 : PNode, pp-use(written_to_def,l)
Nextnodes : 5,6
Node 5 : SNode, c-use(output,a), c-use(output,a),
          c-use(output,a), def(written_to_def,l)
Nextnodes : 6
Node 6 : EndPNode
Nextnodes : 7
Node 7 : SNode, c-use(output,a), c-use(output,a)
Nextnodes : 8
Node 8 : EndPNode
Nextnodes : 10
Node 9 : SNode, c-use(current_port,l)
Nextnodes : 10
Node 10 : EndPNode
Nextnodes : 11
Node 11 : SNode, def(outbuf_count,l), c-use(outbuf,l),
          def(outbuf_ptr,l)
Nextnodes : 0
Branches: 8
  
```

4.2.2 Flows examples

Each of the following flows are examples of the different types may occur and that exist in the C++ system. Direct calls and branch type flows are easily seen within specific methods and are not illustrated in this report. (Although the l,d,l type branch flow given below can be seen in the FlushOutput method source in Figure 4.) Code samples for the indirect flow types are given in Appendix A, Figures 6 to 9.

```

Flows - direct calls
<ProcessControlBlock:5,direct,ProcessControlMsg><a,d,a>
<ProcessBlock:5,direct,ProcessChar>
<Multibuffer:1,direct,TigStreamProcess:1>
<ProcessChar:12,direct,FlushOutput:1>
Flows - branches
<TigStreamProcess:2,direct,TigStreamProcess:3>
<FlushOutput:2,direct,FlushOutput:8>
Flows - indirect
<TigStreamProcess:1,control,ProcessTick:5,6>
<TigStreamProcess:1,conf_file,Multibuffer:1>
none
<ProcessStream:7,terminate,ProcessStream:16,17>
<Multibuffer:10,outbuf,FlushOutput:11>

```

```

Type
<a,d,a>
<a,d,1>
<1,d,a>
<1,d,1>
Type
<a,d,a>
<1,d,1>
Type
<a,a,a>
<a,a,1>
<1,a,a>
<1,a,1>
<1,1,1>

```

case where an object has many “simple” methods i.e. methods with few branches, it may be more efficient to consider the testing of these methods and their interactions together within the context of the object rather than attempting test case design focused on the individual methods. The criteria proposed in this report support the coverage analysis requirements both for the individual methods and for the interactions occurring between them.

4.2.3 Flows totals for Multibuffer object

The following summarises the totals for the counts of flows found in the Multibuffer object.

```

Class Flows: direct - 12calls+195branches,
             indirect - 247, total 454
Blocked Flows: direct - 5calls+15branches,
              indirect - 33, total - 53
Object Flows: local 454, inherited - 161,
             blocked - 53, total - 562

```

These counts show a significant number of flows that should be covered during testing. However, it is not proposed that a single test case is needed for every flow. A test case executed for a single method may cover several branches. Similarly, a test case executed for an object that includes calls to more than one method may cover several indirect flows. The significance of the figures is in the indication that there are many possibilities for methods to interact indirectly via object data. These interactions are not always obvious and may be missed during test case design, especially in the case where the focus of the testing and the coverage measurement is on individual methods. The static analysis may be used to drive the design of test cases in terms of the ordering needed for method invocations. In the

5 Conclusion

In this report a hierarchy of criteria for structural coverage analysis of objects has been presented. The analysis of a commercial system developed in C++ was used to support the motivation for the development of these criteria. This analysis shows a significant number of object flows that occur between methods that are not detectable by method coverage analysis alone. These are indirect flows between methods via object-level data. Tool support for coverage analysis of OO software primarily targets method coverage [2] although method-method direct calls are traceable by some.

The flows detectable by static analysis can be further analysed and used to assist in determining an ordering of execution of object methods. This kind of “grey-box” approach is similar to that described in [5].

The types used to categorise the flows in the model developed here can provide useful information for the management and control of the testing process. Some savings can be made when inherited tested features form part of a new object. The profile of the object test model derived from the static analysis may be used as an in-

indicator of testability and also to drive the test strategy.

The information provided by the static analysis may be traceable from detailed designs although further work is needed in order to achieve this.

Bibliography

- [1] R. V. Binder. Modal Testing Strategies for Object-Oriented Software. *IEEE Computer*, Vol. 29 (11), pages 97–99, July 1997.
- [2] R. V. Binder. Object-Oriented Testing: What’s New? *Object Magazine*, pages 21–23, July 1997.
- [3] B. Haworth, C. Kirsopp, M. Roper, M. Shepperd and S. Webster. Towards the development of adequacy criteria for object-oriented systems. In *Proceedings of the 5th European Conference on Software Testing Analysis and Review*, pages 417–427, Edinburgh, Scotland, Nov. 1997.
- [4] M. J. Harrold and G. Rothermel. Performing Data Flow Testing on Classes. Technical Report 94-114, Clemson University, Clemson, 1994.
- [5] P. Jüttner, S. Kolb, U. Naumann, and P. Zimmerer. A Complete Test Process in Object-Oriented Software Development. In *Proc. 7th International Quality Week*, San Francisco, CA, May 1994. Software Research Institute.
- [6] E. Kit. *Software testing in the real world : improving the process*. ACM Press, 1995.
- [7] A. S. Parrish, R. B. Borie, and D. M. Cordes. Automated Flow Graph-based Testing of Object-Oriented Software Modules. *Journal of Systems and Software*, Vol. 23 (2), pages 95–103, Nov 1993.
- [8] A. Spillner. Four Kinds of Class Modality Four Kinds of Class Testing? In *Proceedings of the 6th European Conference on Software Testing Analysis and Review*, Munich, Germany. To be published Nov/Dec 1998.
- [9] H. Zhu, P. A. V. Hall and J. H. R. May. Software Test Coverage and Adequacy. In *Technical Report No 94/15*, Open University, Milton Keynes, UK, 1994.
- [10] H. Zhu. A Formal Analysis of the Subsume Relation between software Test Adequacy Criteria. In *Technical Report No 94/18*, Open University, Milton Keynes, UK, 1994.

Appendix A

```
// (c) 1997 Tiger Communications plc
//=====
// TigStreamProcess constructor
//=====
TigStreamProcess::TigStreamProcess
( char *conf_name          // Name of config file.
)
{
    .                               // Nodes
    .                               // 1 SNode
    .                               //

    conf_file = new CONF_FILE(conf_name);           //

    input = TigIOPort::CreatePort(conf_file, "Input"); //
    output = TigIOPort::CreatePort(conf_file, "Output"); //
    control = TigIOPort::CreatePort(conf_file, "Control"); // 1, def(control)
    monitor = TigIOPort::CreatePort(conf_file, "Monitor");
    rawdata = TigIOPort::CreatePort(conf_file, "RawData");
    alarmport = TigIOPort::CreatePort(conf_file, "Alarm");

    .
    .
    .
}

//=====
// ProcessTick - idle process while nothing is happening.
// Override this to provide checks when no data received.
// But don't forget to call it!
//=====
void TigStreamProcess::ProcessTick(void)
{
    // Call the idle method for all existing I/O ports.
    input->Tick();                               // Nodes
                                                // 1 SNode

    if ( output != NULL )                       // 2 PNode
        output->Tick();                          // 3 SNode
                                                // 4 EndPNode

    if ( control != NULL )                      // 5 PNode with pp-use(control)
        control->Tick();

    if ( monitor != NULL )
        monitor->Tick();

    if ( rawdata != NULL )
        rawdata->Tick();

    if ( alarmport != NULL )
        alarmport->Tick();
}
}
```

Figure 6: Indirect flow example for <a,a,a> type

```

// (c) 1997 Tiger Communications plc
//=====
// TigStreamProcess constructor
//=====
TigStreamProcess::TigStreamProcess
( char *conf_name          // Name of config file.
)
{
    .                               // Nodes
    .                               // 1 SNode
    .                               //

    conf_file = new CONF_FILE(conf_name);          // 1, def(conf_file)

    input = TigIOPort::CreatePort(conf_file, "Input");
    output = TigIOPort::CreatePort(conf_file, "Output");
    control = TigIOPort::CreatePort(conf_file, "Control");
    monitor = TigIOPort::CreatePort(conf_file, "Monitor");
    rawdata = TigIOPort::CreatePort(conf_file, "RawData");
    alarmport = TigIOPort::CreatePort(conf_file, "Alarm");

    .
    .
    .
}

//=====
// Multibuffer constructor
//=====
Multibuffer::Multibuffer(char *conf_name) : TigStreamProcess(conf_name),ports()
{
    char *pname;
    int l;

    action_on_unknown_port = conf_file->GetInt("Input", "OnUnknownPort", 0);    //Nodes //1, c-use(conf_file)
    port_change_string = UnescapeString(conf_file->GetString("Input",
        "PortChangeString"));

    port_str_ptr = port_change_string;

    .
    .
    .
}

```

Figure 7: Indirect flow example for <a,a,l> type

```

// (c) 1998 Tiger Communications plc
//=====
// ProcessStream - Open the input stream, then process it.
//=====
void Multibuffer::ProcessStream(void)
{
    if ( input == NULL || outbuf == NULL || ident_buf == NULL )           // Nodes
        return;                                                            // 1,2,3
                                                                            // 4
                                                                            // 5

    if ( input->Open() )                                                    // 6
    {
        terminate = FALSE;                                                 // 7 def(terminate)
        for ( current_port = (MultibufferPort *)ports.Head();
              current_port != NULL;
              current_port = (MultibufferPort *)current_port->Next() )
        {
            if ( current_port->output == NULL || !current_port->output->Open() )
            {
                terminate = TRUE;
            }
        }
        current_port = NULL;

        while ( !terminate )                                               // 16 pp-use(terminate)
        {
            inbuf_count = input->Read(inbuf, inbuf_size);

            ProcessAnyControlMessages();

            .
            .
            .
        }
    }
}

```

Figure 8: Indirect flow example for <l,a,l> type

```

// (c) 1998 Tiger Communications plc
//=====
// Multibuffer constructor
//=====
Multibuffer::Multibuffer(char *conf_name) : TigStreamProcess(conf_name), ports()
{
    char *pname;
    int l;

    // Nodes
    .
    .
    .

    outbuf_len = conf_file->GetInt("Output", "BufferSize", 1024);
    if ( outbuf_len < 20 )
        outbuf_len = 20;
    outbuf = new char[outbuf_len]; // 10, def(outbuf)
    if ( outbuf == NULL )
    {
        logprintf("Failed to allocate memory for output buffer.");
    }
    .
    .
    .
}

//=====
// FlushOutput - write the buffered output to the port.
//=====
{
    // Nodes
    .
    .
    .

    output->Write("\n", 1);

    written_to_def = TRUE;
}
output->Write(outbuf, outbuf_count); // 11, c-use(outbuf)
output->Close();
.
.
.
}

```

Figure 9: Indirect flow example for <l,l,l> type

THE DYNAMIC INFORMATION FLOW TESTING OF AN OBJECT

Bill Bently
μ-Research

<http://www.mu-research.com>
wgb@earthlink.net

Bob Binder
RBSC Corporation

<http://www.rbsc.com>
rbinder@rbsc.com

QUALITY WEEK EUROPE 1998

Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

AA2

INTRODUCTION

INTRA-CLASS TESTING

the central problem of object-oriented testing

- How to test an object (structurally)
- How to choose method sequences

Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

A1

BACKGROUND

- Currently, in OO testing, integration testing is dominant and difficult
- Industry trend is toward component software (JavaBeans™)
- Specification of client objects will be unknown because even the client objects will be unknown.
- Integration testing will not even be possible!

DYNAMIC INFORMATION FLOW ANALYSIS

an advanced form of path analysis

- Facilitates visualization of intra-class paths
- Identifies fundamental method sequences which under composition form necessary test sequences
- Demonstrates that path coverage measurement is necessary for determining the effectiveness of test sequences

INTRODUCE μ -PATH

- Main contribution
- A new conceptual tool for testing
- Represents flows through methods

SCOPE

- Informal theory
- Intuitive presentation using simple Java™ examples
- No automated tool (yet)
- Simplified model
- Limitations inherited from path testing and static analysis

INTRODUCTION

STRATEGY IS BASED ON

“A Theory of Test Efficiency” [Bently 1993]

- Testing is a battle with combinatorics
- Turns conventional testing theory upside down
- What tests are unnecessary?
- What are the fundamental elements of necessary tests?

Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

E2

THEORY

2-SEQUENCE

independent Java methods



sequencing of methods does not affect behavior

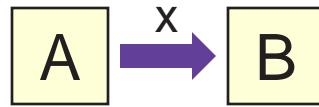
Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

F2

THEORY

2-SEQUENCE

interactive Java methods



shared instance variables create flows

data flow is sufficient for representing flows
between methods

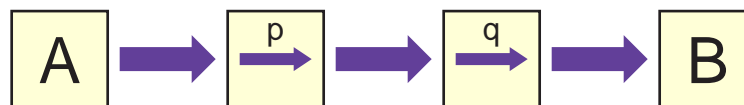
Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

FF1

THEORY

N-SEQUENCE

path through methods is a μ -path



Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

G1

INFORMATION FLOW

builds on control flow and data flow analysis

composable elements

- memory access elements
- data flow elements
- control flow element

two levels

- α intra-method
- μ intra-class

α LEVEL FLOW ANALYSIS EXAMPLE

```

public int add1(int a) { /* segment #1 */
    int b=0;           /* segment #1 */

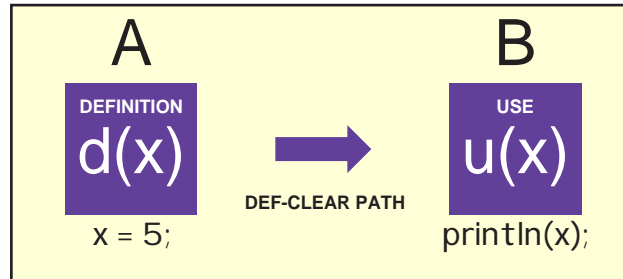
    if(a > 0)          /* segment #2 */
        b = 1 + a;    /* segment #3 */
    return(b);        /* segment #5 */
}

```

DATA FLOW TESTING

data flow strategies are diverse and complex
[Clarke et al. 1986]

du-pair



Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

J3

ELEMENTS OF INFORMATION FLOW ANALYSIS

α memory access

- **definition** value is bound to a variable $d_1(b)$
- **use** value of variable is accessed $u_5(b)$
- **p-use** use contained in a predicate $pu_2(a)$

α data flow

input element \longrightarrow output element

- **du-pair** definition \longrightarrow use $d_1(b) \longrightarrow u_5(b)$
- **ud-pair** use \longrightarrow definition $u_3(a) \longrightarrow d_3(b)$

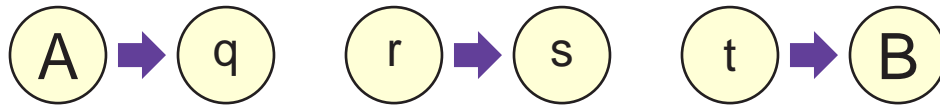
Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

K2

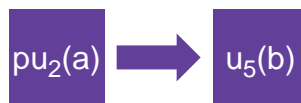
THEORY

THE NEED FOR A MORE GENERAL FLOW ANALYSIS

data flow



example of flow relationship that is not data flow



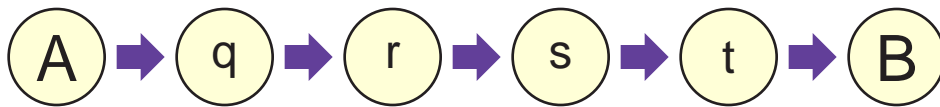
Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

L3

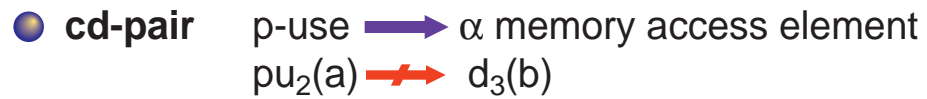
THEORY

ELEMENTS OF INFORMATION FLOW ANALYSIS

α control flow



α control flow



p-use that can block information flows

complementary pair

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

M3



ELEMENTS OF INFORMATION FLOW ANALYSIS

μ level

μ memory access element

- μ -**definition** definition of an instance variable μ -d₅(power)
- μ -**use** use of an instance variable μ -u₅(power)
- μ -**p-use** μ -use contained in a predicate μ -pu₂(safety)



μ data flow element

- μ -**du-pair** μ -definition  μ -use
 μ -d₃(safety)  μ -pu₂(safety)

ELEMENTS OF INFORMATION FLOW ANALYSIS

μ level

μ control flow element

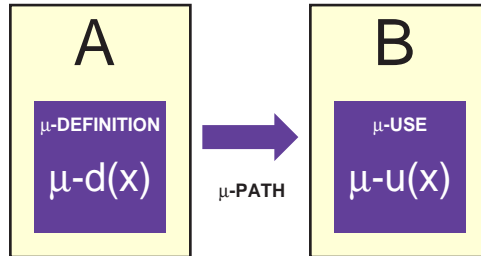
- μ -**ud-pair** μ -use  μ -definition
 μ -d₃(safety)  μ -u₂(safety)
- μ -**cd-pair** occurs when μ -ud-path flows through α -cd-pair

μ complementary pair

- if complement is also a μ -cd-pair

THEORY

μ PATH



generalization of def-clear path

types

- static (structural)
- dynamic (run-time)

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

01

THEORY

DYNAMIC INFORMATION FLOW

Examples of dynamic μ -states

disappearance of μ -u(x)

```
y=1;  
if(y < 0)  
  y=x+2;
```

disappearance of μ -d(x)

```
y=1;  
if(y < 0)  
  x=2;
```

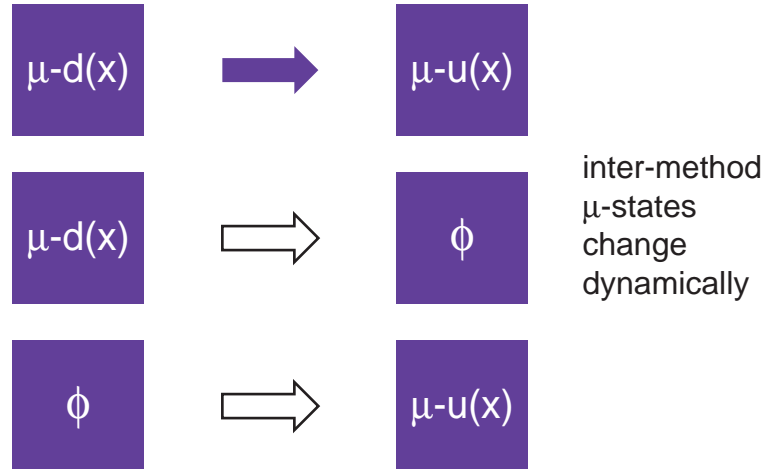
Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

P2

THEORY

DYNAMIC INFORMATION FLOW

μ -flows are dynamic



Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

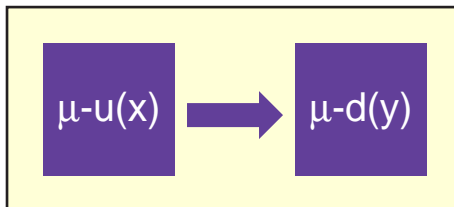
QQ2

THEORY

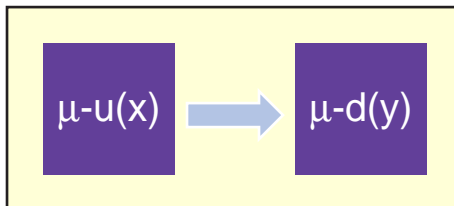
DYNAMIC INFORMATION FLOW

μ -flows are dynamic

intra-method μ -states appear



and disappear at run-time

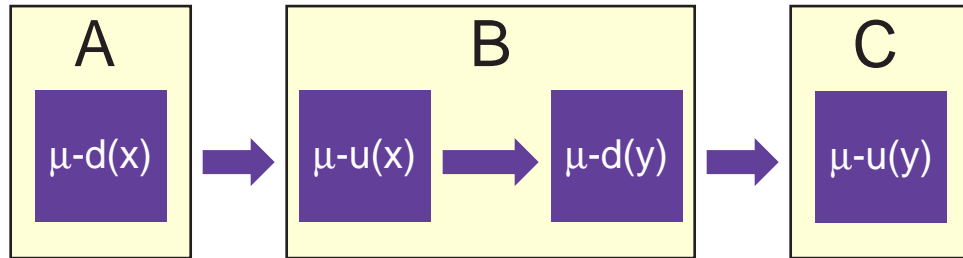


Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

Q2

3-SEQUENCE

Effective testing requires a run-time μ -path between and through methods



testing of μ -ud-pair
 μ_{23} strategy

DYNAMIC INFORMATION FLOW intra-class test strategies

spectrum



- μ_0 all μ memory access elements
- μ_1 all μ data/control flow elements
- μ_{23} all μ 2 and 3-sequences
- μ_{path} all μ -paths
- μ_{*23} all μ^* 2 and 3-sequences
- μ_{allseq} all (necessary) sequences

each method in an effective test sequence is associated with at least one μ -path

THEORY

PRINCIPLES OF INFORMATION FLOW INTRA-CLASS TESTING

#1 - an effective test is a run-time μ -path

#2 - testing a run-time μ -path requires that each μ -cd-pair along path be tested
(insofar as this is possible through sequencing)

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

T1

EXAMPLE

μ LEVEL FLOW ANALYSIS EXAMPLE - 1

```
/* PowerControl V1.0  
ROUGH SPECIFICATION
```

```
This class simulates a simple power control (electric lawnmower etc.)  
with 4 buttons:
```

```
PowerUp button turns power on.
```

```
PowerDown button turns power off.
```

```
Safety button toggles safety on (1) and off (0).
```

```
    If safety is on, the PowerUp and MorePower buttons  
    are deactivated. The initial condition is safety on.
```

```
MorePower
```

```
    Each time the MorePower button is pressed, power level  
    advances one level. There are four power levels: off(0),  
    low(1), medium(2) and high(3).
```

```
*/
```

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

U3

EXAMPLE

μ LEVEL FLOW ANALYSIS EXAMPLE - 2

```
public class PowerControl {
    /* CLASS VARIABLES */
    int power;
    int safety;

    PowerControl(){
        power = 0;
        safety = 1;
    }

    public void PowerUp(){
        if(safety == 0)
            power = 1;
    }

    public void PowerDown(){
        power = 0;
    }

    public void ToggleSafety(){
        if(safety == 0)
            safety = 1;
        else
            safety = 0;
    }
}
```

Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

V4

EXAMPLE

μ LEVEL FLOW ANALYSIS EXAMPLE - 3

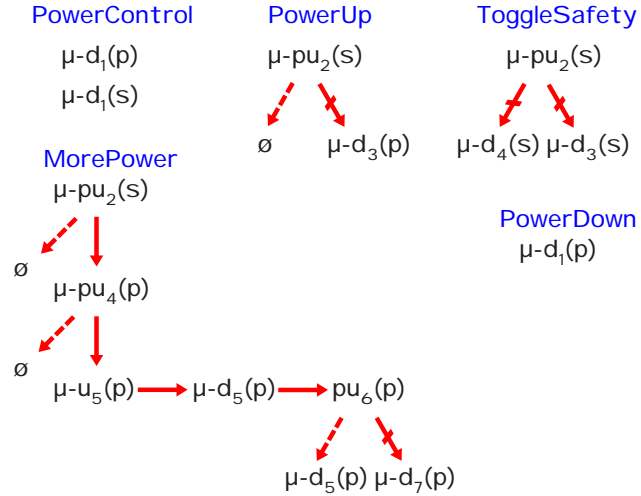
```
public void MorePower(){ /* segment# 1 */
    if(safety == 0) { /* segment# 2 */
        if(power != 0) { /* segment# 4 */
            power = power + 1; /* segment# 5 */
            if(power > 3) /* segment# 6 */
                power = 3; /* segment# 7 */
        }
    }
}
```

Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

W4

EXAMPLE

STEP 1 - CONSTRUCT μ/α GRAPHS

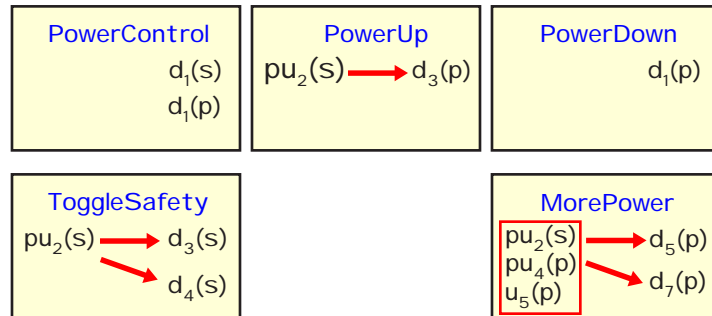


Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

X4

EXAMPLE

STEP 2 - CONVERT μ/α GRAPHS INTO μ -BOXES

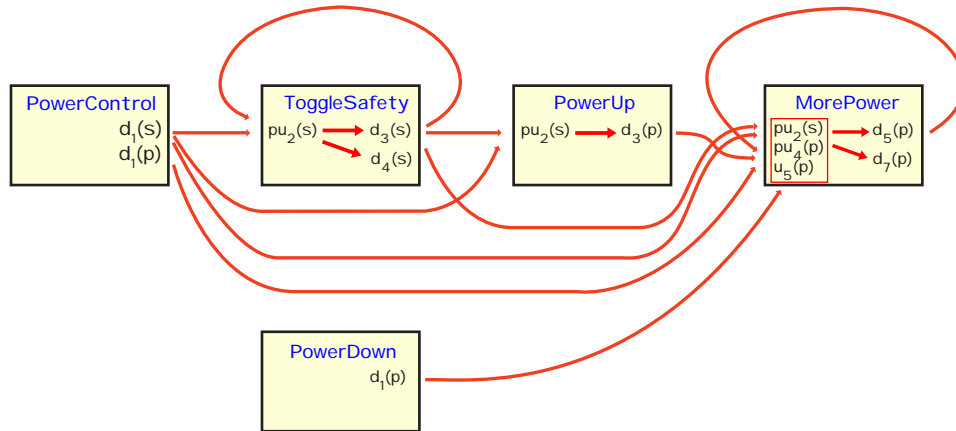


Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

Y3

EXAMPLE

STEP 3 - DERIVE THE μ -PATHS

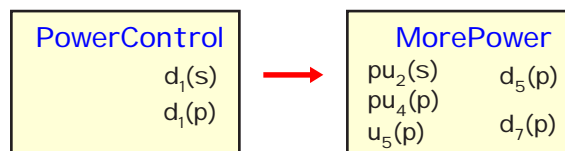


Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

Z3

EXAMPLE

STEP 4 - CONSTRAIN TESTING TO RUN-TIME μ -PATHS



Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

_A3

EXAMPLE

STEP 5 - TEST μ -cd-pairs

μ -pu ₂ (s)	\rightarrow	μ -pu ₄ (p)	either test case below
μ -pu ₄ (p)	\rightarrow	μ -u ₅ (p)	either test case below
pu ₆ (p)	\dots	μ -d ₅ (p)	PC TS PU MP MP
pu ₆ (p)	\rightarrow	μ -d ₇ (p)	PC TS PU MP MP MP MP

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

_B2

EXAMPLE

TEST CASES

PC PD MP

PC MP

PC PU

PC TS MP

PC TS PU MP

PC TS TS MP

PC TS TS PU

PC TS PU MP MP

PC TS PU MP MP MP MP

A simple static data flow strategy, such as all μ -du-pairs, would miss the test cases in the box and include unnecessary test cases (PC PU MP and PC MP MP)

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

_C3

CONCLUSION

DYNAMIC INFORMATION FLOW TESTING

- elucidates the structure of intra-class paths
- identifies necessary fundamental subsequences (but not all necessary sequences)

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

_D1

CONCLUSION

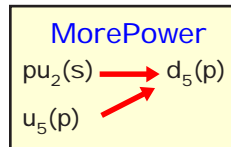
PRINCIPLE #3 OF INFORMATION FLOW INTRA-CLASS TESTING

- To assure effective intra-class testing, path execution must be monitored at run-time

Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

_E1

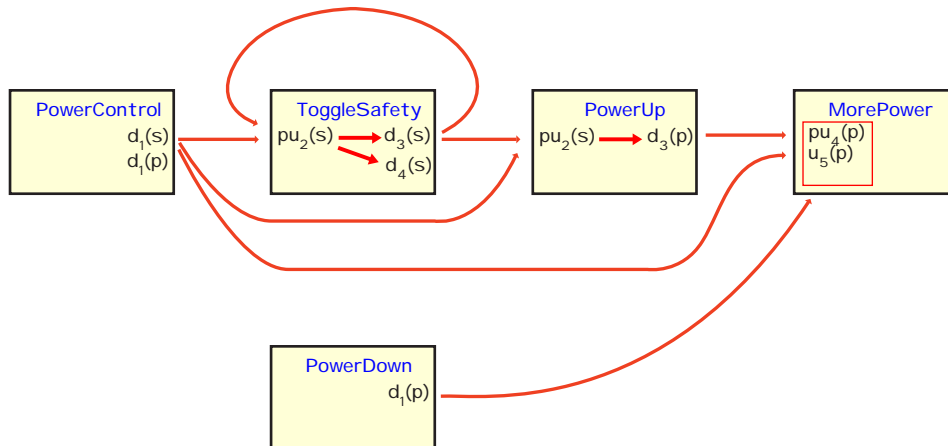
MERGING INFORMATION FLOWS IN MorePower



Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

_F3

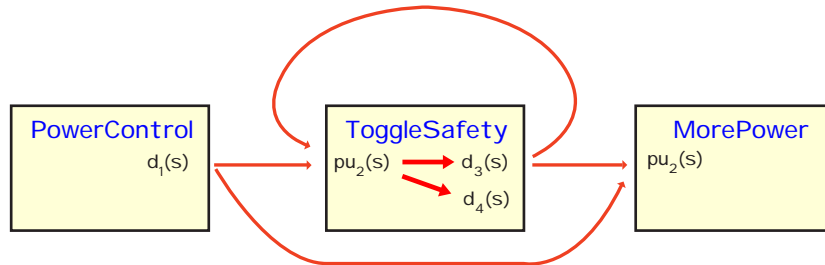
μ -GRAPH OF ELEMENTARY p-INPUT μ -PATHS FOR MorePower



Copyright 1998 RBSC Corporation and μ -Research. All Rights Reserved.

_G4

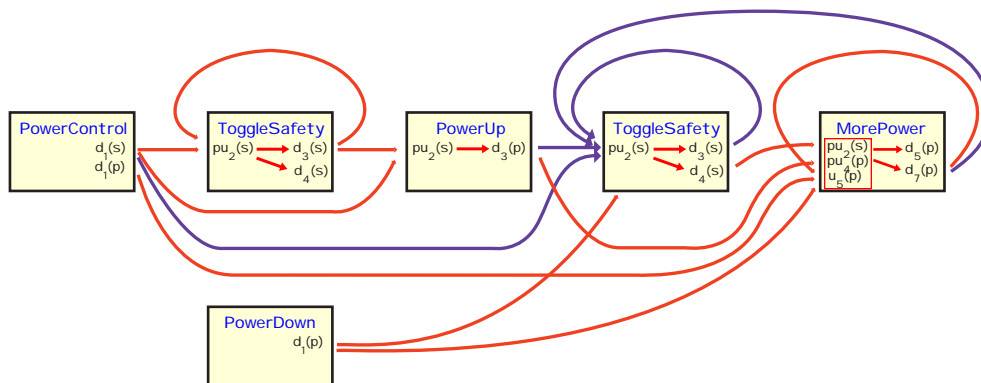
μ-GRAPH OF ELEMENTARY s-INPUT μ-PATHS FOR MorePower



Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

_H4

μ-GRAPH OF MorePower WITH COMBINATION OF ELEMENTARY μ-PATHS



flows corresponding to μ*-ud-pairs are shown in blue

Copyright 1998 RBSC Corporation and μ-Research. All Rights Reserved.

_I5

The dynamic information flow testing of an object

William G. Bently
μ-Research
wgb@earthlink.net

Robert V. Binder
RBSC Corporation
rbinder@rbsc.com

ABSTRACT

The central problem of object-oriented testing is determining method activation sequences for the testing of a class. In the literature, this is called the “intra-class” level of testing. For object-oriented languages such as Java™, specification-based strategies are further developed than structural test strategies. Although both are essential for effective testing, structural strategies are especially well suited for the Java environment, where dynamic linking and bean technology make it virtually impossible to predict the sequences that will be invoked by client objects at run-time.

A testable model of the interaction among class methods is needed for test design, at the class interface level [6], [7]. An orthogonal model of intra-class interactions at the implementation level is necessary to assess the adequacy of a test suite at class scope [5]. This paper extends the class scope implementation model by developing information-flow paths [4] from the class flow graph.

Information flow analysis elucidates the implicit paths that result as methods access instance and class variables. This theory is capable of identifying fundamental subsequences which can be composed to produce almost all necessary method activation sequences.

A surprising result of this theory, is that a test that exercises a particular sequence may not be an effective test of that sequence. The theory indicates that a special form of path coverage must be performed during testing to assess the effectiveness of tests.

The exposition of the theory will be informal, using simple Java examples to demonstrate the basic concepts.

INTRODUCTION

Integration testing predominates during the testing of applications written in object-oriented languages such as Java. Integration testing could be eased and development costs reduced if we had an effective technique for testing an object prior to integration. But how should an object be tested?

In this paper, we introduce a new structural strategy for the testing of an object. It addresses the central problem of object-oriented testing: the selection of method

sequences. This strategy applies the basic concepts of dynamic information flow testing [4] to the class scope implementation model [5].

The intra-class flow elements, introduced in this paper, are abstractions of the flow elements described in earlier papers on information flow testing [2,3,4]. These new elements yield insight into the structure of intra-class paths.

scope

As a form of path testing, dynamic information flow testing shares the primary limitations of path testing strategies:

Sufficiency As compared to exhaustive testing, information flow analysis greatly reduces the number of sequences to be tested, but it cannot, in general, determine a sufficient set of test sequences.

Infeasibility It is often impossible to construct test cases that execute a specific path.

Determining input conditions for the execution of a specific path Even when a path is feasible, it may be very difficult to determine how to execute a specific path.

Hypotheses of path testing We shall assume the well known hypotheses underlying path testing: restriction to those errors that are related to a certain pattern of control flow and that are observable when those patterns are executed.

For clarity, our model does not account for the additional complexity introduced by other features of object-oriented programs, such as inheritance, polymorphism, recursion, instance creation, generic types, complex types, block scope, name scoping, cloning and idiosyncrasies of specific object-oriented languages. Information flow analysis is subject to the limitations of static analysis such as array indices that are known only at run-time, members of structured data type that are known only at run-time, aliasing, side effects, data flow anomalies and safe approximation.

The discussion is at the intuitive level. It is not rigorous and does not cover the details of dynamic information flow theory. The topic of this paper is the testing of intra-class flows through instance variables. It does not address inter-class testing nor the testing of interprocedural flows due to argument passing and return values.

Simple, highly-contrived Java programs are used to illustrate basic concepts. Although Java is the vehicle for describing our strategy in this paper, the strategy can be adapted for testing programs written in other object-oriented languages and even procedural languages. The terms “dynamic information flow testing” and “information flow testing” will be used interchangeably.

INTRA-CLASS TESTING - THE SEQUENCING OF METHODS

We distinguish three levels of Java testing:

α	intra-method
μ	intra-class
π	inter-class (applet, application, bean)

For later convenience, the Greek letters are used to designate structures associated with the various levels. The intra-method level (α) is presented mainly as background for generalizing and abstracting flow analysis for application at the level of object testing: the intra-class (μ) level. The inter-class level (π) will be the subject of a future paper.

The application of traditional structural test methods to the intra-method level is straightforward, since it resembles the testing of a single procedure. The testing of methods that send messages to each other within a class resembles conventional interprocedural testing. But when we cross the class boundary and begin to examine method interactions that are due to messages received from outside of the class, we enter a new region of testing: intra-class testing. It is in this region that existing structural test strategies appear to break down.

In intra-class testing, the input space consists of the method sequences for a class. Method activation sequences imply paths among class variables. Conventional test methods appear to provide little guidance in how to test these paths. At first glance, we seem to be back to exhaustive testing: i.e., trying all the sequences of method invocation.

PATH TESTING

terms

Program element is the code element being tested. It may be a complete program, such as a Java application, or a single procedure, such as a Java method.

Point in an execution thread is the state of program execution directly preceding or following the execution of a program statement.

Entry point is the state directly preceding execution of first statement of a program element. A program element has one entry point.

Exit point is the state directly following execution of a statement that exits a program element. A program element may have one or more exit points.

Basic block is a consecutive sequence of program statements with a single entry point, the first statement, and a single exit point, the last statement.

Segment is a basic block, a conditional statement or the decision outcome of a conditional statement.

Path is a consecutive sequence of segments.

Test case is a set of inputs which causes the execution of a specific path through the program element.

analogy to path testing

The *all-sequences* strategy is analogous to the classical *all-paths* strategy which requires a test case for each path in the program. The combinatorics inherent in non-trivial programs makes brute force strategies such as *all-sequences* and *all-paths* impractical. Over several decades, researchers have developed path testing strategies based on control flow analysis, data flow analysis and information flow analysis. Each test method represents a way of sampling the input space of a program with the goal of approximating the effectiveness of *all-paths*. This analogy suggests that path testing strategies, properly adapted, are applicable to intra-class testing.

TESTING SEQUENCES OF TWO METHODS

We begin by examining the simplest (non-trivial) sequence: two methods, which will be designated as A and B. If the computation in A can have no effect on the computation in B, then testing different sequences yields no new behavior.



FIGURE 1

In information flow testing, program elements such as A and B are said to be “independent.” The central principle of information flow testing is that, if A and B are independent, then the two program elements can be tested separately. Any sequences of A and B constitute unnecessary tests.

Is there a way that independence can be determined through an examination of the structure of the program elements? The application of data flow analysis was an important milestone [10] in the quest for a structural, intra-class testing strategy. In the case of testing two methods, independence can be ascertained through this form of analysis. Figure 2 illustrates how sequencing constraints are created by data flows through shared instance variables.

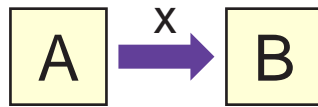


FIGURE 2

In progressing beyond two method sequences, the flow analysis must be capable of dealing with intra-method flows. In the following sections, we explain how data flow analysis is extended to form information flow analysis, which is capable of representing all intra-class flows. Essentially, information flow analysis is a more general means for establishing structural independence.

INFORMATION FLOW AT α LEVEL

Information flow analysis builds on its predecessors, control flow analysis and data flow analysis. Like data flow analysis, information flow analysis models a program as a set of definitions and uses of variables. Like control flow analysis, information flow analysis models a program as a set of (control flow) paths. Information flow analysis examines how information is transferred through program variables (memory locations). Dynamic information flow analysis was developed as the foundation for a new generation of structural testing strategies and associated coverage measures.

During a single execution thread from the entry point to an exit point of a non-trivial program element, only one out of a possible multitude of control flow paths is taken. The predicates in conditional statements select the single path by blocking all other possible paths. In a similar manner, only a subset of possible data flows is executed. This subset is selected by the ability of predicates to block all other possible data flows. The novel aspect of information flow analysis is the incorporation of special structures which model the ability of predicates to block flows.

composable information flow elements

Information Flow Analysis models a program as a collection of information flows. The basic building blocks of information flows are the information flow elements:

- memory access elements
- data flow elements
- control flow elements

These elements are composable, so any information flow may be represented by a sequence of information flow elements.

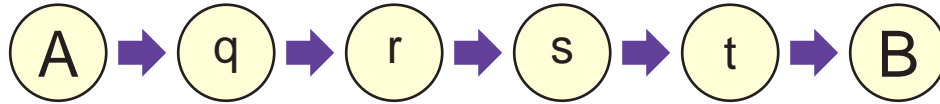


FIGURE 3

α -level flow analysis example

The simple Java method listed in Figure 4 will be used to illustrate flow analysis concepts.

```

public int add1(int a) { /*      segment #1      */
    int b=0;           /*      segment #1      */

    if(a > 0)         /*      segment #2      */
        b = 1 + a;   /*      segment #3      */
    return(b);       /*      segment #5      */
}
  
```

FIGURE 4

DATA FLOW TESTING

terms

Def-clear path A path from point A to point B is definition clear with respect to a variable, X, if X is not assigned a value along the path (except possibly at points A or B).

Reaches The value assigned to a variable at point A "reaches" the point B if the path between A and B is a def-clear path for that variable.

intuitive concept of data flow

Data flow test strategies are myriad, complex and strewn with subtle differences [8]. The purpose of this section is to introduce the flow elements that are used in information flow analysis. More general descriptions of data flow testing are contained in [1].

Let A and B be two points in an execution thread. Suppose X is assigned a value at point A and that value is used at point B. The intuition underlying data flow analysis is that there should be at least one test case which executes a path between the two points [13].

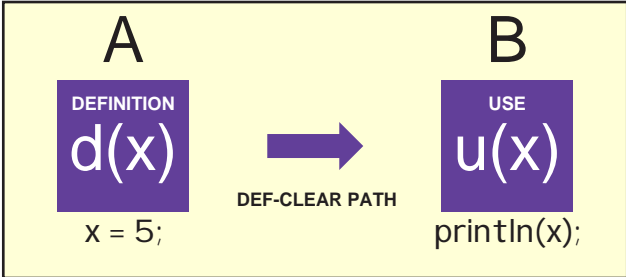


FIGURE 5

Some flow relationships between two points can be represented by a simple data flow. With reference to the listing in Figure 4, a simple data flow exists between the assignment of the value '0' to variable 'b' in segment# 1 and the use of this value of 'b' in segment# 5.

This relationship is known by different names in the data flow literature and has been defined in slightly different ways. Our construct (the du-pair) is essentially the same as the concept presented in one of the earliest papers on data flow testing [11].

elements of information flow analysis - α memory access

The memory access elements form the basis for both data flow and information flow analysis:

- **definition** point at which a value is bound to a variable.
example assignment of value '0' to 'b' in segment# 1.
symbol $d_1(b)$

- **use** point at which the value of a variable is accessed.
example access of the value of 'b' in segment# 5.
symbol $u_5(b)$

- **p-use** a special type of use; a use which is contained in the predicate of a conditional transfer statement.
example access of the value of 'a' in segment# 2.
symbol $pu_2(a)$

The symbolic representation of the a memory access element is subscripted by the segment number in which the element occurs. Formally, the information flow elements have a prefix indicating the level of program abstraction (α or μ). For notational

simplicity, the prefix is omitted for α -elements.

elements of information flow analysis - α data flow

The data flow elements are ordered pairs of memory access elements. The first memory access element is the 'input' and the second is the 'output'.

- **du-pair**

<i>input</i>	definition
<i>output</i>	use
<i>example</i>	the simple data flow above
<i>symbol</i>	$d_1(b) \longrightarrow u_5(b)$

For a du-pair to exist, there must be a def-clear path from the definition to the use. A du-pair represents information flow through a single variable.

- **ud-pair**

<i>input</i>	use
<i>output</i>	definition
<i>example</i>	the relationship between the use of the variable 'a' and definition of variable 'b' in segment# 3.
<i>symbol</i>	$u_3(a) \longrightarrow d_3(b)$

The use and definition of a ud-pair are contained in a single statement. The value of the variable in the use is employed in establishing the value of the definition. Since the variable in the use may be different from the variable in the definition, the ud-pair is one construct that is used to represent information flow between variables.

the need for a more general flow analysis

The data flow elements may be combined to represent more complex flow structures. For example, k-dr interactions [15] have been proposed to capture indirect flows, i.e. those which are propagated through a chain of du-pairs connected by ud-pairs.

But there exist flow relationships that cannot be represented by data flow relationships alone. In the example, there is no data flow between the variable 'a' in segment# 2 and the variable 'b' in segment# 5, yet the value of 'a' has a definite influence on the value of 'b'. Such flows involve the interaction of the control structure and the data flow structure of a program. In the compiler optimization [9] and dependency literature [12,16], these are called "control" dependencies.

An arbitrary flow relationship may be made up of both data flow and control flow relationships. Figure 6 schematically illustrates that data flow provides an incomplete model of flow.

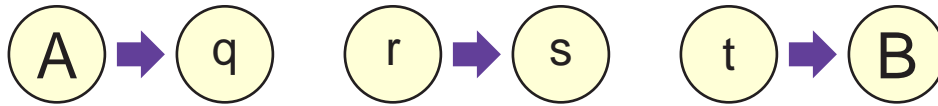


FIGURE 6

Information Flow Analysis adds a fundamental element, the control flow element, to fill these gaps.

INFORMATION FLOW TESTING

terms

α -*graph* is a graph in which the nodes are α memory access elements and the edges are α information flow elements.

α -*path* is a connected sequence of α information flow elements, in which the output of one element is the input of the next.

intuitive concept of information flow

The intuitive concept underlying information flow is a generalization of the concept underlying data flow. Recall Figure 5. In data flow analysis, the memory access at point A is a “write”. In information flow analysis, this memory access may be either a “write” or “read”. In data flow analysis, memory access at both points is to the same variable (X). In information flow analysis, the use at point B may access X or another variable, Y. If the value of X at point A can affect the value of the variable accessed at point B, then there is an information flow relationship between points A and B. The intuition underlying information flow analysis is that there should be at least one test case which executes each element of information flow between A and B.

elements of information flow analysis - α control flow

The control flow element is the unique flow element in information flow analysis. It is an ordered pair of memory access elements, but the input is always a p-use.

- **cd-pair**

<i>input</i>	p-use
<i>output</i>	memory access element
<i>example</i>	$pu_2(a)$ and the definition $d_3(b)$
<i>symbol</i>	$pu_2(a) \rightleftarrows d_3(b)$

The input use appears in a conditional expression which can directly block:

- execution of the output definition or output use, or
- a def-clear path for the output definition which flows through the input

- p-use, or
- execution of the input p-use of other cd-pairs, or
- a def-clear path for definitions in the transitive closure of another cd-pair

Graphically, a cd-pair is represented by an arrow from the input p-use to the output memory access element.

In the example, the input p-use is $pu_2(a)$ and the output definition is $d_3(b)$. The predicate use of 'a' controls the execution of $d_3(b)$ and thereby determines whether or not the definition of 'b' in segment# 3 will reach the use of 'b' in segment# 5. Like the ud-pair, the cd-pair can be used to represent information flow between different variables.

Note that cd-pairs come in pairs. Each p-use generates two cd-pairs, since a predicate has two outcomes. The cd-pair generated by the alternative outcome is called the "complement" or "complementary pair." When testing a path through a cd-pair, it is necessary to test both the cd-pair and its complementary pair.

The examples in this paper have simple predicates (only one p-use in a conditional expression). If a conditional expression contains multiple p-uses, then the logical effectiveness of each p-use must be taken into consideration during testing [3].

The cd-pair is a refinement of the uu-pair, which was introduced in earlier papers on dynamic information flow analysis [2]. The "cd" prefix refers to "Cd testing", which was invented by Edward Miller [14] and was the progenitor of dynamic information flow testing.

INFORMATION FLOW AT μ LEVEL

Information flow exists at several levels. We have already seen examples of α -level flow. The information flows of interest for intra-class testing are at the μ level. Information flows at the method level are caused by argument passing, return values and instance variables. Although the affect of argument passing and return values can be represented in terms of information flow analysis, the following discussion is restricted to flows through instance variables.

Information flow analysis is applied to intra-class testing by constructing suitable abstractions of the α elements which operate at the μ level.

elements of information flow analysis - μ memory access

- μ -**definition** definition of an instance variable.
example the definition of power in segment# 5 of the MorePower method in the μ example below.
symbol μ - $d_5(\text{power})$

- μ -use use of an instance variable.
example the use of in power in segment# 5 of MorePower.
symbol μ -u₅(power)
- μ -p-use a special type of μ -use; a μ -use which is contained in the
predicate of a conditional transfer statement.
example the p-use of safety in segment# 2 of MorePower.
symbol μ -pu₂(safety)

A μ memory access element is associated with the method in which it appears. As in α elements, each μ memory access element is subscripted by the segment number in which the element occurs.

The μ data flow and control flow elements are ordered pairs of μ memory access elements. The first μ memory access element is the 'input' and the second is the 'output'.

elements of information flow analysis - μ data flow

- μ -du-pair ordered pair of μ memory access elements
input μ -definition
output μ -use
example the definition of safety in segment# 3 of
Togglesafety and the use of safety in segment# 2
of the PowerUp.
symbol μ -d₃(safety) \longrightarrow μ -pu₂(safety)

The μ -du-pair occurs between methods.

elements of information flow analysis - μ control flow

- μ -ud-pair ordered pair of μ memory access elements.
input μ -use
output μ -definition
example the use of safety in segment# 2 of PowerUp and the
definition of power in segment# 3 of PowerUp.
symbol μ -u₂(safety) \longrightarrow μ -d₃(power)
- μ -cd-pair described below

The μ -ud-pair and the μ -cd-pair occur within a single instance of a method.

In a μ -ud-pair, an α -path from its input μ -use to its output μ -definition is called a " μ -ud-path". A μ -ud-pair must have at least one μ -ud-path.

A μ -cd-pair occurs when a μ -ud-path passes through an α -cd-pair. A μ -ud-path passes through an α -cd-pair if:

- the input μ -use is either the input p-use of an α -cd-pair or is connected to the input p-use of an α -cd-pair via an α -path, and
- the output μ -definition is either the output definition of the same α -cd-pair or is connected to the output definition of the same α -cd-pair via an α -path.

The complement of the μ -cd-pair must be tested if the complement is also a μ -cd-pair.

analogous structures

The μ -path is a generalization of a def-clear path for an α -du-pair. A def-clear path for an α -du-pair begins with an α -definition and ends with an α -use. Similarly, a μ -path begins with a μ -definition and ends in a μ -use. Whereas a def-clear path weaves through the data flow structure of a single method, a μ -path crosses method boundaries and weaves through the information flow structure of one or more methods. Just as an α -cd-pair can block an α -du-pair, a μ -cd-pair can block a μ -path.

μ -box

Once we introduce information flow elements that can span the boundaries of methods and classes, it becomes necessary to identify the scope within which information flows occur. The notational device for this purpose is the “ μ -box”.

All visible effects of a single method call are grouped together in a μ -box. The box encloses all the μ -elements associated with the method call. By convention, input uses appear on the left hand side of the box and output definitions appear on the right. Since any memory access element inside a μ -box is a μ -element, the μ prefix is dropped. Any link between a use and definition that appears within a μ -box is a μ -ud-pair.

μ -graph

A μ -graph is a flowgraph in which the nodes are μ memory access elements, and the edges are μ information flow elements. Normally, μ -elements are shown inside of μ -boxes.

μ -path

Essentially, a μ -path is a path in a μ -graph. A μ -path is a connected sequence of μ information flow elements, in which the output of one element is the input of the next. A μ test path begins with a μ -definition and ends with a μ -use. The μ -path represents an information flow at the μ level. A μ -subpath is a connected subsequence of μ -

elements in a μ -path. A static μ -path is called a “structural” path. Normally, the term “ μ -path” refers to a dynamic (run-time) path. There are often structural paths in programs that do not correspond to (dynamic) μ -paths.

A μ -path is composed of fundamental subpaths called elementary μ -paths. A graph of all μ -paths in a particular objects has special nodes, μ -nodes, where μ -paths begin, end, converge or diverge. Let A and B be two μ -nodes. An elementary μ -path is a maximal element of the set of all μ -paths that begin at A and end at B. (This means that an elementary μ -path is not a μ -subpath of any other μ -path from A to B.) Appendix A provides examples of elementary μ -paths. The elementary μ -path structure characterizes the information flow structure of an object, and therefore how it should be tested.

A complete μ -path is a maximal element of the set of all μ -paths for a particular object.

DYNAMIC INFORMATION FLOW TESTING

Dynamic information flow testing reflects the changing nature of flows at run-time. During execution, flows such as α -paths and μ -paths appear and disappear, depending on the transient states of predicates. Figure 7 illustrates how a predicate can cause a μ -use of ‘x’ to “disappear.”

```
y = 1;  
if ( y < 0 )  
    y = x + 2;
```

FIGURE 7

Figure 8 illustrates how a predicate can cause a μ -definition of ‘x’ to disappear.

```
y = 1;  
if ( y < 0 )  
    x = 2;
```

FIGURE 8

These changing μ -states cause μ -du-pairs to dynamically appear and disappear, as shown in Figure 9.

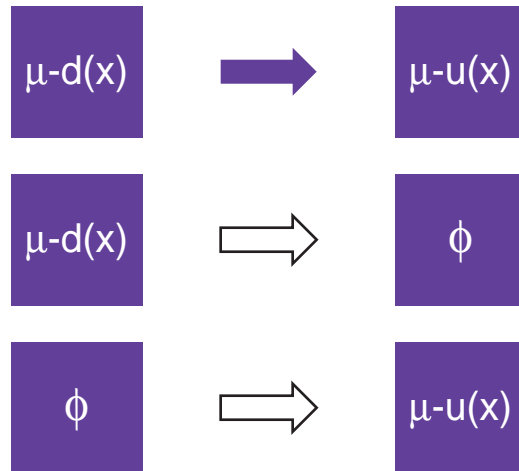


FIGURE 9

Effective testing requires a (run-time) μ -path between methods. A 2-sequence can be tested, but if the μ -path between methods is not executed, then the test is not effective. This simple observation is the basis of a fundamental result of this research.

testing sequences of three methods

In a similar manner, changing μ -states cause μ -ud-pairs to appear and disappear at run-time as illustrated in Figure 10.

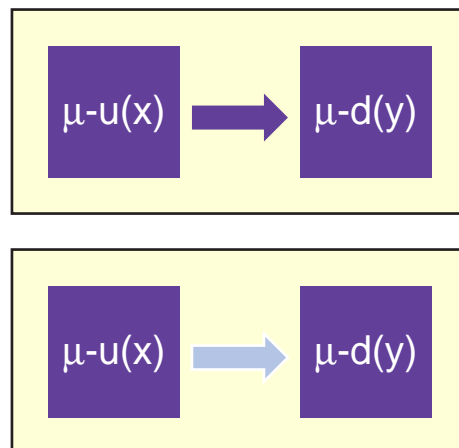


FIGURE 10

In the case of testing three or more methods, effective testing requires a μ -path from the initial μ -definition to the final μ -use. The μ -path traverses not only μ -du-pairs, but μ -ud-pairs within methods, as illustrated in Figure 11.

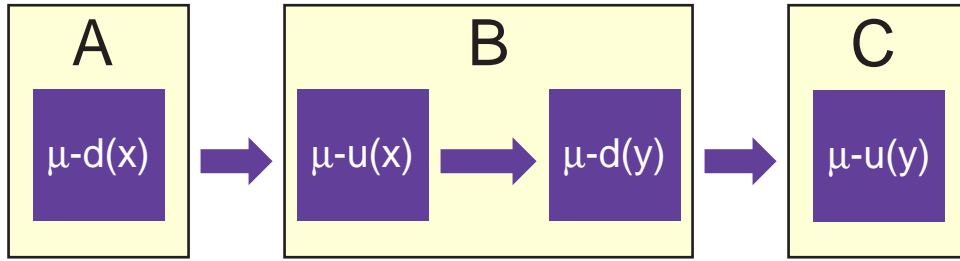


FIGURE 11

This figure illustrates that the testing of a μ -ud-pair requires the execution of at least three methods. The μ -path through the μ -ud-pair must contain a μ -subpath which begins at a μ -definition in the preceding method, passes through the μ -ud-pair, and ends in a μ -use in the succeeding method.

dynamic information flow intra-class test strategies

As in control flow and data flow testing, a family of test strategies and corresponding coverage measures of increasing effectiveness can be defined at the μ level. For instance:

COVERAGE MEASURE	STRATEGY
μ_0	<i>all μ memory access elements</i>
μ_1	<i>all μ data flow and μ control flow elements</i>
μ_{path}	<i>all μ-paths</i>
μ_{allseqs}	<i>all necessary sequences</i>

Note that the strategy *all μ -paths* is, in general, not equivalent to *all necessary sequences*. *All necessary sequences* is more extensive than *all μ -paths* since it includes combinations of elementary μ -paths (see example in Appendix A).

Although empirical studies will be necessary to determine the reliability level associated with each coverage measure, our initial estimate is that μ_1 should be a realistic goal for normal commercial applications. A higher level of assurance could be obtained with μ_{23} coverage, which approximates μ_{path} . A μ_{23} -cover consists of test cases which completely test all feasible 2-method and 3-method sequences. A 2-method sequence is completely tested if each μ -du-pair between the methods has been executed at least once. A 3-method sequence is completely tested by executing every μ -subpath that begins in the first method and ends in the third method.

PRINCIPLE #1 OF INFORMATION FLOW INTRA-CLASS TESTING

An effective test is a run-time μ -path which begins at a μ -definition and ends at a μ -use. Between methods, the path must pass through μ -du-pairs.

Within methods, the path must pass through μ -ud-pairs.

Principle #1 has a critical implication for intra-class testing which is discussed in the section on dynamic monitoring.

A single test case may cause the execution of several complete μ -paths. Each method in an effective test sequence is associated with at least one complete μ -path.

Principle #2 states that all μ -cd-pairs in a μ -ud-path should be tested.

PRINCIPLE #2 OF INFORMATION FLOW INTRA-CLASS TESTING

If a μ -ud-path passes through a cd-pair, then it is a μ -cd-pair. The μ -cd-pair and its complement (if it is a μ -cd-pair) should be tested (insofar as it is possible to do so through sequencing).

μ LEVEL FLOW ANALYSIS EXAMPLE

A simple Java class will be used to illustrate information flow intra-class testing:

```
/* PowerControl V1.0  
ROUGH SPECIFICATION
```

This class simulates a simple power control (electric lawnmower etc.) with 4 buttons:

PowerUp button turns power on.

PowerDown button turns power off.

Safety button toggles safety on (1) and off (0).

If safety is on, the PowerUp and MorePower buttons are deactivated. The initial condition is safety on.

MorePower

Each time the MorePower button is pressed, power level advances one level. There are four power levels: off(0), low(1), medium(2) and high(3).

```
*/
```



```

public class PowerControl {
    /* CLASS VARIABLES */
    int power;
    int safety;

    PowerControl(){
        power = 0;
        safety = 1;
    }

    public void PowerUp(){
        if(safety == 0)
            power = 1;
    }

    public void MorePower(){
        if(safety == 0) {
            if(power != 0) {
                power = power + 1;
                if(power > 3)
                    power = 3;
            }
        }
    }

    public void PowerDown(){
        power = 0;
    }

    public void ToggleSafety(){
        if(safety == 0)
            safety = 1;
        else
            safety = 0;
    }
}

```

FIGURE 12

notation for test cases

There are five methods, which will be abbreviated for later convenience:

PC	PowerControl (constructor)
PU	PowerUp
PD	PowerDown
TS	ToggleSafety
MP	MorePower

A test case for this class, which is a sequence of method calls, will be designated by listing the appropriate series of method calls (using the above abbreviations). For example, the test case, "PowerControl, ToggleSafety, PowerUp, MorePower, MorePower" would be abbreviated as:

PC TS PU MP MP

A brute force intra-class testing strategy would be to begin by executing all permutations of these methods. The number of permutations is 4 factorial or 24. But, this would not be sufficient, since exhaustive testing must also account for the possibility of cycles. As cycles of subsequences are added, the number of test cases quickly explodes, and it becomes apparent that exhaustive testing is impractical even when there are only a small number of methods.

The initial goal of this analysis is to obtain the μ -path structure of the object.

In the following graphs, the variable names power and safety will be abbreviated as 'p' and 's' respectively.

step 1 - construct μ/α graphs

First, the α -graphs of the individual methods (including the constructor) are converted into a hybrid form called the μ/α graph. This is accomplished by replacing each a memory access element with its μ counterpart. Output μ -definitions that are not contained in a method are replaced by the symbol ' ϕ ', which represents an "empty" definition. A μ -ud-path cannot terminate in a ' ϕ '.

The μ/α graphs are shown in Figure 13.

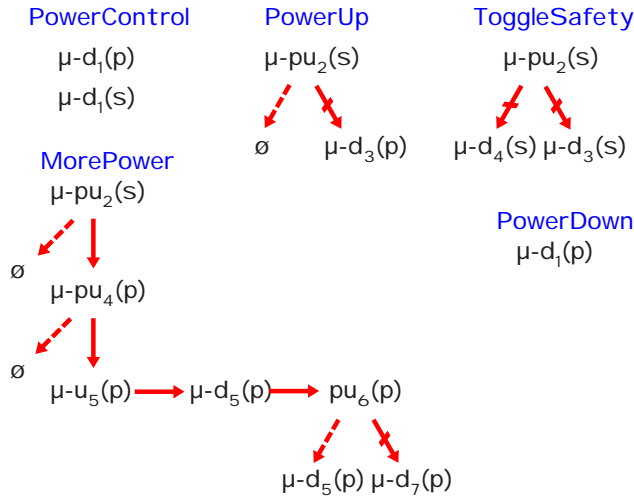


FIGURE 13

step 2 - convert μ/α graphs into μ -boxes

Next, we examine these methods from the viewpoint of a class sending messages to the PowerControl object. From this external viewpoint, the methods are summarized as μ -boxes:

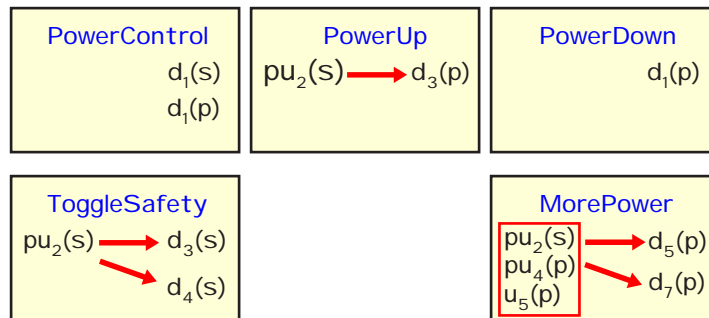


FIGURE 14

The flow diagram in MorePower represents six μ -ud-pairs. Only the two maximal paths are shown. The inputs are grouped, since if any μ -ud-path in MorePower is executed, then all input uses will be executed.

step 3 - derive the μ -paths

The structural μ -paths may be derived in a simple manner. For each μ -box, connect each output definition to all the matching uses in other μ -boxes. If there is a matching input in the same box, a loop around that box is created. In the PowerControl example, this approach results in the μ -graph shown in Figure 15.

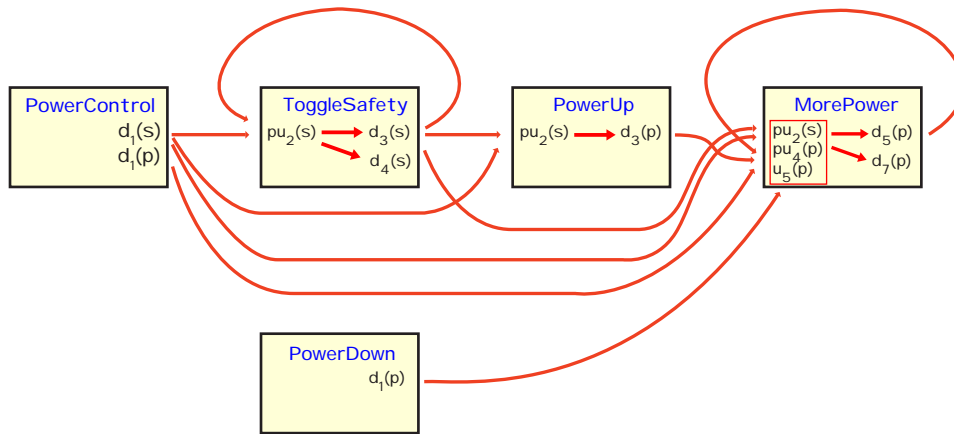


FIGURE 15

RESULT #1

Information flow analysis allows us to visualize and thereby gain an intuitive understanding of intra-class paths.

The structural μ -paths are obtained by tracing paths in the above graph. This model can serve as the basis for an *all μ -paths* test strategy. A model suitable for more closely approximating the *all necessary sequences* strategy is presented in Appendix A.

This μ -graph illustrates only paths inside the receiving class that are created by class variables. Method arguments (messages) and return values can create external paths which are in the sending class.

Static analysis allows the structural paths to be summarized by the regular expression:

$$PC ((TS^* (PU \mid \epsilon)) \mid PD) MP MP^*$$

To simplify the analysis, we begin by excluding the expression containing further repetitions of MP. This result is:

$$PC ((TS^* (PU \mid \epsilon)) \mid PD) MP$$

By limiting repetitions of TS to two, the initial test set is:

- PC PD MP
- PC MP
- PC PU MP
- PC TS MP
- PC TS PU MP
- PC TS TS MP
- PC TS TS PU MP

step 4 - constrain testing to run-time μ -paths

This step illustrates the dynamic nature of information flow, and one of the results of this research (Result #3).

The static analysis performed above provides a only a first approximation to a μ_{path} cover. The resulting path set contains control flow paths that do not correspond to μ -paths at run time. These extraneous paths are the consequence of imprecision in static analysis.

Consider the simple test case:

PC MP

The first pass through MorePower is illustrated in Figure 16.

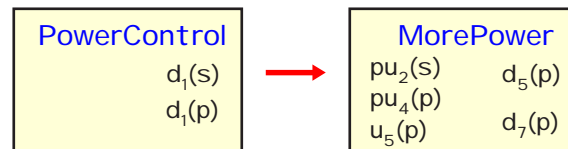


FIGURE 16

An examination reveals that no μ -path through this method can be executed. In fact, neither on the first pass nor on subsequent passes through MorePower, are any μ -ud-paths available. The loop created by the output definitions of power and input uses of power is not available since safety is zero.

This restriction to only one execution of MorePower applies to all test cases in which the incoming value of safety is '1'. In such test cases, all further repetitions of MorePower are unnecessary paths according to information flow theory.

Similarly, all test cases in which the input value of power is zero (in MorePower) also constitute unnecessary test cases. This quickly prunes the initial test set to only one test case that can serve as the beginning subsequence of test cases with repetition of MP:

PC TS PU MP

The initial test set must be modified, since some test cases do not correspond to μ -paths:

PC PU MP becomes PC PU, since there is no μ -path through PU.

PC TS TS PU MP becomes PC TS TS PU, since there is no μ -path through PU.

Note that PD is a special case, and does not require a μ -path through it, since it is a terminal method.

step 5 - test μ -cd-pairs

In this step, we apply Principle #2 of information flow intra-class testing.

The two μ -cd-pairs in the ToggleSafety method are executed by the subsequences:

```
PC TS PU
PC TS TS PU
```

which are covered by the initial test set.

MorePower is more complex and has four μ -cd-pairs. Principle #2 implies that all four μ -cd-pairs and their complements should be tested. The first two μ -cd-pairs do not have complements, since there are no μ -ud-paths through the alternate decision outcomes. The last two μ -cd-pairs are complements of each other. The two test cases shown (and the corresponding μ -ud-paths) are adequate for testing all four μ -cd-pairs.

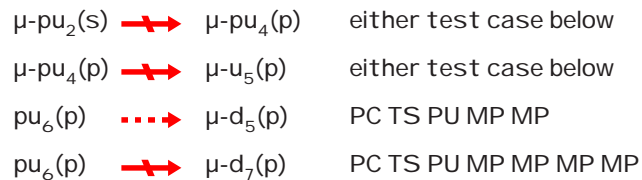


FIGURE 17

Adding these two test cases to the current test set yields:

```
PC PD MP
PC MP
PC PU
PC TS MP
PC TS PU MP
PC TS TS MP
PC TS TS PU
PC TS PU MP MP
PC TS PU MP MP MP MP
```

RESULT #2

Information flow analysis is capable of determining the fundamental sequences of methods to be tested.

It is instructive to compare this test set with a test set generated by a static data flow criterion (*all μ -du-pairs*). The simple data flow approach would miss the last five (necessary) test cases and include two unnecessary test cases (PC PU MP and PC MP MP).

limitations of information flow approach

Although information flow was able to significantly reduce the number of test cases in the above simple example, we must caution that the approach is only capable of identifying fundamental test sequences (or conversely, unnecessary test sequences). Due to the presence of loops and the limitations of the static analysis underlying information flow testing, it is not, in general, capable of identifying all necessary test cases.

implications for dynamic monitoring

The simple test case (PC MP) in the above example illustrates how the structural paths obtained by connecting the inputs and outputs of μ -boxes may, in some cases, be ineffective. But how do we know if a given test is effective (with respect to exercising μ -paths)? This could be determined through inspection, which was easy for a simple example like the one above, but, in general, predicting path executions can be technically very difficult. The only known general, practical solution is to monitor path execution to assess if the paths corresponding to a set of tests are effective (in the information flow sense).

RESULT #3 PRINCIPLE #3 OF INFORMATION FLOW INTRA-CLASS TESTING

To assure effective intra-class testing, path execution must be monitored at run-time.

Coverage analysis is not a new concept in software testing. Branch coverage analysis is becoming commonplace, and path coverage monitors have been developed by researchers.

CONCLUSION

This paper has described an initial foray into the challenging area of object-oriented testing utilizing path testing as a conceptual tool. This initial application of information flow analysis to intra-class testing demonstrates that:

- information flow elucidates the structure of intra-class paths.
- information flow testing can be used to determine fundamental sequences of methods to be tested.
- a special form of path coverage analysis is necessary to insure that test sequences are effective.

Although it has been known for many decades that path testing is a good option for achieving high test effectiveness, this research is the first to indicate an application in which path coverage analysis is a necessity for assuring the effectiveness of test results. Even when necessary sequences of methods are executed, the tests are not effective (with respect to the hypotheses of path testing) unless methods are in the proper dynamic states.

Path testing has potential to help unravel other challenging facets of object-oriented testing. Good testing practices require that a test plan incorporate both white-box (structural) and black-box testing techniques. In this paper, we have examined how information flow analysis can be applied to the structural testing of an object. We believe that a systematic means for black box testing at class scope can be obtained in a similar manner, through the application of information flow analysis to the state-model of a class.

REFERENCES

- [1] Beizer, B. *Software Testing Techniques* (second edition). New York: Van Nostrand Reinhold, 1990.
- [2] Bently, W.G. Moving toward data use testing. In Proceedings, 3rd annual Software Quality Week. San Francisco: Software Research, Inc., May 1990.
- [3] Bently, W.G. An introduction to Cd testing. In Proceedings, 4th annual Software Quality Week. San Francisco: Software Research, Inc., May 1991.
- [4] Bently, W. G. Software test efficiency and information flow analysis. In Proceedings, 6th annual Software Quality Week. San Francisco: Software Research, Inc., May 1993.
- [5] Binder, R.V. The FREE-flow graph: implementation-based testing of objects using state-determined flows. In Proceedings, 8th annual Software Quality Week. San Francisco: Software Research, Inc., May 1995.
- [6] Binder, R.V. State-based testing. *Object Magazine* 5(4):75-78, July-August 1995.
- [7] Binder, R.V. State-based testing: sneak paths and conditional transitions. *Object Magazine* 5(6): 87-89, October 1995.
- [8] Clarke, L.A., Podgurski, A., Richardson, D.J., and Zeil, S.J. An investigation of data flow path selection criteria. Workshop on Software Testing, Banff, Canada, July 1986, pp. 23-31.
- [9] Ferrante, J., Ottenstein, K.J., and Warren, J.D. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, Vol. 9(3), (July 1987), pp.

319-349.

[10] Harrold, M.J. and Rothermel G. Performing Data Flow Testing on Classes. Second ACM SIGSOFT Symposium on the Foundations of Software Engineering, December 1994.

[11] Herman, P.M. A data flow analysis approach to program testing. The Australian Computer Journal, Vol. 8, No. 3, November 1976, pp. 92-96.

[12] Jackson, D. and Rollins, E.J. A new model of program dependences for reverse engineering. Proc. of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering [SIGSOFT '94], December 1994, New Orleans, LA, pp. 2-10.

[13] Marx, D.I.S. and Frankl, P.G. The path-wise approach to data flow testing with pointer variables. Proc. of the 1996 International Symposium on Software Testing and Analysis [ISSTA], January 1996, San Diego, CA, pp. 135-146.

[14] Miller, E.F. and Howden, W.E., eds. *Tutorial: Software Testing and Validation Techniques*. IEEE Computer Society, 1981.

[15] Ntafos, S.C. On required element testing. IEEE Trans. Soft. Eng., Vol. SE-10, no. 6, November 1984, pp. 795-803.

[16] Podgurski, A. and Clarke, L.A. The implications of program dependences for software testing, debugging and maintenance. Proc. of the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification (TAV3), December 1989, Key West, FL, pp. 168-178.

APPENDIX A - DIRECTIONS FOR FURTHER RESEARCH

We are currently investigating how to “bridge the gap” between *all μ -paths* and *all necessary sequences* (insofar as this is possible with information flow testing). The gap consists of necessary sequences that are not exercised by the execution of a complete μ -path. In this appendix, we will use PowerControl to illustrate a possible solution.

The μ -graph in Figure 15 is capable of representing all μ -paths in PowerControl. There are necessary sequences that cannot be derived (directly) from this graph. These sequences are generated by converging information flows. The convergence of information flow within MorePower is portrayed in Figure 18, which shows one way in which inputs involving the two variables, ‘s’ and ‘p’, combine to produce a single output of ‘p’.

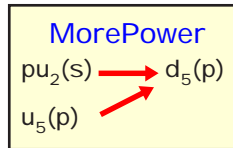


FIGURE 18

The sequences are represented by the simultaneous execution of elementary μ -paths. To more closely approximate *all necessary sequences*, it is necessary to test combinations of converging elementary μ -paths.

To construct a graph which represents these combinations, we first construct separate μ -graphs for the two sets of elementary μ -paths that converge within *MorePower*. The μ -graph of all elementary μ -paths that end at a μ -use of 'p' in *MorePower* are shown in Figure 19. Similarly, the μ -graph of all elementary μ -paths ending at a μ -use of 's' in *MorePower* are shown in Figure 20.

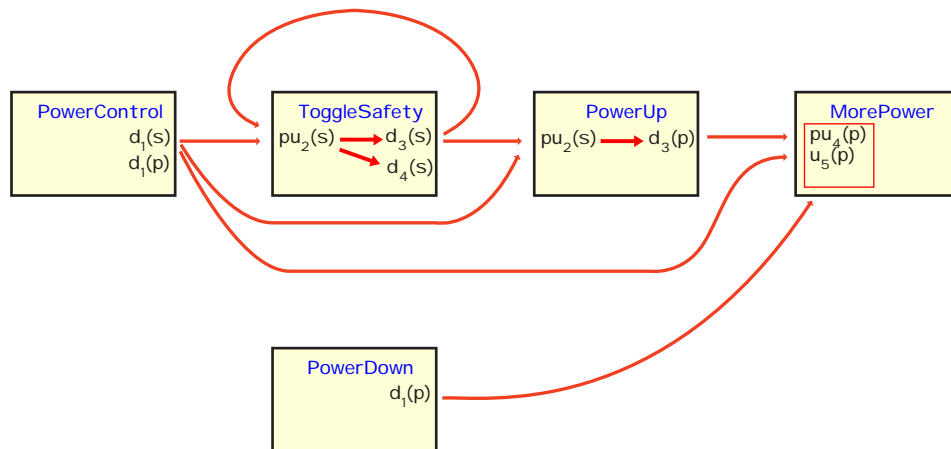


FIGURE 19

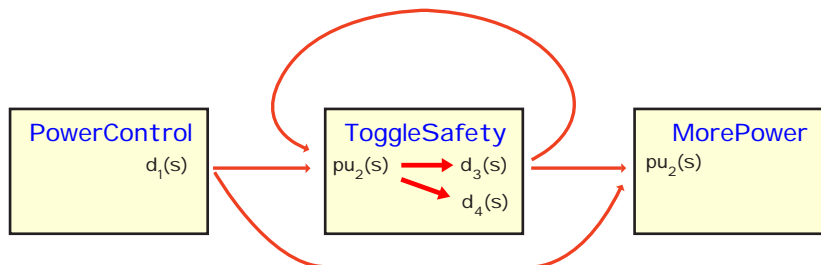


FIGURE 20

Each figure represents a set of elementary μ -paths. The “product” of the two sets of elementary μ -paths is obtained by taking each path from one set and interleaving it with each path from the other set. The “interleave” product will normally not be unique, and it is possible for the product of two paths to be two or more paths.

Incorporating the feedback path around MorePower leads to the resulting product, shown as a μ -graph in Figure 21. The representation is not unique, due to commutativity of some path products.

In this case, the effect of the path product has been accounted for by adding a simple construct, the μ^* -du-pair. We do not yet know if this is possible for all path products. If it is, then simple approximations, such as μ^*_1 (all μ -elements including all μ^* -du-pairs) become available for testing path products. The coverage measure μ^*_{23} , which would completely test all 2-sequences and 3-sequences, including those composed of μ^* -du-pairs, would be a close approximation to *all-sequences*.

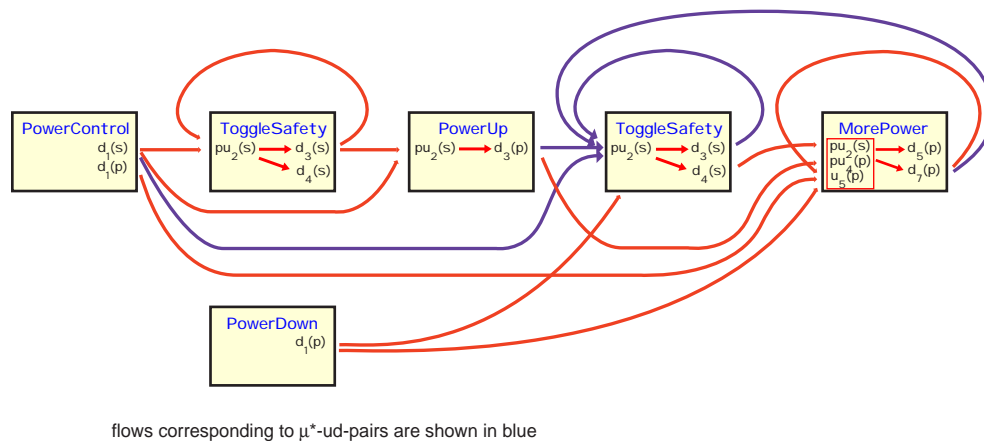


FIGURE 21

The regular expression is:

$$PC ((TS^* PU) | \epsilon | PD) TS^* MP (TS^* MP)^*$$

The initial path set is:

- PC PD MP
- PC MP
- PC PU MP
- PC TS PU MP
- PC PD TS MP
- PC TS MP
- PC PU TS MP
- PC TS PU TS MP
- PC PD TS TS MP
- PC TS TS MP
- PD PU TS TS MP
- PC TS PU TS TS MP
- PC TS TS PU MP

PC TS TS PU TS MP
PC TS TS PU TS TS MP

After removal of non-runtime μ -paths and addition of feedback paths around MP, the final test set is:

PC PD MP
PC MP
PC PU
PC TS PU MP
PC PD TS MP
PC TS MP
PC TS PU TS MP
PC PD TS TS MP
PC TS TS MP
PC TS PU TS TS MP
PC TS PU MP TS MP
PC TS PU MP MP
PC TS PU MP MP MP MP

Rev. L

COMPONENT INDEPENDENCE FOR SOFTWARE SYSTEM RELIABILITY

Denise Woit & Dave Mason

Department of Maths, Physics and Computer Science
Ryerson Polytechnic University
`dwoit@scs.ryerson.ca`
`dmason@arg.ryerson.ca`

1998 Nov 11

Component Reliability

- ⇒ • Introduction
- System Reliability
 - Estimation from Component Reliability
 - Hardware Reliability Models
 - Software Models
 - Issue of Independence
 - Rules for Software Independence
 - Experimental Results
 - Conclusion

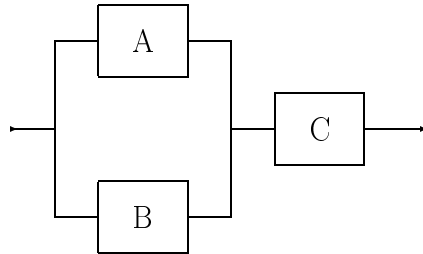
System Reliability

- probability of successful operation per usage, or time period in a given environment
- statistical measure/models
- test results are the *data* for a model
- requires operational profile, e.g., expected usage/input distribution
- typically at system level

Estimation from Component Reliability

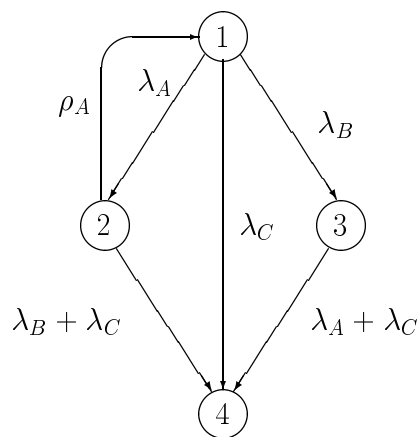
- common in other engineering fields:
 - statistical models to combine component reliabilities
 - more cost/time effective (system testing, component reuse)
- interest for software:
 - supports module reuse, COTS components
 - more cost/time effective
 - supports treatment of software development as engineering discipline

Simple system



- A three component system
- For working system: component C and one of A or B must be working

Markov Model



Markov model for the simple system from previous slide

- Generally:

$$R(t) = P[\text{in a success state at time } t]$$

$$= 1 - P[\text{in failure state at time } t]$$
- Here, because failures are independent:

$$R(t) = P_1(t) + P_2(t) + P_3(t)$$

$$= 1 - P_4(t)$$
- Standard Markov analysis to calculate $P_i(t)$
- Markov assumptions:
 - state transitions independent: λ_A constant, regardless of how system arrived in state 1
 - failure rates independent: $P[A \text{ fail} \mid B \text{ fail}] = P[A \text{ fail}]$

Various Markov Software Models

- nodes: components;
- sequential execution;
- $arc_{i,j} : P[\text{transition from component } i \text{ to } j]$
- $R_{Sys}(t) = P[\text{in a success state at time } t]$

$$= 1 - P[\text{in F at time } t]$$
- $R_{Sys} = P[\text{transition from "start" to "success"}]$

$$= 1 - P[\text{transition from "start" to F}]$$
- uses typical Markov analysis as above

- Same Markov assumptions:
 - state transitions independent
 - component/state failures independent
- Software system problems:
 1. component independence: if A calls B λ_A *depends on* λ_B
 2. component independence: B might change A's *state*, affecting λ_A
 3. Markov diagrams assumed similar to system flow diagrams (Parnas invoke v.s. use)
- Solutions:
 - (1,3) impose rules governing “structure” of system, components, model. Employ *invoke* rather than *use*
 - (2) later...

Simple Program - Module A

```

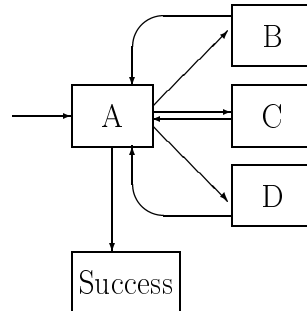
void A() {
    int x=0;
    do {
        if (x%2==0)
            B(x);
        else
            C(x);
        x=x+1;
    } while (x<10);

    y=D(x);
    printf("y=%d\n",y);
}

(define (A)
  (define x 0)
  (define (loop)
    (if (even? x)
        (B x)
        (C x))
    (set! x (+ x 1))
    (if (< x 10)
        (loop)))
  (loop)
  (set! y (D x))
  (format #t "y=~a~%" y))

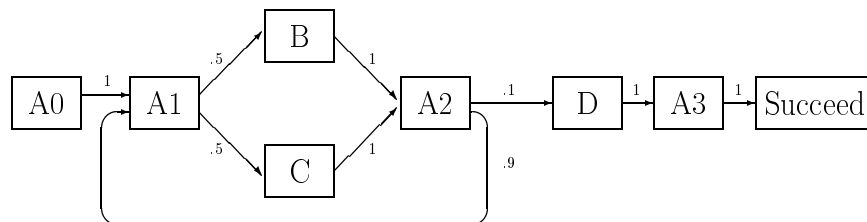
```

Simplistic Flow Diagram for Module A

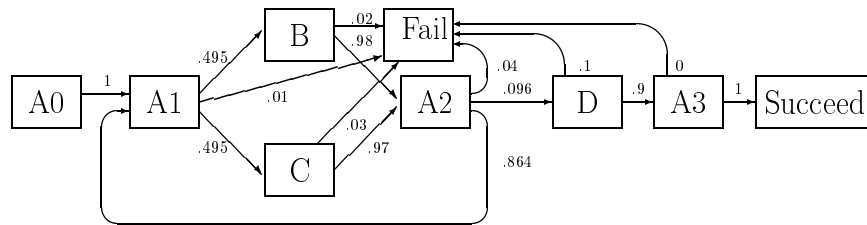


- Problem: *sequence* of control lost;
same R attributed to different fragments of A
- Solution: divide A into fragments to model system components

Correct Flow Diagram for Module A



Reliability Model for Module A



CPS Version - Module A

```
(define (A)
  (define (a1)
    (if (even? x)
        (B a2 x)
        (C a2 x)))
  (define (a2 result)
    (set! x (+ x 1))
    (if (< x 10)
        (a1)
        (D a3 x)))
  (define (a3 result)
    (set! y result)
    (format #t "y=~a~%" y))
  (define x 0)
  (a1))
```

Issues of Component Independence

Markov model requires independence of component reliabilities

- if module A invokes a broken module B, and thereby produces a wrong result, that is **not** a failure of A
- if a broken module A invokes module B with erroneous parameters and B thereby produces a wrong result, that is **not** a failure of B
- if module A and correct module B are incorrectly connected such that B fails, that is **not** a dependence between the modules, it is a design failure
- if module A invokes module B and it changes some state that module A depends upon, then A and B **are** dependent (Problem 2)

Program State

- state includes:
 - files
 - I/O registers
 - cursor position on screen
 - main memory
- atomicity is a desirable characteristic
- we discount the following from state:
 - stack/heap/disk space
 - CPU time
 - require separate proof that usage bounded

Rules for Software Independence

- no external state mutation/call-by-reference
- all accesses/updates to a variable are grouped as a sequence of critical sections
- restrict each critical section to a single fragment

How?

- pure functional programming (Erlang – Ericson, SML/NJ – Lucent), **or**
- use *no* global variables and ensure *all* updates to any variable are in a single fragment

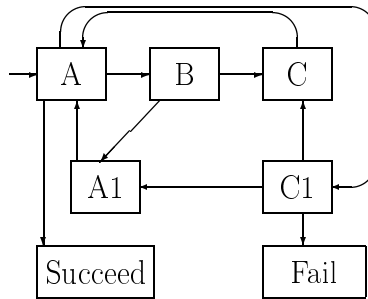
Experimental Results

- version of **grep** written in the functional subset of Scheme
- arbitrarily defined module boundaries (5 modules)
- broke it into fragments (6 new continuations)
- calculated reliability values

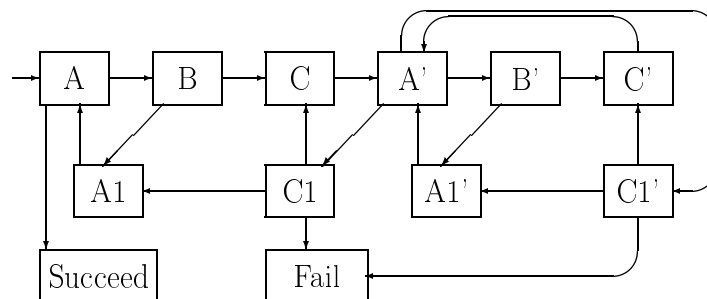
Original Data

Bug Name	Component Reliability	11 fragments
<code>compare_range</code>	79.23	7.39

- system reliability with this bug was about 75%
- recursive call from C to A creates an apparent path from A to C1 (the fragment containing the error)



Unrolling



- by unrolling, frequency of the phantom path is reduced and true system reliability is better approximated
- diagram above is unrolled once, producing the 35 fragment version in the table on the next slide
- problem only arises with mutually recursive **modules**

Unrolling

Bug Name	Component Reliability	35 fragments		131 fragments
		11 fragments	separate	
<code>compare_range</code>	79.23	7.39	63.65	63.64

- unrolling an additional time produced some improvement
- the operational profile seen by the module has little effect on the calculated reliability
- the calculated reliability is **always conservative**, so safety is preserved, even if the desired reliability cannot be attained.
- unrolling can be done at calculation time and does not have to be done to the program itself
- a better approach, “conditional paths”, was described in Quality Week Europe'97

Conclusion

- it is possible to build software modules that are independent, and therefore usable in Markov models
- presently restricted to purely functional module fragments

Future Work

- getting data on more programs
- extend to handle critical fragment groups (to support mutation)

Component Independence for Software System Reliability

Denise M. Woit

dwoit@scs.ryerson.ca

David V. Mason

dmason@scs.ryerson.ca

School of Computer Science
Ryerson Polytechnic University
350 Victoria Street
Toronto, Ontario
Canada M5B 2K3

Abstract

For a typical software system, it is generally considered infeasible to calculate system reliability from the reliabilities of its constituent components because software systems, unlike hardware systems, tend to violate the underlying independence assumptions inherent in the usual reliability calculations. We present a set of component design and interaction rules which, if followed in software development, can produce systems with the highly independent components necessary in order to legitimately calculate system reliability from component reliability. We present a system which follows our rules, and show that in this case system reliability calculated from component reliabilities was very close to the true system reliability.

1 Introduction

Software system reliability estimates are typically based upon data collected while testing the system as a whole [7, 10]. However, there is growing interest in estimating system reliability from the reliabilities of its constituent components. This technique is both pragmatically appealing, and supportive of the treatment of software development as an engineering discipline.

In the hardware realm, Markov-based models are commonly used to calculate system reliability from component reliabilities; this approach is preferred because of its cost-effectiveness. Because the underlying mathematical models for such calculations assume

component independence, hardware components are designed to be as independent as possible; any remaining dependencies are factored into the models [6, 7].

Unfortunately, the hardware models of reliability composition are considered inapplicable in the software realm because software components tend to violate the component independence assumption of the basic model. It is widely considered impossible or intractable to design software components to meet this requirement [7].

We have constructed design rules that allow the development of software components with the necessary independence, and with interaction properties that parallel those of physical systems, so that they are amenable to analysis with Markov models. The use of functional programming languages facilitates the construction of these highly independent components. We show that application of our rules can result in systems which do not violate the underlying assumptions of the typical reliability composition models.

We also discuss the limitations of first-order Markov models of software systems, as outlined in [12]. We explore techniques to mitigate these limitations. Conditional statements are specified and then used to automatically transform basic models into those that more accurately describe software component interaction. We present an example for which our transformations produce reliability estimates that are far more accurate than are possible with the traditional model.

The design rules and the model transformation tools combine to allow Markov models to be usable in de-

giving reasonable estimates of system reliability from the reliabilities of system components.

2 Hardware Models

For hardware systems, or systems that are some combination of hardware and software, estimations of overall system reliability from constituent component reliabilities are obtained via Markov or semi-Markov models [6, 5, 4, 8, 3]. In Markov models, system behavior is modeled by a set of system *states*, $\{S_1, S_2, \dots, S_n\}$, and transition rates/probabilities among states, $T_{i,j}, i, j, = 1, 2, \dots, n$. The Markov model assumes $T_{i,j}$ depends *only* upon S_i . This is known as the *Markov property*.

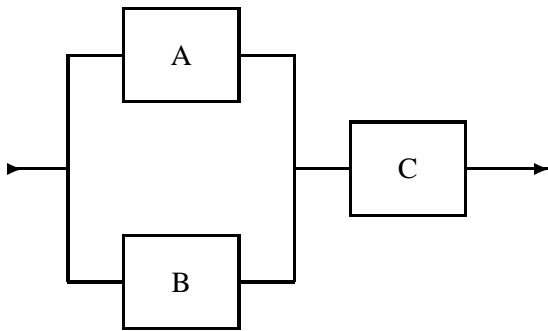


Figure 1. A simple 3 component system

In hardware systems, an S_i is usually considered some distinct combination of working and failed components. For each S_i , each $T_{i,j}$ is composed of the failure rates or repair rates of its components. States are partitioned into those representing system failure, and those not. Reliability is calculated as the probability of the system residing in a non-failure state [6]. Consider the simple three component hardware system of figure 1. Suppose that for this system to be functioning, it must be the case that component C and one of components A or B must be functioning. A Markov model for this simple system is presented in figure 2. In this model, states 1, 2 and 3 are *success states* (system can function) and state 4 is a *failure state* (system does not function). State 1 corresponds to all components, A, B and C, functioning. State 2 corresponds to B and C functioning, but A not. State 3 corresponds to A and

C functioning, but B not. State 4 corresponds to C not functioning with all combinations of A and B functioning/not. λ_i is the failure rate for component i . ρ_i is the repair rate for component i . Generally: $R(t) = P[\text{in a success state at time } t] = 1 - P[\text{in failure state at time } t]$. Here, $R(t) = P_1(t) + P_2(t) + P_3(t) = 1 - P_4(t)$, and standard Markov analysis is used to calculate $P_i(t)$. Assumptions of the Markov model are:

- state transitions are independent: λ_A constant, regardless of how system arrived in state 1
- failure rates are independent: $P[X \text{ fail} \mid Y \text{ fail}] = P[X \text{ fail}]$

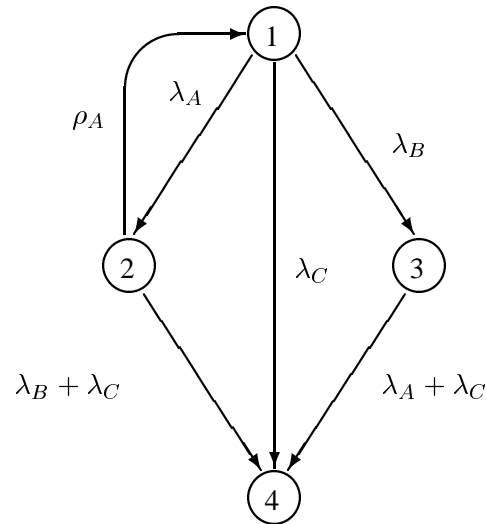


Figure 2. A Markov model for figure 1

3 Software Models

A similar approach has been presented for systems comprised entirely or partly of software components [2, 3, 5]. A state, S_i is a set of components under execution. $T_{i,j}$ is the probability of execution transition from S_i to S_j . For systems with sequential execution properties, an S_i contains only one component; for systems with non-sequential execution properties, an S_i contains more than one component. In the sequel, we consider sequential systems, without loss of generality; thus, $S_i \equiv C_i$, where C_i is the i^{th} component of the system, and $T_{i,j}$ is the probability of

execution transition from C_i to C_j . The ordering of the components is not relevant except that C_1 must be distinguished as the *start* component—that which is initially executed upon system start-up. A “termination”, or “system success” state, S , is included to represent successful termination of the software.

For systems comprised entirely or partly of software components Markov models have been used to calculate measures such as steady-state system availability and system reliability from the reliabilities of constituent hardware/software components [5, 3, 2]. First, the model is modified by including a “fail” state, F . Arcs are included from each component of the system (except the termination component) to F . An arc from C_i to F represents a failure of component C_i . If component C_i has reliability r_i , then the probability of failure is $T_{i,F} = 1 - r_i$. The probabilities on the remaining arcs emanating from C_i are each modified by multiplying them by r_i , as each of their probabilities is now reduced because of the addition of the new arc to F . Thus, the arc from C_i to F corresponds to the probability that C_i will fail; the remaining arcs from C_i correspond to the probabilities that C_i will successfully pass control on to another component of the system. When a graphical version of system component interaction is thus modified, the only absorbing states in the graph are the termination state and F . Therefore, any path through the graph beginning with the start state, and ending with the termination state, represents a failure-free execution of the system. Any path through the graph starting with the start state, and ending with the failure state, F , represents a failed system execution.

When a continuous measure is desired, the system reliability is $P[\text{in a non-failure state at time } t] = 1 - P[\text{in } F \text{ at time } t]$. For a discrete measure, system reliability is $P[\text{absorption at } S] = 1 - P[\text{absorption at } F]$. These measures can be calculated by solving a system of linear equations.

When Markov and semi-Markov models are utilized for systems with software components, the resulting reliability estimations are not meaningful unless the Markov model is a good representation of the actual system. Because they relate to establishing the Markov property, the following two factors significantly influence the adequacy of the representation: (1) the nature of transition properties within the given

system; (2) component independence in the given system. We believe that typical systems involving software components have properties such that modeling these systems in a Markovian fashion is not feasible. Because the underlying Markov property is violated, the resulting system reliability estimations are not meaningful. In the sequel, we will describe how a system can be designed or modified so that the transition properties *are* amenable to Markovian analysis. We will also outline our rules which can be used to create system components with the independence properties required for Markovian analysis.

4 Transition Properties and CPS

In this section we identify inconsistencies between software systems and the Markov models typically used to represent them. We outline how these problems can be overcome by using Continuation Passing Style (CPS)[1].

Parnas [11] differentiates between the mutually exclusive relations “uses” and “invokes”. $USES(C_i, C_j) = \text{iff } C_i \text{ calls } C_j \text{ and } C_i \text{ will be considered incorrect if } C_j \text{ does not function properly. } INV(C_i, C_j) = \text{iff } C_i \text{ passes control to } C_j \text{ but does not use } C_j$. Thus, if $USES(X, Y)$, then the reliability of component X will *incorporate* the reliability of Y . Their reliabilities (and thus failure rates, as reliability = 1 – failure-rate) will be dependent, and we write $RelUSES(X, Y)$. An assumption of the Markov model is that $T_{i,j} \equiv INV(C_i, C_j)$.

A software component often uses the results of other components to transform its input to output. Consider component A , which is presented in programming languages C and $Scheme$ in Figure 3. A typical Markov model for A is given in Figure 4.

A problem is apparent when comparing the model of Figure 4 to the system in Figure 3. Figure 4 is modeling component interactions for which $T_{i,j} \equiv INV(C_i, C_j)$ holds. However, in the actual component interactions of the system given in Figure 3, it is *not* true that $T_{i,j} \equiv INV(C_i, C_j)$. Figure 4 indicates $INV(A, B)$, but the system of Figure 3, indicates $USES(A, B)$: an inconsistency between system and model. Thus, the typical Markov model is not an accurate description of the actual system.

There is a further problem apparent when compar-

```

void A() {
1 int x=0;
2 do {
3   if (x%2==0)
4     B(x);
5   else
6     C(x);
7   x=x+1;
8 } while (x<10);
9
10
11 y=D(x);
12 printf(
13  "y=%d\n",y);
}

```

```

(define (A)
  (define x 0)
  (define (loop)
    (if (even? x)
        (B x)
        (C x)))
    (set! x (+ x 1))
    (if (< x 10)
        (loop)))
  (loop)
  (set! y (D x))
  (format #t
    "y=~a~%" y))

```

Figure 3. Component A

ing the model of Figure 4 to the system in Figure 3. RelUSES(A,B), RelUSES(A,C), and RelUSES(A,D) hold by Figure 3. Thus, the reliability of A will already incorporate the reliabilities of components B, C, and D. In fact, for any system thus modeled, the reliability of the *start* component, C_1 , will already incorporate the reliabilities of components C_i , where C_i is the transitive closure of USES on C_1 . Thus, the reliability for C_1 is the overall system reliability, making moot the entire exercise of calculating system reliability from component reliabilities.

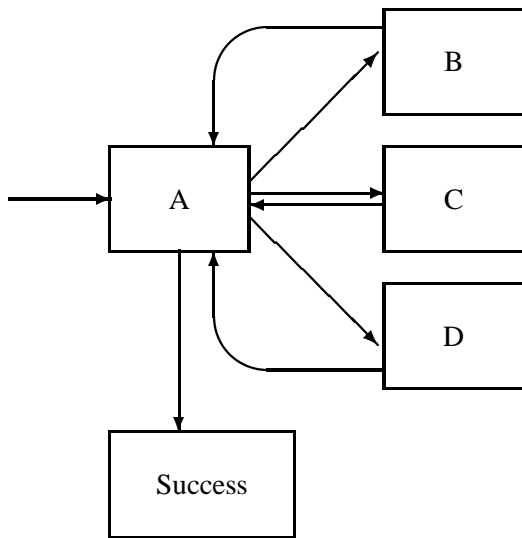


Figure 4. Simplistic model of A

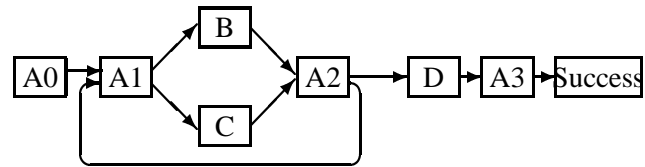


Figure 5. Markov model after CPS conversion

The problems described above are solved if the flow of information in the software system is via Continuation Passing Style (CPS), described below. It is important to note that a system can be *transformed* into CPS; it need not be initially *designed* using CPS principles. When a system is in CPS form, all of the component relations will be INV; none will be USES. Each component will perform an atomic transformation from input to output. The component, A, of Figure 3, can be transformed into CPS by dividing it into *fragments*, A0, A1, A2, and A3 as follows.

- A0:** set $x=0$ and invoke A1 (line 1)
- A1:** depending on result of if statement, invoke B or C (lines 3-6)
- A2:** increment x ; depending on result of decision, invoke A1 or D (lines 2,7-10)
- A3:** set y and print (lines 11-13)

Components B and C must be modified to invoke their continuation, A2. Component D must be modified to invoke its continuation, A3. The components of the CPS-converted system are thus A0, A1, A2, A3, B, C, D, and the corresponding Markov model is given in Figure 5. Note that the two problems above are now solved. Each component performs an atomic transformation of input to output, and uses the results of no other component in its transformation. Thus, INV holds for all component interactions. The reliability of any component is not dependent on that of other components because no USES relation exists among components and because the components use no global variables.¹

¹The relationship between reliability and program state is presented in Section 5.

4.1 CPS Transformation

When the system is CPS converted (or designed) all of the components are related by INV, none of the components are related by USES, and each component performs an atomic transformation of its input to output. The Markov model is thus applicable to software components in terms of component transformations.

To CPS convert a procedure/function, one partitions the component into *fragments*, which are sequences of instructions that do not involve a call to a component. Each fragment may be considered to be a function. Fragments become components in the CPS converted system. The new set of components *invoke* other components, passing along relevant program state and the next component to be executed. For example, consider component C in Figure 6. The section prior to the call to F becomes a fragment, C1. The section following the call to F becomes a fragment (function), C2. A must execute in the state, S1, that is visible in C before the call to F. C2 must execute in the state, S2, that is visible in C after the call to F. The components of CPS conversion are C1, F and C2. C1 passes F both C2 and S1. A will perform its function in state S1, and it in turn will pass control and S2 (including the result of F) to C2. C2 will perform its function in state S2.

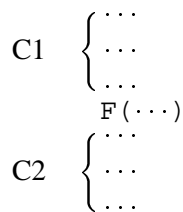


Figure 6. Component C

We have found CPS conversion straightforward when using functional programming languages because of the high-level data-structure facilities. In a language such as C, it is slightly more complicated, but can be facilitated using macros. CPS conversion of component A from Figure 3 using the programming language Scheme is given in Figure 7.

```

(define (A)
  (define (a1)
    (if (even? x)
        (B a2 x)
        (C a2 x)))
  (define (a2 result)
    (set! x (+ x 1))
    (if (< x 10)
        (a1)
        (D a3 x)))
  (define (a3 result)
    (set! y result)
    (format #t "y=~a~%" y))
  (define x 0)
  (a1))

```

Figure 7. CPS conversion of component A from Figure 3

5 Component Independence

When components are developed as, or converted to, CPS, the transition properties among components are consistent with those required for Markov modeling. CPS compliance, however, is not a sufficient condition for the Markov model to apply—the components of the system must also fail independently. When only the INV relation exists among components, it is impossible for component A to fail because of a failure of a component B which it calls. Thus CPS mitigates this issue of dependence. It is important to note that if A calls B with erroneous parameters, causing B to produce an incorrect result, this is *not* an indication that A and B are dependent. If A and B are incorrectly connected such that B fails, that is *not* a dependence between A and B, it is a system design failure. A and B *are* dependent, however, if A calls B, and B modifies some state that A depends upon. The state of a system includes global variables, mutable data-structures, and I/O state such as the position of file read/write pointers, values in device registers, segments of files that can be read/written, etc. State also includes system parameters such as the amount of free memory or the time of day. If these are relevant to the given system,

it must be proven by the system designer that they do not compromise the independence of the components.

Rules that will help establish component independence are as follows:

1. Design the system in (or convert the system into) CPS.
2. Code in a programming that constrains pointers and automates memory management, such as Java, ML, Scheme or Ada.
3. Use the functional-programming paradigm at the component level. Within a component, the functional paradigm is not required, but no mutable data-structures can exist *between* components.
4. Updates to I/O state must be within a single component. Further, the component cannot make any *assumptions* about the current state. Any knowledge about the existing state must be established by the component intending to modify it.

Systems which conform to our design rules above will not violate the Markov properties. Thus, using Markovian analysis to derive system reliability estimates from component reliabilities will be legitimate. We note, however, that many systems cannot conform to our rules because they require maintenance of some I/O state. For such systems, state must be incorporated into the reliability calculations, as outlined in [13].

6 Experimental Results

We developed a version of the Unix utility `grep` which conforms to all of our rules for independence. It is CPS-compliant and uses the functional-programming paradigm between components, but not within single components. The code was originally written functionally, but not CPS-compliant, using five components. It was subsequently CPS-converted. When divided into fragments, six new continuations were created, giving a total of eleven fragments. In order to calculate system reliability, we seeded errors into four fragments, and calculated the ensuing reliability of each fragment. System reliability was calculated with standard Markov methods. It was compared to the “true” reliability, which was obtained by testing

the system as a whole. The operational profile experienced by any given fragment during system testing was identical to that it experienced during individual fragment testing. This was important in order to be able to compare true vs. calculated system reliabilities in a meaningful way.

Initial results showed that calculated system reliability was much lower than the true system reliability. The discrepancy was traced to spurious paths through our Markov model of fragment interaction. Although the Markov property held for our system model, several fragment paths were modeled that were impossible, or improbable, in the actual system. This is a shortcoming of the Markov model, and was first discussed in [12]. The solution to this problem requires state-splitting, as outlined in [12]. One round of state-splitting created 35 fragments. With the new model, calculated system reliability was very close to true system reliability (a difference of about 5%.) We obtained moderate improvement when the state was split into 135 fragments (a difference of about 1%). It is important to note that state-splitting need not be performed manually. With the tools described in [12], conditional statements can be defined abstractly describing component interaction. The tool can automatically transform these conditional statements into the corresponding first-order Markov model, as required. There is little correlation between the final number of states in the calculated Markov model and the number of conditions required to specify them. It is possible to accurately describe a system of several hundred states with only a handful of conditional statements.

Besides using state-splitting to obtain a more accurate Markov model, we also experimented with recalculating reliabilities of fragments based on their now different operational profiles. Surprisingly, we found only negligible improvements in our calculated reliability estimates. This is empirical evidence supporting previous theoretical work which showed that alterations in the operational profile result in much smaller relative changes in the reliability estimates [9].

7 Conclusions

We outlined ways in which typical software systems, unlike hardware or combination systems, violate the underlying assumptions of Markov models. We

described why such models are inappropriate for calculating system reliability from reliabilities of system components. We then outlined rules which, if followed in software development, could produce system components which are independent, and systems which thus are amenable to Markovian analysis. We created a system which corresponded to our rules, and applied the typical Markov analysis to determine system reliability from component reliabilities. Our experimental results were promising, with calculated system reliability being close to true system reliability.

References

- [1] APPEL, A. W. *Compiling with Continuations*. Cambridge University Press, 1992.
- [2] CHEUNG, R. A user-oriented software reliability model. *IEEE Trans. Software Engineering* 6, 2 (March 1980), 118–125.
- [3] FRIEDMAN, M., AND VOAS, J. *Software Assessment*. John Wiley & Sons, New York, 1995.
- [4] KRISHNAMURTHY, L., AND MATHUR, A. The estimation of system reliability using reliabilities of its components and their interfaces. In *Proceedings 8th Intl. Symposium on Software Reliability Engineering* (Albuquerque, New Mexico, Nov. 1997).
- [5] LAPRIE, J.-C., AND KANOUN, K. *Software Reliability and System Reliability*. In Lyu [7], 1996, pp. 27–70.
- [6] LEWIS, E. *Introduction to Reliability Engineering*, 2 ed. John Wiley & Sons, New York, 1996.
- [7] LYU, M., Ed. *Handbook of Software Reliability Engineering*. McGraw-Hill, New York, 1996.
- [8] LYU, M. *Reliability Theory, Analytical Techniques and Basic Statistics*. In Lyu [7], 1996, pp. 747–779.
- [9] MUSA, J. Sensitivity of field failure intensity to operational profile errors. In *Proceedings ISSRE (Fifth International Symposium on Software Reliability Engineering)* (Monterey, California, Nov. 1994), IEEE, pp. 334–337.
- [10] MUSA, J. Applying operational profiles in testing. In *Proc. 10th Intl. Software Quality Week* (San Francisco, CA, May 27–30 1997).
- [11] PARNAS, D. On a “Buzzword”: Hierarchical structure. In *IFIP Congress* (1974), North-Holland Publishing Co., pp. 336–339.
- [12] WOIT, D. Specifying component interactions for modular reliability estimation. In *First Intl. Quality Week Europe* (Brussels, Belgium, Nov. November 4–6 1997).
- [13] WOIT, D., AND MASON, D. Software component independence. In *High Assurance Software Engineering, (HASE’98)* (Washington, DC, Nov. 1998).



Testing Metrics for Requirement Quality

2nd International Software Quality Week Europe '98
Software Assurance Technology Center

Space Flight Center - NASA

Theodore Hammer, NASA/GSFC

301-286-7475 thammer@pop300.gsfc.nasa.gov

Linda Rosenberg, Ph.D., Unisys Federal Systems

Lenore Huffman, Unisys Federal Systems



Overview

Introduction

Requirements Specification

- Language
- Structure

Requirements Verification

- Volatility
- Traceability

Metrics Application

Lessons Learned

Conclusion



Objectives

Goal: High quality requirements

Expectation:

High quality requirements ==>

More effective testing (better, cheaper, faster)

Questions:

What constitutes quality in requirements?

How does requirement quality relate to testing?

What metrics are applicable?



Definitions

IEEE:

Quality - Degree *requirements* possesses desired combination of

Quality Attribute - Characteristic of the *requirements* that affects the quality and is measurable using quality metrics

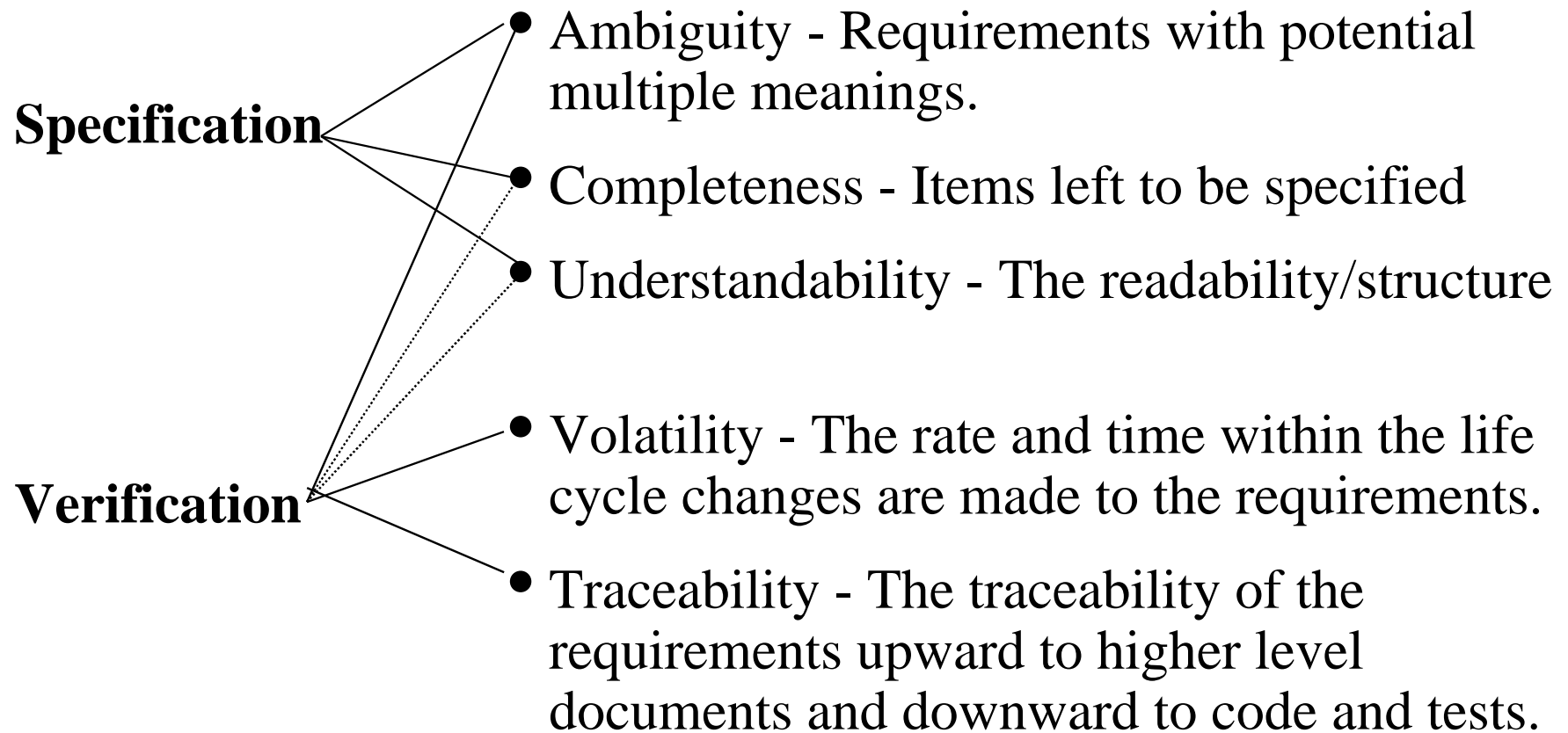
Evaluation of quality => measurable attributes of *requirements*

Test case - to verify compliance with a specific requirement

Test coverage - degree to which a set of tests addresses all specified requirements for a given system



Requirements Attributes





Requirement Specification Metrics

- **Ambiguity** = Weak Phrases (adequate, as appropriate, as applicable, but not limited to, normal, if practical, timely, as a minimum) + Options (can, may, optionally)

Completeness = TBD + TBA + TBS + TBR

- **Understandability** = Numbering Scheme
- **Traceability** = Number of Items traced to tests, between builds, between levels of detail

Number of Requirements: = *Imperatives* (shall, must, will, required, responsible for, should, are to, are applicable) + *Continuances* (below:, as follows:, following:, listed:, in particular, support:, :)



Requirements Document Analysis Example

Automated Requirements Measurement Tool*

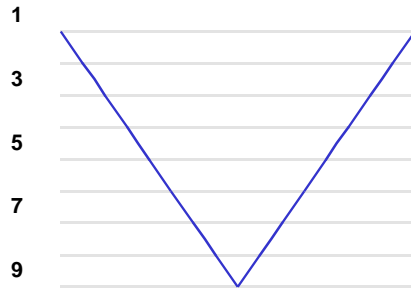
56 DOCUMENT	Lines of Text - Count of the physical lines of text	Imperatives - shall, must, will, should, is required to, are applicable, responsible for	Continuances - as follows, following, listed, in particular,	Directives - figure, table, for example, note:	Weak Phrases - adequate, as applicable, as appropriate, as a minimum, be able to, be capable, easy, effective, not limited to, if practical	Incomplete (TBD, TBS)	Options - can, may, optionally
Minimum	143	25	15	0	0	0	0
Median	2,265	382	183	21	37	7	27
Average	4,772	682	423	49	70	25	63
Maximum	28,459	3,896	118	224	4	32	130
Stdev	759	156	99	12	21	20	39
Project X	34,664	1,176	714	873	13	480	187

*available free from <http://satc.gsfc.nasa.gov>



Structure Level at Which Imperative Occurs

Derived

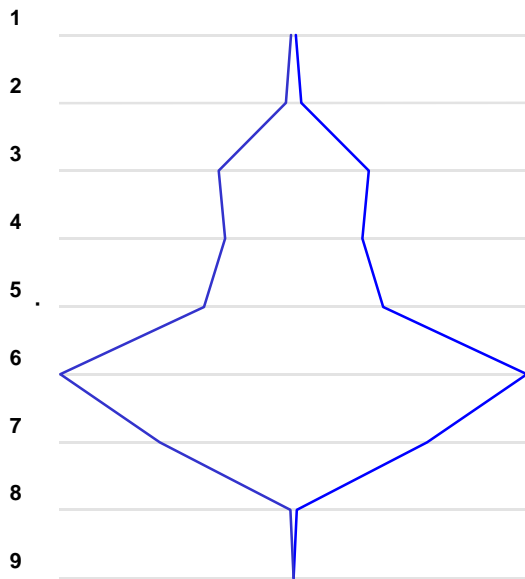


Expected

Detailed



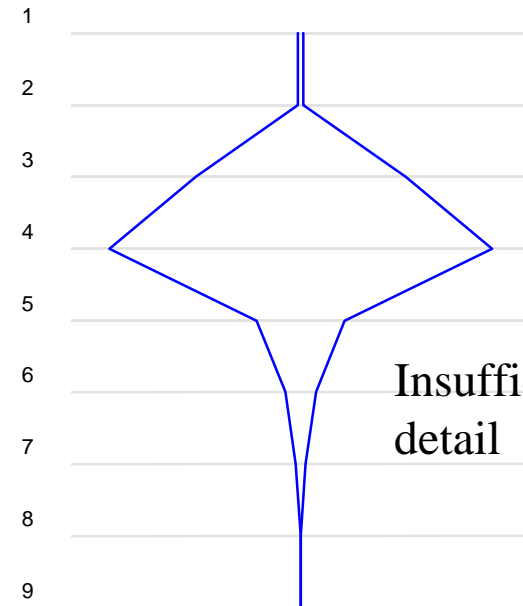
Derived



Too much detail
too soon in
development

Actual

Detailed



Insufficient
detail



Requirement Verification

Issues critical to testing:

Volatility:

- Is requirement volatility zero?

Is requirement movement between builds stable?

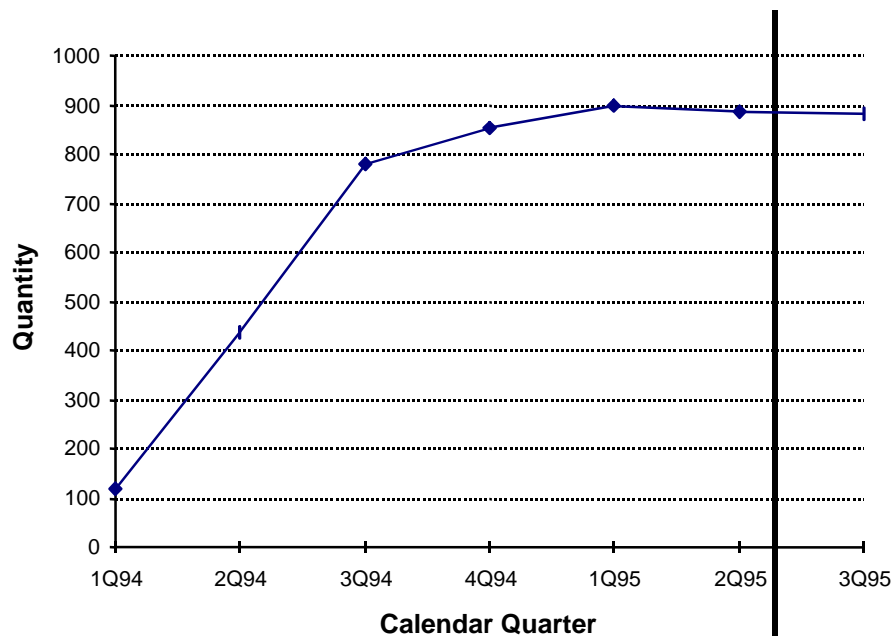
Traceability:

- Do all requirements trace to higher and lower level documents?

Are all requirements tested? Do they trace to a test?

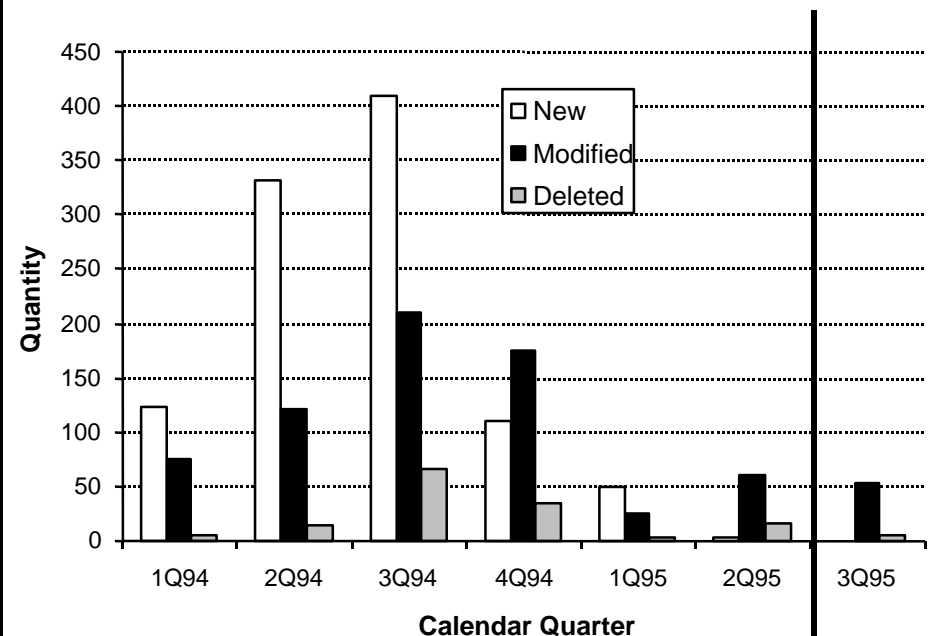
Requirement Volatility

Total Number of New Requirements



**CDR
Looks Good!
(Stable)**

Modifications to Requirements



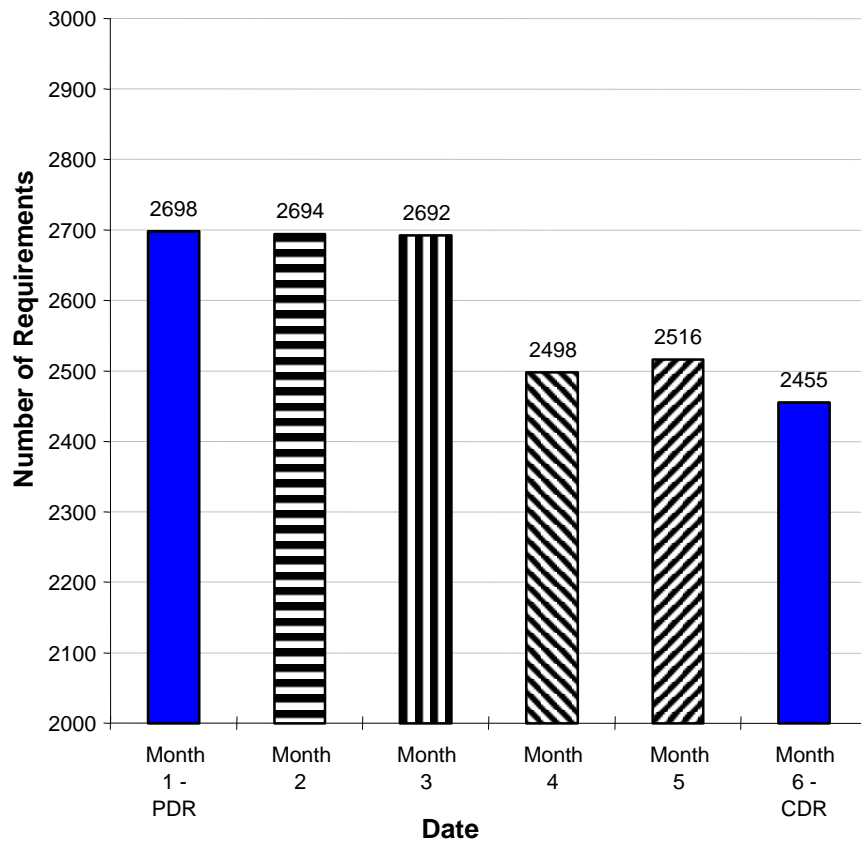
**CDR
Excessive Changes!
NOT Stable**

Combination of BOTH views indicate risk area - requirements are NOT YET stable

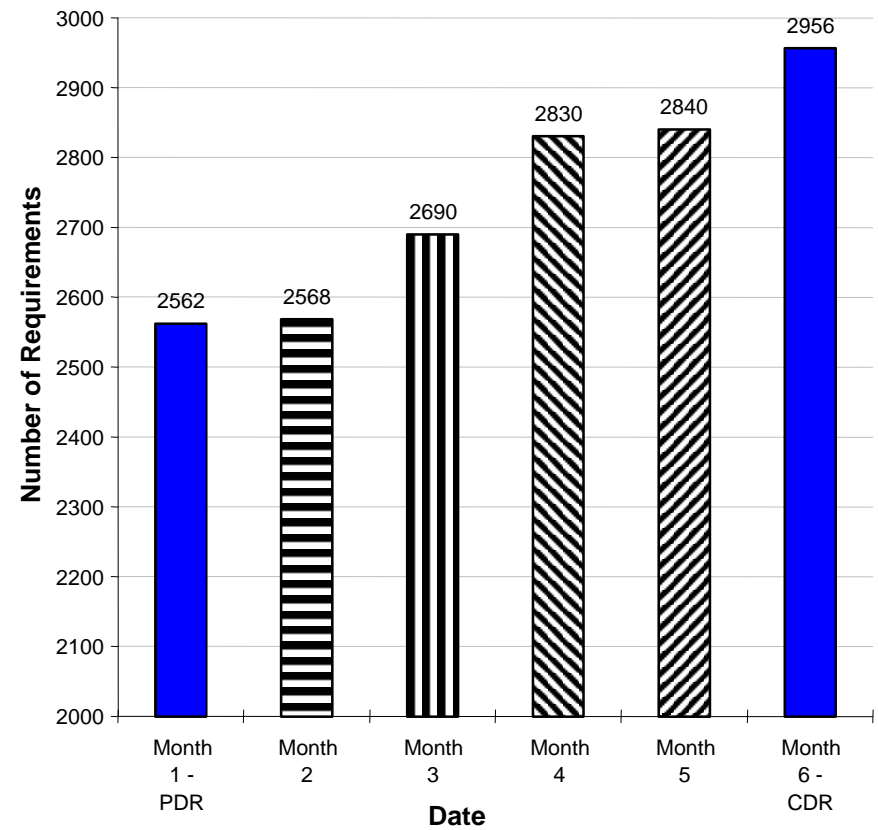


Requirement Traceability - By Build

BUILD A



BUILD B

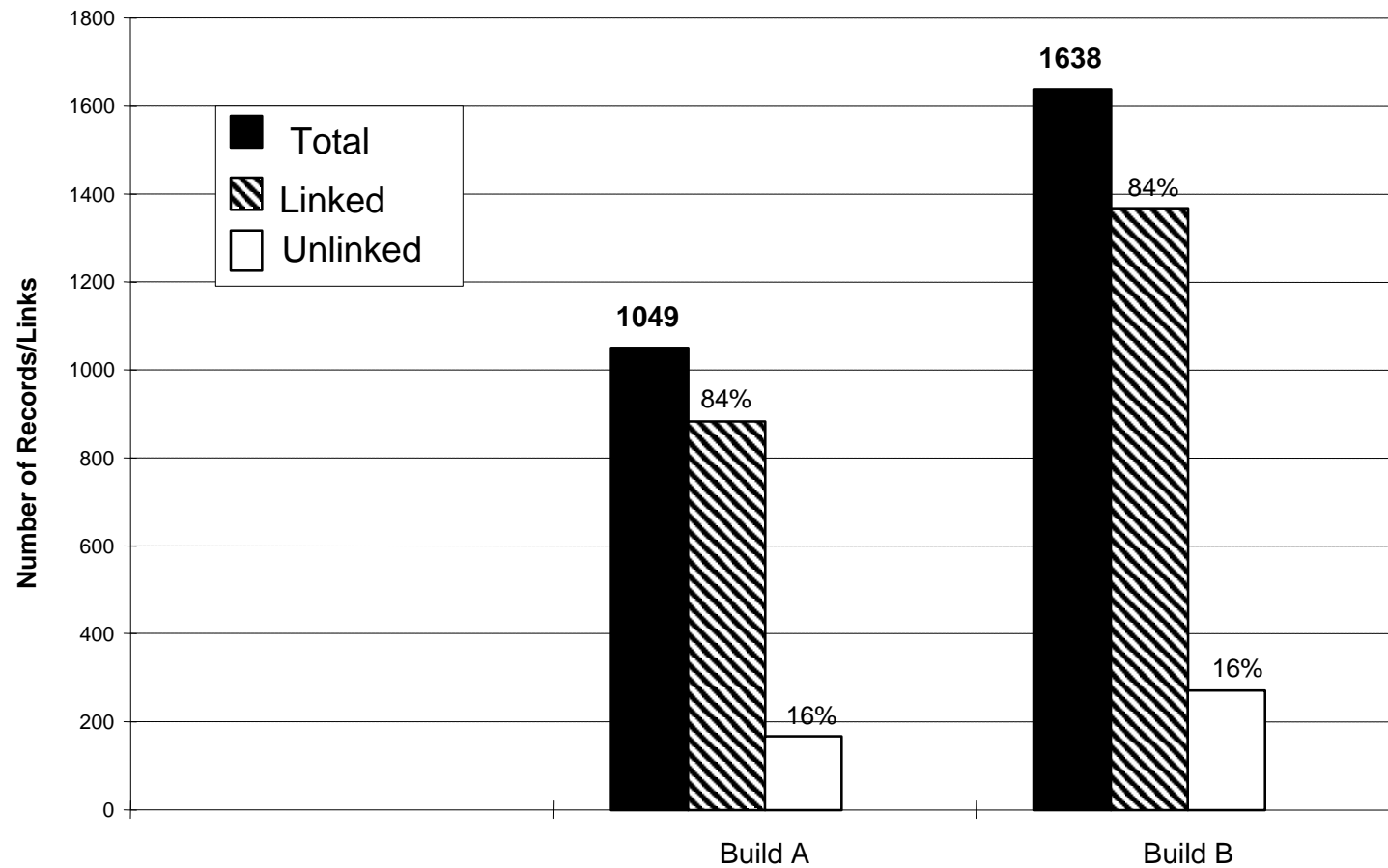


(Detailed)



Requirement Traceability

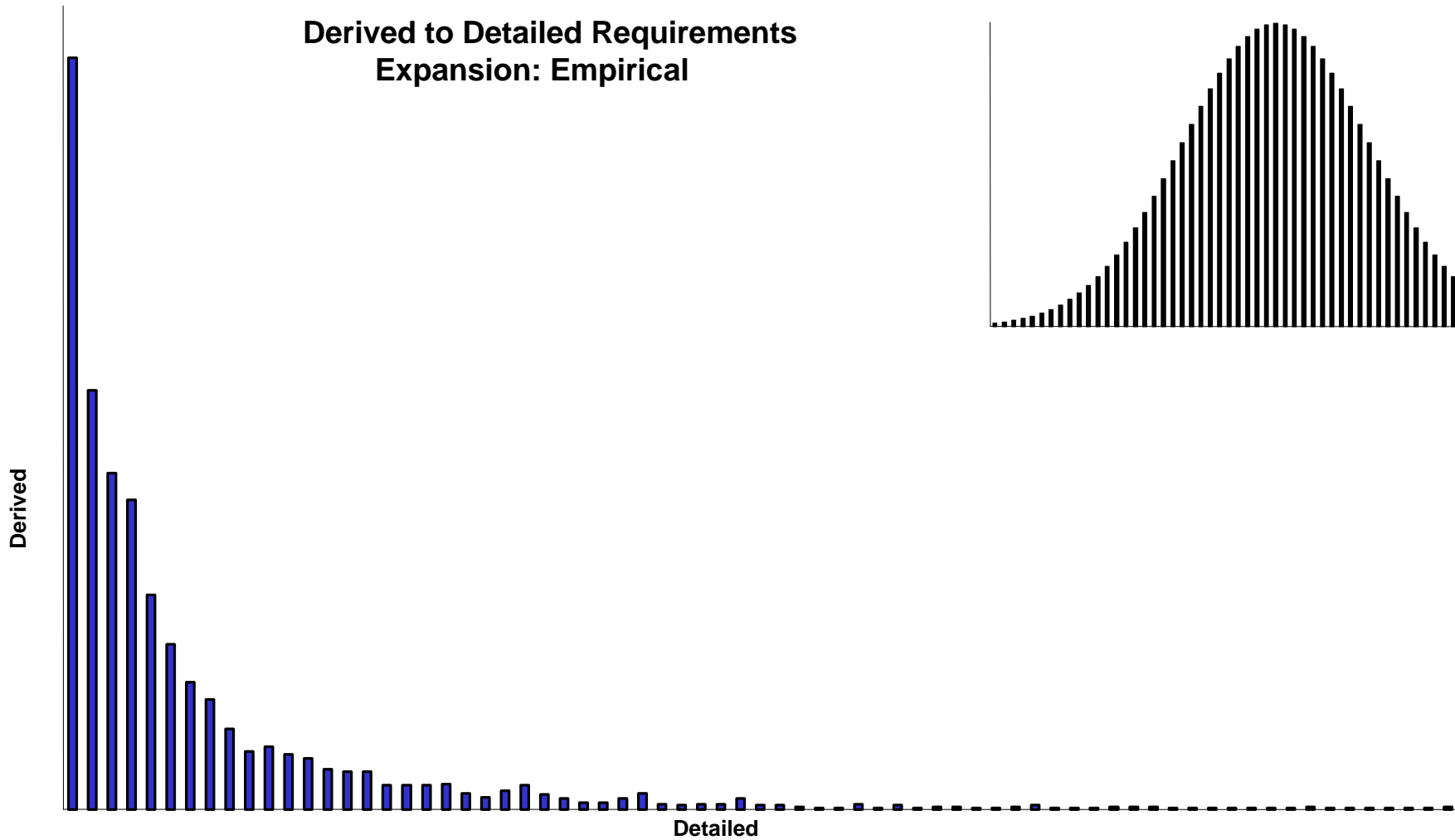
Derived to Detailed





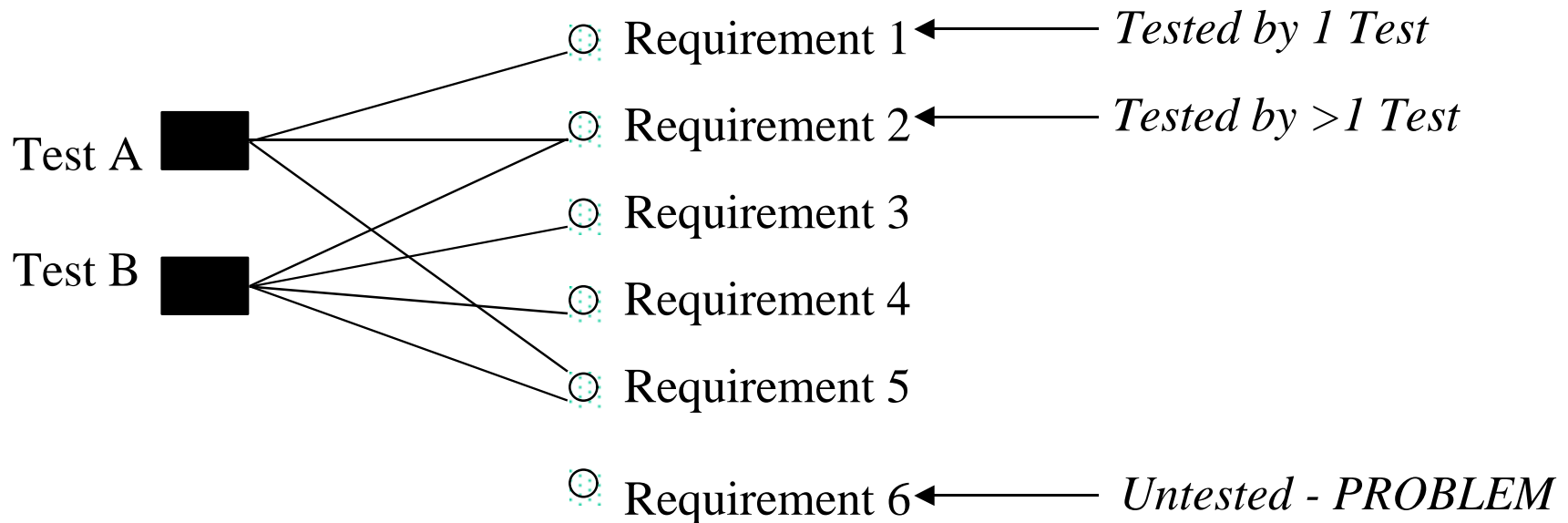
Requirement Decomposition

Derived to Detailed Requirements
Expansion: Empirical





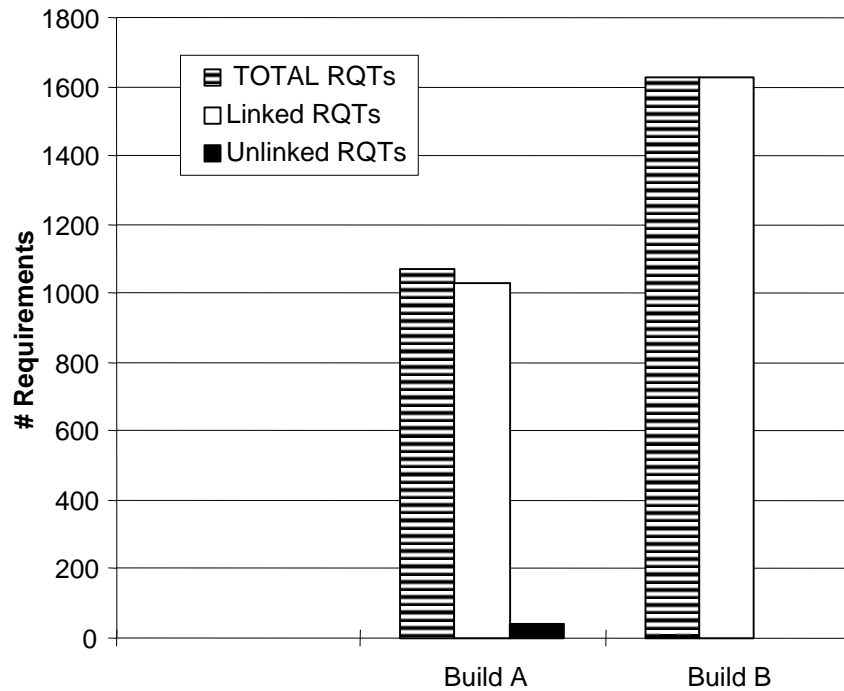
Traceability - Requirement Links to Tests



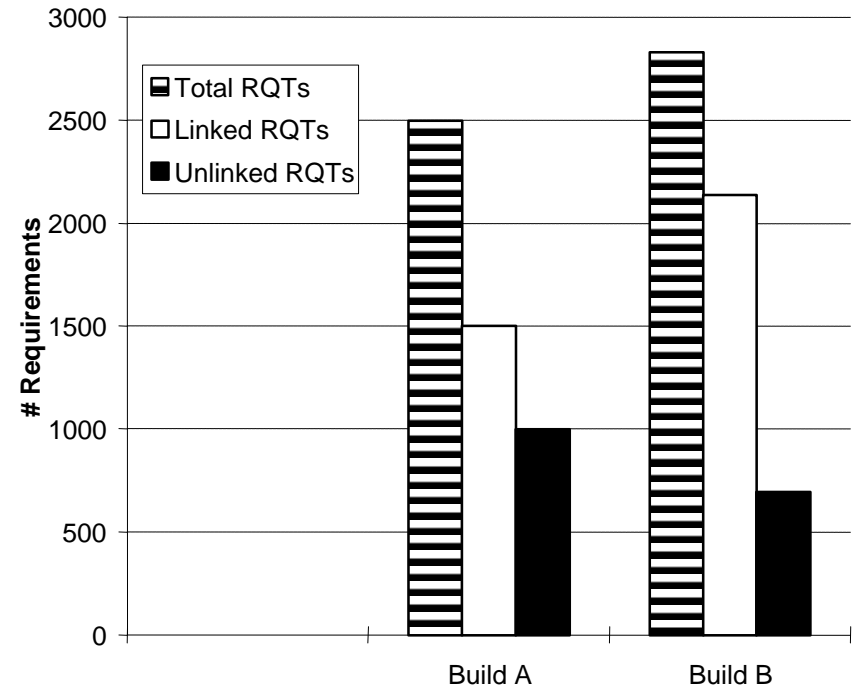
Sample Linkage



Requirement Verification - Trace to Test



Derived Requirements



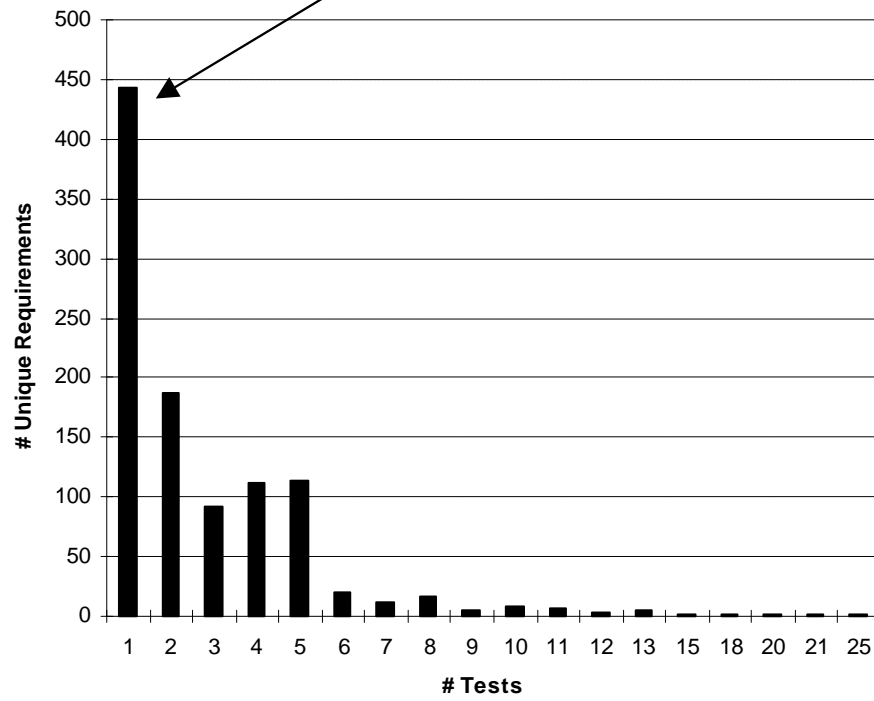
Detailed Requirements



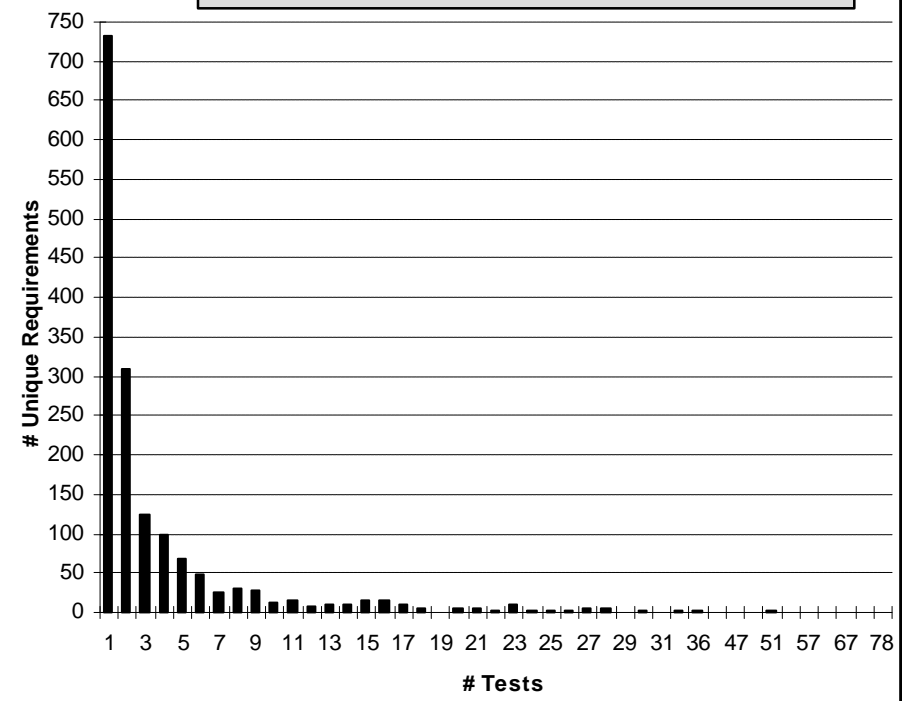
Test Span

System test Profile (CDR)

445 requirements are each tested by only 1 test



Build A



Build B



Requirement Repository Metric Capabilities

	Word Processor	Spreadsheet	Relational Database	Requirement Tool
Document size	X			
Dynamic changes over time				X
Release size	X	X	X	X
Requirement expansion profile			X	X
Requirement types	X	X	X	X
Requirement verification			X	X
Requirement volatility	X	X	X	X
Test coverage			X	X
Test span			X	X
Test types	X	X	X	X



Metric Application

- Specification Quality (from the ARM Tool) :
 - Prototyping, Special Studies
 - Later refinement and allocation to later build

Requirement Expansion and Volatility (from Requirement Management Tools):

- Reassessment
- Prototyping, Special Studies, Reallocation

and Coverage (from Requirement Management

- Completion of test matrix incrementally
- Focused review of test procedures



Lessons Learned

- Metrics should be used in the requirement phase

Requirements management tools should be used as much as possible

Metrics must be incorporated into management decision and risk management feedback loops



Conclusion

- Quality documentation: complete, concise, clear ==> leads to quality testing program
Requirement volatility impacts testing and must be

Verification program: fully traceable and structured

Effective requirement management: appropriate application of requirement database tool through which the requirements are maintained through the

Metrics are a powerful tool that provide insight into testing of requirements.

Testing Metrics for Requirement Quality

Dr. Linda H. Rosenberg, Ph. D., Theodore Hammer, Lenore Huffman

Key Words: Requirements, Metrics, Quality, Testing

1. INTRODUCTION

It is generally accepted that requirements are the foundation upon which the entire system is built. And that requirement verification and validation is needed to assure that the functionality representing the requirements has indeed been delivered. However, all too often requirements are not satisfied, leading to a process of fixing what you can and accepting the fact that certain functionality will not be there. A better approach is to develop requirements that are complete, concise and clear, and that provide the implementer a clear blueprint with which to build the system. This is not done by magic but through the application of tools and metric analysis techniques in the areas of requirement specification and requirement verification.

Because both parties must understand requirements that the acquirer expects the provider to contractually satisfy, specifications are usually written in natural language. The use of natural language to prescribe complex, dynamic systems has at least two severe problems: ambiguity and inaccuracy. Many words and phrases have dual meanings that can be altered by the context in which they are used. Defining a large, multi-dimensional capability within the limitations imposed by the two dimensional structure of a document can obscure the relationships between individual groups of requirements. It is important to know the attributes for requirement quality:

- Ambiguity - Requirements with potential multiple meanings.
- Completeness - Items left to be specified.
- Understandability - The readability of the document.
- Volatility - The rate and time within the life cycle changes are made to the requirements.
- Traceability - The traceability of the requirements upward to higher level documents and downward to code and tests.

Requirements based testing is critical in the implementation of software systems. Automated tools, if properly used, open the door to assessing the scope and potential effectiveness of the test program. Proper implementation of a database to not only track requirements at each level of decomposition, but also the tests associated with the verification of these requirements affords the project a wealth of information. From this database the project can gain important insight into the relationship between the test and requirements.

This paper will demonstrate how metrics can help in these three areas of requirement development. Examples will be provided how metrics can identify areas of weakness that should be corrected, through the use of data from a large NASA project, Project X. Lessons learned will also be listed to aid in keeping a project, large or small, on track.

2. REQUIREMENT SPECIFICATION

The importance of correctly documenting requirements has caused the software industry to produce a significant number of aids [1] to the creation and management of the requirements specification documents and individual specifications statements. However very few of these aids assist in evaluating the quality of the requirements document or the individual specification statements themselves. The SATC has developed a tool to parse requirements documents. The Automated Requirements Measurement (ARM) software was developed for scanning a file that contains the text of the requirements specification. During this scan process, it searches each line of text for specific words and phrases. These search arguments (specific words and phrases) are indicated by the SATC's studies to be an indicator of the document's quality as a specification of requirements. ARM has been applied to 56 NASA requirement documents. Seven measures were developed, as shown below.

1. Lines of Text - Physical lines of text as a measure of size.
2. Imperatives - Words and phrases that command that something must be done or provided. The number of imperatives is used as a base requirements count. [Shall, must or must not, is required to, are applicable, responsible for, will, should]
3. Continuances -Phrases that follow an imperative and introduce the specification of requirements at a lower level, for a supplemental requirement count. [As follows, below, following, in particular, listed, support]
4. Directives – References provided to figures, tables, or notes.
5. Weak Phrases - Clauses that are apt to cause uncertainty and leave room for multiple interpretations measure of ambiguity. [Adequate, as applicable, as appropriate, as a minimum, be able to, but not limited to, be capable of, effective, easy, effective, if effective, if practical, not limited to, normal, timely]
6. Incomplete – Statements within the document that have TBD (To be Determined) or TBS (To Be Supplied).
7. Options - Words that seem to give the developer latitude in satisfying the specifications but can be ambiguous. [Can, may, optionally]

It must be emphasized that the tool does not attempt to assess the correctness of the requirements specified. It assesses individual specification statements and the vocabulary used to state the requirements, and also has the capability to assess the structure of the requirements document.¹

To see how this tool would be used to assess the “quality” of the requirements document, the Project X Derived requirements document was analyzed using the ARM Tool. Table 1 shows the results.

¹ This tool is available at no cost from the SATC web site <http://satc.gsfc.nasa.gov>

56 DOCUMENT	Lines of Text - Count of the physical lines of text	Imperatives - shall, must, will, should, is required to, are applicable, responsible for	Continuances - as follows, following, listed, in particular, support	Directives - figure, table, for example, note.	Weak Phrases - adequate, as applicable, as appropriate, as a minimum, be able to, be capable, easy, effective, not limited to, if practical	Incomplete (TBD, TBS)	Options - can, may, optionally
Minimum	143	25	15	0	0	0	0
Median	2,265	382	193	21	37	7	27
Average	4,772	682	423	49	70	25	63
Maximum	28,459	3,896	118	224	4	32	130
Stdev	759	156	99	12	21	20	39
Project X	34,664	1,176	714	873	13	480	187

Table 1 - Requirements Specification Analysis Example

Several things can be seen from this analysis. First, the document shows some strengths. There appears to be a good number of imperatives, and the number of weak phrases is low as compared to the family of NASA documents processed through the ARM tool to date. However, the document shows some significant weaknesses. The document has a large amount of text given the number of imperatives. This gives an indication of being a wordy document, which can have the effect of obscuring the requirements, preventing the requirements from being clear and concise. The document also has a large number of incomplete requirements, containing TBDs and TBSs. It could even be said that this document is not ready for use on this point alone, as this implies that there is still uncertainty about what the system is required to do. It is very difficult to build a system that has undefined requirements. Also this document has a large number of options, which increases the uncertainty about what is really required of the system that is to be developed. Options leave decisions about what the system is to do to the implementers, many times without sufficient direction or instruction about option selection criteria. As a result the implementation varies widely, anything from some of the options to none at all (especially since these items are options and not “really” required).

A further understanding of the requirements documentation can be achieved by looking at the document structure. Figure 2 shows the expected structure, based on other NASA documentation, and actual structure for documentation from Project X. The expected structure is a graphical representation of the numbering structure used within the requirements documentation. The levels represent sub tiers within a section. For example four sub tiers would be 1.0, 1.1, 1.1.1, and 1.1.1.1. The expected graph for the Derived Specification indicates that there are many more high level requirements than detailed requirement expansions. This makes sense, as the Derived Specification is to define the overall requirements of the system and not provide details. The expected graph for the Detailed Specification shows the opposite. There are many more detailed expansions of the requirements than of high level statements. Again this makes sense, as the detailed requirements document is to be the basis for the implementation of the system. The Project X documentation show some disturbing weaknesses. The Derived Specification shows a trend to over specify some of the requirements too early in the life cycle. The Detailed Specification shows not enough detail. The weakness of the Detailed Specification

may be resultant from the trend to over specify requirements in the parent, Derived Specification, or most probably is the result of the Derived Specification having too many incomplete requirements and options (as seen from the first analysis using the ARM Tool).

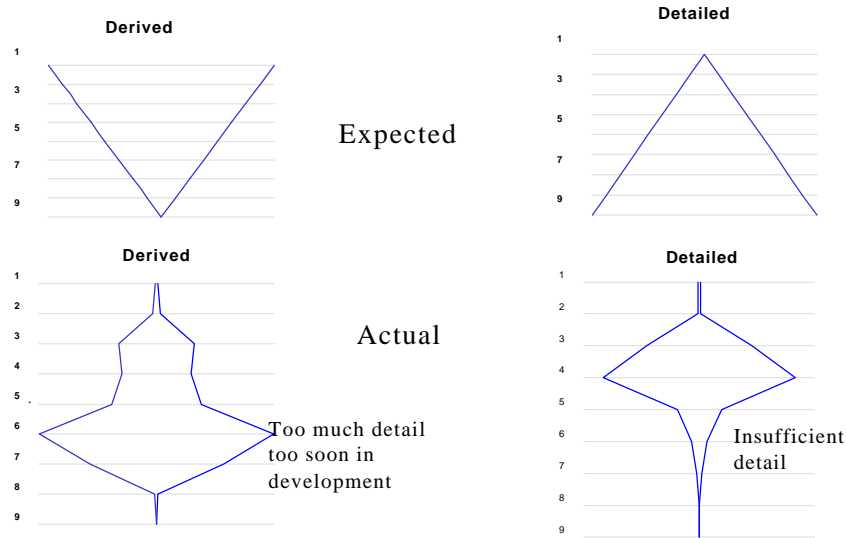


Figure 2 - Structure Level at Which Imperative Occurs

Obtaining a good quality specification has always been a desire of engineers but there has been little available in terms of analysis tools that would allow them to visualize the quality of the documentation. Now with the ARM Tool the quality aspects of the documentation can be visualized in such a way as to allow actions to be taken to improve the documentation.

3. REQUIREMENT VERIFICATION

Requirements testing is another important aspect of the requirements phase. Though this may not be seen as directly related to the issue of developing quality requirements, it is crucial because delivered capability cannot be determined without an effective verification program. In looking at the verification program, a further understanding of the nature of the requirements must be attained. This is done by looking at requirement stability and expansion. The linkage of requirements to test cases is reviewed, and then a test profile is made to characterize the entire test program. Again, data from Project X is used to demonstrate the utility of metrics in understanding requirement verification.

Requirement stability impacts the verification effort in that testing can not be planned or designed with the requirements continually in a state of flux.

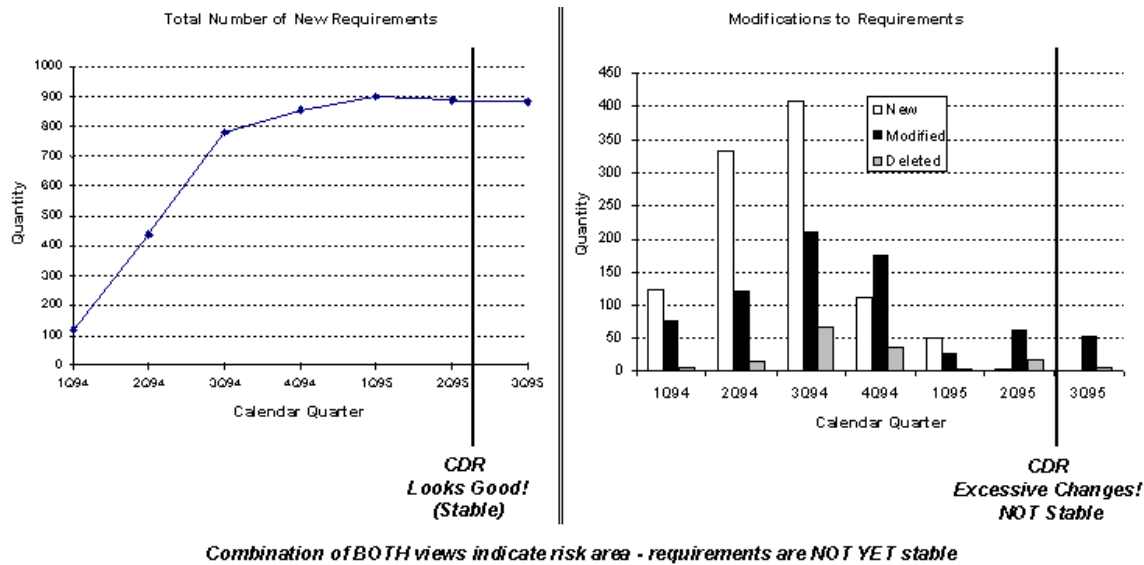


Figure 3 - Requirement Stabilization - Volatility

Figure 3 show how metrics can be used to gain insight into requirement stability and the importance of looking a particular issue in more than one way. This figure shows that the total number of requirements stabilized in time for the Critical Design Review (CDR), which is what is desired. However, when one looks at requirement stability in terms of new, modified, and deleted requirements one notices that the requirements are not that stable. There is almost constant change occurring in the modification of requirements. This will endanger the verification program. Another way of viewing requirement stability is to look at the allocation of requirements to the individual builds or releases. Figure 4 show the allocation of Detailed requirements to Build A and Build B for Project X.

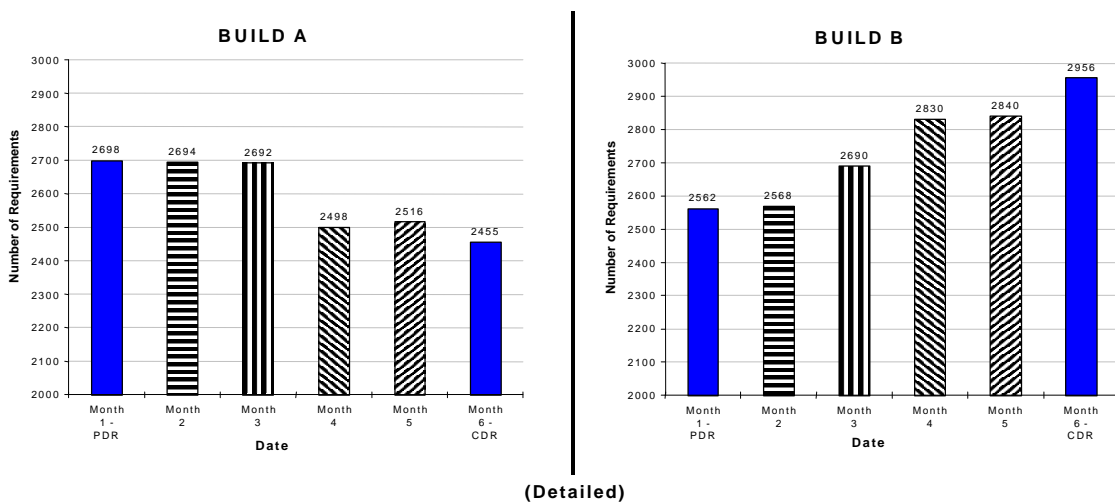


Figure 4 - Requirement Stabilization by Build

What can be seen is that requirements are continually being moved or reallocated from Build A to Build B. This instability will make the implementation and verification of Build B difficult, as many requirements have been pushed into the last build in the development effort.

Requirement stability can be viewed in terms of requirement traceability and expansion. Requirements traceability is the linkage of the requirements at one level to the requirements at the next lower level. If there is missing linkage, a case can be made that possibly more requirements need to be written. Requirement expansion is the measure of how many requirements at the Detailed level were written to completely satisfy the Derived requirements. If there is little expansion in the number of requirements, a case may again be made again that there should be more requirements written to provide the level of detail necessary to implement the system. Figure 5 shows the linkage of Derived requirements to Detailed requirements.

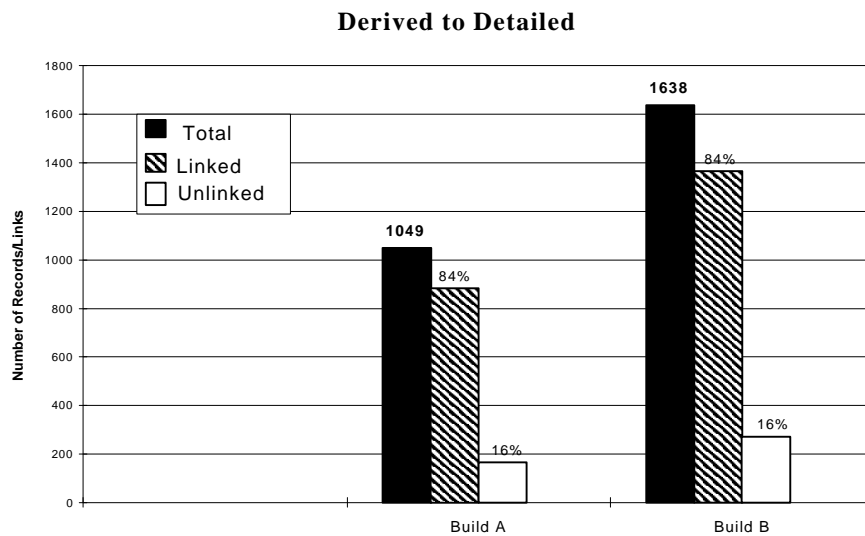


Figure 5 - Requirement Traceability

In both cases there is missing linkage (white bar of graph) between Derived and Detailed requirements, indicating that the Detailed requirements are potentially incomplete if a CDR was held for any one of these builds.

In reviewing requirement expansion, a comparison is made with data compiled from NASA projects which leads to an expected curve for requirement expansion that is bell shaped. This reflects that few requirements are expected to have little expansion or be expanded to a large number of requirements at the next lower level. As a result, there tends to be an average number of requirements written to decompose the Derived requirements to the next level. Figure 6 shows the situation for Project X.

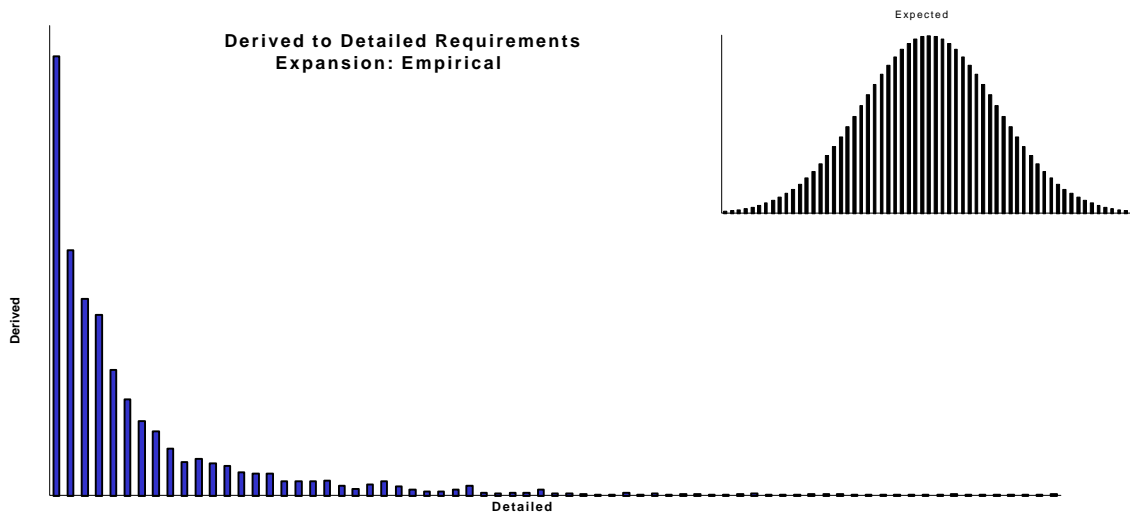


Figure 6 - Requirement Decomposition

Here we see that the Derived requirements for the most part have not been expanded while there are a few that have many requirements written to expand on the Derived requirements. This situation correlates very well with the metrics developed from the analysis of the documentation structure mentioned above, where the structure of the Detailed requirements specification showed a lack of detail. This lack of detail not only jeopardizes the implementation effort but also the development of effective verification procedures.

The objective of an effective verification program is to ensure that every requirement is tested, the implication being that if the system passes the test, the requirement's functionality is included in the delivered system [1,2]. An assessment of the traceability of the requirements to test cases is needed. It is expected that a requirement will be linked to a test case, and may well be linked to more than one test case as shown in Figure 7 [3,4].

The important aspect of this analysis is to determine which requirements have not been linked to any test cases at all.

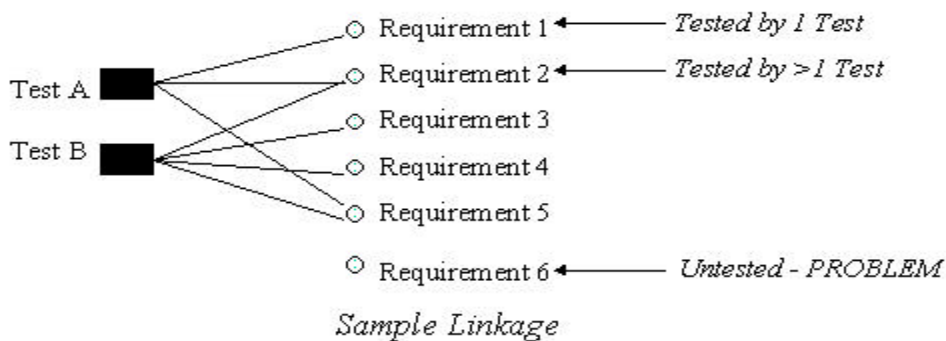


Figure 7 - Requirement Verification - Trace to Test Linkage

Figure 8 shows that the traceability of requirements to test cases for Project X around the CDR time frame for Build A. The information was extracted from the requirements management database used in support of the development effort. The profiles show several problems.

The test program for Build B is further along than that for Build A, when it is Build A that will be developed and tested before Build B. Resources may have been inappropriately allocated to the development of the test program for Build A. Lastly, the test program for the Detailed requirements is behind that for the test program for the Derived requirements. Again, this is backwards. The first tests to be executed will be that for the Detailed requirements, the system tests, and after that tests for the Derived requirements will be executed, the acceptance tests. An explanation for this problem may be found is a previously presented metric. Remember the metric showing the push of Detailed requirement from Build A to Build B. This movement of requirements from Build A to Build B may well be the cause of the lack of traceability of requirements to test cases. The test case developers may be having difficulty in keeping up with the changes in requirements resulting in a number of requirements in each build without a link to a test case.

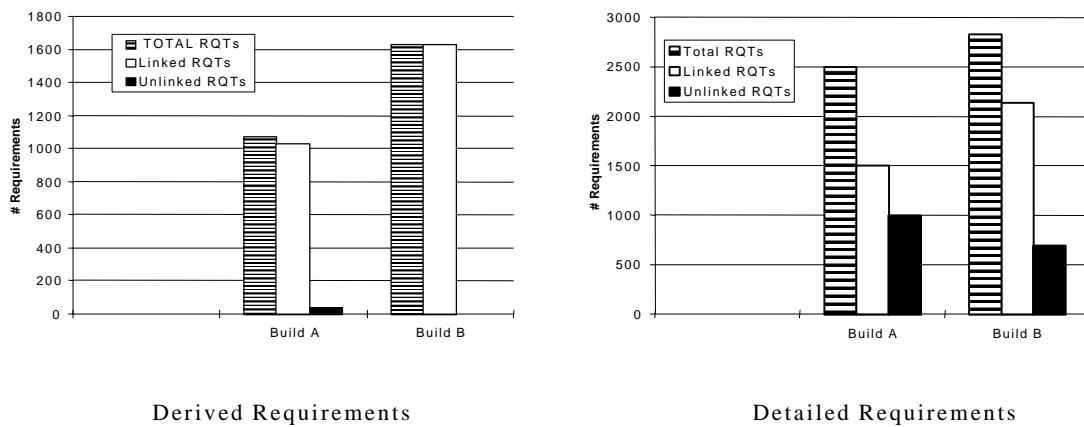


Figure 8 - Requirement Verification Trace to Test

Not only is it important to understand whether all the requirements are linked to test cases, but also to understand the character of the test program. This can be done by looking at the profile and relationship of requirements to test cases. This provides an understanding of the nature of the test program. Figure 9 shows an expected profile of unique requirements per test case based on data from NASA projects [5].

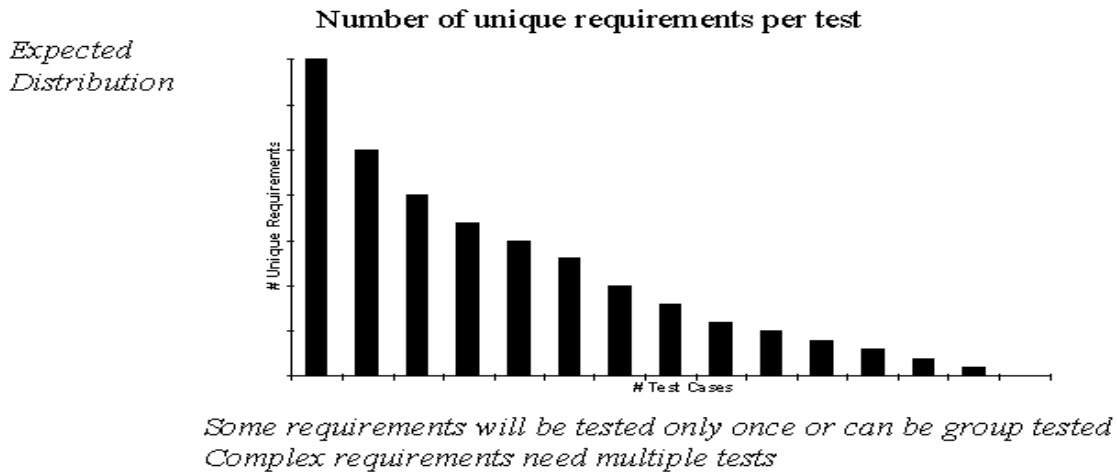


Figure 9 - Test Program Characterization Tests per Requirement

This profile shows that there is an expectation that there will be a large number of requirements tested by only one test case, and that there will be some number of requirements that will be tested by a multiple number of test cases. It is expected that the upper bound of multiple test cases will range in the tens. This makes sense, as more complicated requirements may require different test cases to thoroughly verify all aspects of the requirement. However, there is a limit on the number of test cases. As the number of test cases increases the difficulty in verifying the requirement increases, due to the complication in data analysis, understanding the results of the multiple tests cases, and understanding the impact of multiple test case results on the verification of the requirement. Figure 10 shows the requirement to test case profile for Project X. There is a good indication that there are a large number of requirements covered by just one test, making for a simple, easy to evaluate test program for a significant part of the system requirements. However, there are several instances for both Build A and B where there are several tests for unique requirements. Notice that for Build A that one requirement has been linked to 25 test cases, and in Build B that one requirement is linked to 51 test cases. This large number of test cases may well make it impossible to verify that these requirements have been implemented.

In summary the verification program for Project X has some strengths; the total number of new requirements is stable, and the Derived requirements have good linkage to tests for the acceptance test program. But there are also significant weaknesses. There was a shifting of requirements between builds late in the requirement phase. Requirements were not completely decomposed from the Derived requirements to the Detailed requirements. The Detailed test program showed a significant number of requirements without links to tests. Test programs for both Derived and Detailed requirements showed some excessive testing of requirements.

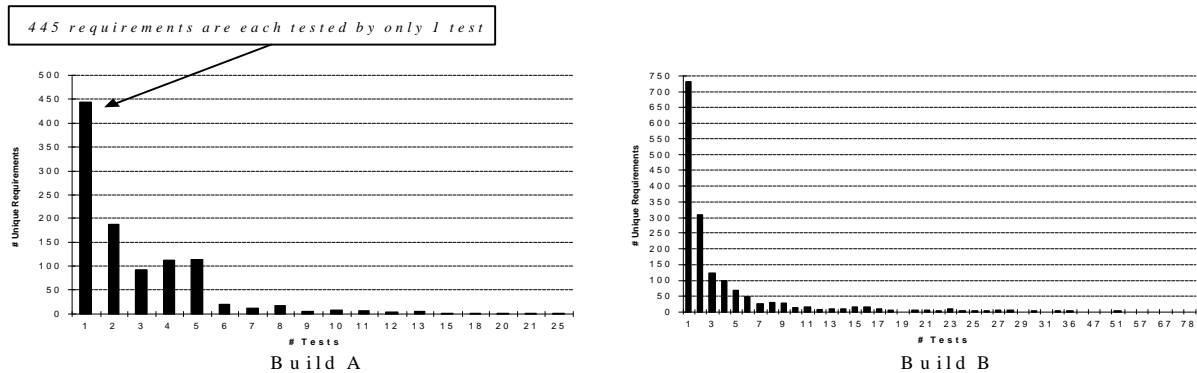


Figure 10 - Test Program Characterization Tests per Requirement

4. REQUIREMENT MANAGEMENT

The use of tools to aid in the management of requirements has become an important aspect of system engineering and design. Considering the size and complexity of development efforts, the use of requirements management tools has become essential. The tools which requirement managers use for automating the requirements engineering process have reduced the drudgery in maintaining a project’s requirement set and added the benefit of significant error reduction. Tools also provide capabilities far beyond those obtained from text-based maintenance and processing of requirements. Requirements management tools are sophisticated and complex – since the nature of the material for which they are responsible is finely detailed, time-sensitive, highly internally dependent, and can be continuously changing. Tools that simplify complex tasks require skill and a thorough understanding of their capabilities if they are to perform effectively over the lifetime of a project [6].

There are many requirement management tools to choose from. These range from simple word processors, to spreadsheets, to relational dbs, to tools designed specifically for the management of requirements such as DOORS (Quality Systems & Software - Mt. Arlington, NJ) or RTM Requirements Traceability Management (Integrated Chipware, Inc. - Reston, VA). The key to selecting the appropriate tool is the functionality (See Table 2 for a comparison of tool capabilities) provided and the capability to develop metrics from the data, secondary contained in the tool.

	Word Processor	Spreadsheet	Relational Database	Requirement Tool
Document config. mgt	X		X	X
Document preparation	X			X
Function decomposition			X	X
Report preparation			X	X
Requirement allocation		X	X	X
Requirement config. mgt		X	X	X
Requirement expansion			X	X
Requirement importation				X
Requirement simplification				X
Requirement storage	X	X	X	X
Requirement traceability			X	X
Test coverage/adequacy			X	X
Metrics			X	X

Table 2 - Requirement Repository Capabilities

The metric capability of the tool is important. It should be noted that most of the metrics presented in this paper to demonstrate how to do requirements the right way were developed from the data contained in a requirement management tool. Table 3 shows a comparison of the metric capability associated with the different tools. Clearly, the relational database and requirements management tool provide the capabilities needed to effectively support the management of requirements.

	Word Processor	Spreadsheet	Relational Database	Requirement Tool
Document size	X			
Dynamic changes over time				X
Release size	X	X	X	X
Requirement expansion profile			X	X
Requirement types	X	X	X	X
Requirement verification			X	X
Requirement volatility	X	X	X	X
Test coverage			X	X
Test span			X	X
Test types	X	X	X	X

Table 3 - Requirement Repository metric Capabilities

5. METRICS APPLICATION

The examples cited above from the experiences with Project X provide some sense of how metrics could be applied to real project experiences. The following provide further examples how the application of metrics can and have been used to improve product quality and test processes in support of projects.

The ARM tool not only provides an overview of the requirements specification quality but also provides valuable ancillary information on specifically where in the requirements specification the quality problems (e.g., options, ambiguities) exists. With this information the project has several options available to solve the identified problem. A prototype effort could be established with the ultimate end result of removing the uncertainties of specific functional requirements. A special studies effort could be formed to resolve TBD (To Be Determined) and TBS (To Be Supplied) uncertainties from the specification. Another avenue could be to assign the problem functional area to a later build with a requirement baseline review established to support the later build schedule. The review is critical. Assigning TBD and TBS requirement uncertainties to a later build without an established resolution date for the issues will only jeopardize the success of the later build and possibly the total system delivery.

Through the use of requirement management and configuration management tools the project can develop metrics on requirement expansion and requirement volatility. Information can be developed to determine which functional areas are the least understood (lack of adequate requirement expansion) or are the most volatile (unsettled or confused user needs). This can lead to strategies of focused prototyping, special studies, allocation of the functional area to later builds, reassessing the need for the unsettled functional area, or an adaptive approach to define functionality that can be adaptable to the kinds of changes experienced to date.

Again using requirement management tools metric insights can be obtained of the test program. The completeness of test traceability to requirements, the coverage of the testing (how many tests are traced to a requirement), complexity of the testing (how many requirements are traced to tests), and overall characterization of the test program (does look fairly consistent with expected curves). Detailed insight can help the project to develop strategies for focused assessment of test procedures (do specific procedures really verify the requirements mapped to them), ensure test traceability is complete by the test review for each build, and with understanding of the complexities of the test program establish test priorities.

6. LESSONS LEARNED

The most important lesson learned is that metrics are available and can be an effective tool for the project early in the development life-cycle, specifically the requirements phase. As stated earlier in this paper the return on investment for efforts to remove as many errors as possible from the requirements is very significant. Another lesson learned is the significant benefits from the use of requirement management tools. Many of the metrics presented in this presentation were developed from requirement management tool databases. These tools should be used on any size project if at all possible. And lastly metrics must be thoroughly integrated into the project management processes. The metrics collected must be meaningful and used by the

project to make decisions about the requirements and test program. If metrics are not a part of these processes, then there will be no benefit obtained from the effort expended to develop and report the metrics. Only by using metrics as one of the windows into the quality of products and effectivity of processes can the project receive the return on investment in a metrics program.

7. CONCLUSION

Quality documentation is complete, clear and concise. This used to be considered ethereal concepts, difficult to measure or visualize. Now with the advent of tools, like ARM, metrics can be developed to see the strengths and weaknesses of the requirement documentation. The completeness of the verification program used to be the only aspect that was easily understood. Now through the use of metrics, a project can gain insight into not only the completeness of the test program but to understand the overall characteristics of the verification program. Effective requirement management now demands the appropriate use of management tools and/or databases through the development life cycle. It is through their use that enables the development of metrics to gain insight into the quality of the requirements, take effective action to correct deficiencies, manage requirement volatility and ensure that a complete and effective test program is established to verify the total set of requirements.

8. REFERENCES

- [1] Brooks, Frederick P. Jr., No Silver Bullet: Essence and accidents of software engineering, *IEEE Computer*, vol. 15, no. 1, April 1987, pp. 10-18.
- [2] Hammer, T., Huffman, L., Rosenberg, L., Wilson, W., Hyatt, L., "Requirement Metrics for Risk Identification", Software Engineering Laboratory Workshop, GSFC, 12/96.
- [3] NASA, *Software Assurance Guidebook*, NASA Goddard Space Flight Center Office of Safety, Reliability, Maintainability, and Quality Assurance, 9/89.
- [4] Wilson, W., Rosenberg, L., Hyatt, L., "Automated Analysis of Requirement Specifications", Fourteenth Annual Pacific Northwest Software Quality Conference, 10/96.
- [5] Hammer, T., "Measuring Requirement Testing", 18th International Conference on Software Engineering, 5/97.
- [6] Hammer, T., "Automated Requirements Management – Beware How You Use Tools", 19th International Conference on Software Engineering, 4/98.
- [7] Hansen, Gary W., Hansen, James V., Database Management and Design, Prentice Hall, 1992.
- [8] Chen, M., Han, J., Yu, P. "Data Mining: An Overview from a Database Perspective", *IEEE Transactions on knowledge and Data Engineering*, Vol 8, No. 6, 12/96

9. BIOGRAPHIES

Theodore F. Hammer

Mr. Ted Hammer is the NASA manager for the Software Assurance Technology Center (SATC) at NASA's Goddard Space Flight Center (GSFC). The SATC, through associations with NASA and GSFC projects and organizations, seeks to improve GSFC and NASA software by improving software quality, reducing development risks, and lowering life cycle costs. A prime focus of the SATC is the provision of software metrics support to GSFC and NASA software development and acquisition projects. In order to meet the needs of these projects, the supporting research done by the SATC is essential to allow the assurance activities to keep pace with the changing software development environment. In addition, the SATC develops techniques, provides software assurance tools, and transfers this technology to NASA and industry.

Prior to this position, Mr. Hammer was a member of the Assurance Management Office where he is responsible for managing the overall quality assurance activities for specific ground system implementation projects, with special emphasis on software quality assurance. Mr. Hammer is also responsible for managing software quality assurance activities for selected spacecraft implementation projects.

Mr. Hammer has over 22 years experience in software development and assurance, 9 with the government at GSFC, and 14 with the government and private industry supporting the Naval Sea Systems Command (NAVSEA). Early in his career he was responsible for test software development for the Combat Direction System on destroyer and frigate classes of ships. He then became responsible for the hardware and software upgrades for the Combat Direction System on these same classes. He moved to private industry, Vitro and ISA, supporting NAVSEA by reviewing software development specifications and witnessing software testing. He later returned to government service (NAVSEA) as project engineer responsible for the implementation, installation and upgrade of the ASW Control System hardware and software on DD963 and AEGIS Class ships. He then worked with the Combat Systems Office and was responsible for planning and coordinating the land based test and evaluation of combat system software upgrades to carriers, cruisers, and destroyers.

He joined NASA/GSFC in 1989. Here he supported NASA Headquarters Software Management Assurance Program, where he participated in the review of the early versions of the military software development standard, MIL-STD-498, as well as NASA software development and assurance standards and guidebooks.

Mr. Hammer received a B.S. in Electrical Engineering from the University of Maryland. He is a member of the American Society for Quality.

Theodore F. Hammer
GSFC, Code 302
Greenbelt, MD 20771
(301) 286-7475 (voice)
(301) 286-1701 (fax)
thammer@pop300.gsfc.nasa.gov

Lenore L. Huffman

Lenore L. Huffman is a principal engineer with the Software Assurance Technology Center (SATC). Ms. Huffman has more than 14 years of software engineering and quality assurance experience. She is expert in the design, implementation, and execution of data collection, database structures, and metrics reporting and analysis. She is also expert in the design and use of State-Of-The-Art database reporting systems. Ms. Huffman has extensive experience automating Configuration Management and Problem Reporting Systems and adapting their capabilities to satisfy unique project requirements. She has successfully planned, designed, and implemented software quality assurance projects. Prior to joining the SATC, Ms Huffman developed metrics for software at the Space Telescope Institute, and while working at a chemical research center, was awarded with several U.S. patents. Ms Huffman holds a M.B.A. and a B.S.

Lenore L. Huffman
GSFC, Code 300.1, Bld 6
Greenbelt, MD 20771
301-286-0099 (voice)
Lenore.L.Huffman.1@gsfc.nasa.gov

Linda H. Rosenberg, Ph.D.

Dr. Rosenberg is an Engineering Section Head at Unisys Government Systems in Lanham, MD. She is contracted to manage the Software Assurance Technology Center (SATC) through the System Reliability and Safety Office in the Flight Assurance Division at Goddard Space Flight Center, NASA, in Greenbelt, MD. The SATC has four primary responsibilities: Metrics, Standards and Guidance, Assurance tools and techniques, and Outreach programs. Although she oversees all work areas of the SATC, Dr. Rosenberg's area of expertise is metrics. She is responsible for overseeing metric programs to establish a basis for numerical guidelines and standards for software developed at NASA, and to work with project managers to use metrics in the evaluation of the quality of their software. Dr. Rosenberg's work in software metrics outside of NASA includes work with the Joint Logistics Command's efforts to establish a core set of process, product and system metrics with guidelines published in the *Practical Software Measurement*. In addition, Dr. Rosenberg worked with the Software Engineering Institute to develop a risk management course. She is now responsible for risk management training at all NASA centers, and the initiation of software risk management at NASA Goddard. As part of the SATC outreach program, Dr. Rosenberg has presented metrics/quality assurance papers and tutorials at GSFC, and IEEE and ACM local and international conferences. She also reviews for ACM, IEEE and military conferences and journals.

Immediately prior to this assignment, Dr. Rosenberg was an Assistant Professor in the Mathematics/Computer Science Department at Goucher College in Towson, MD. Her responsibilities included the development of upper level computer science courses in accordance with the recommendations of the ACM/IEEE-CS Joint Curriculum Task Force, and the advisor for computer science majors.

Dr. Rosenberg's work has encompassed many areas of Software Engineering. In addition to metrics, she has worked in the areas of hypertext, specification languages, and user interfaces. Dr. Rosenberg holds a Ph.D. in Computer Science from the University of Maryland, an M.E.S. in Computer Science from Loyola College, and a B.S. in Mathematics from Towson State University. She is a member of Electrical and Electronic Engineers (IEEE), the IEEE Computer Society, the Association for Computing Machinery (ACM) and Upsilon Pi Epsilon.

Dr. Linda Rosenberg
GSFC, Code 300.1, Bld 6
Greenbelt, MD 20771
301-286-0087 (voice)
linda.rosenberg@gsfc.nasa.gov

TestFrame

Testing with Action Words

a Quality Approach
to (Automated)
Software Testing

Hans Buwalda

CMG FINANCE B.V.

THE NETHERLANDS

e-mail: hans.buwalda@cmg.nl

agenda

- 1 introduction: testing and quality
- 2 testing with action words
- 3 organisation of the process

testing is necessary

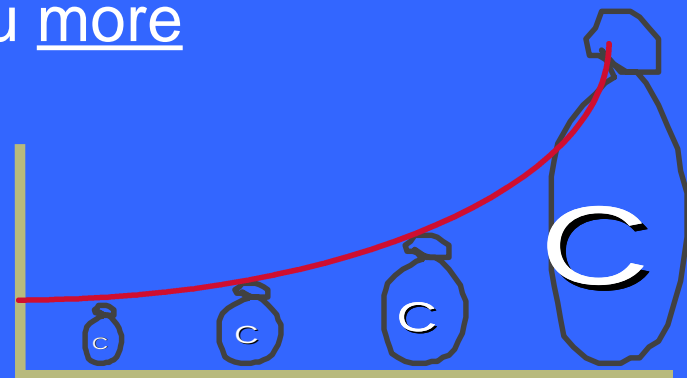
TICK

TICK

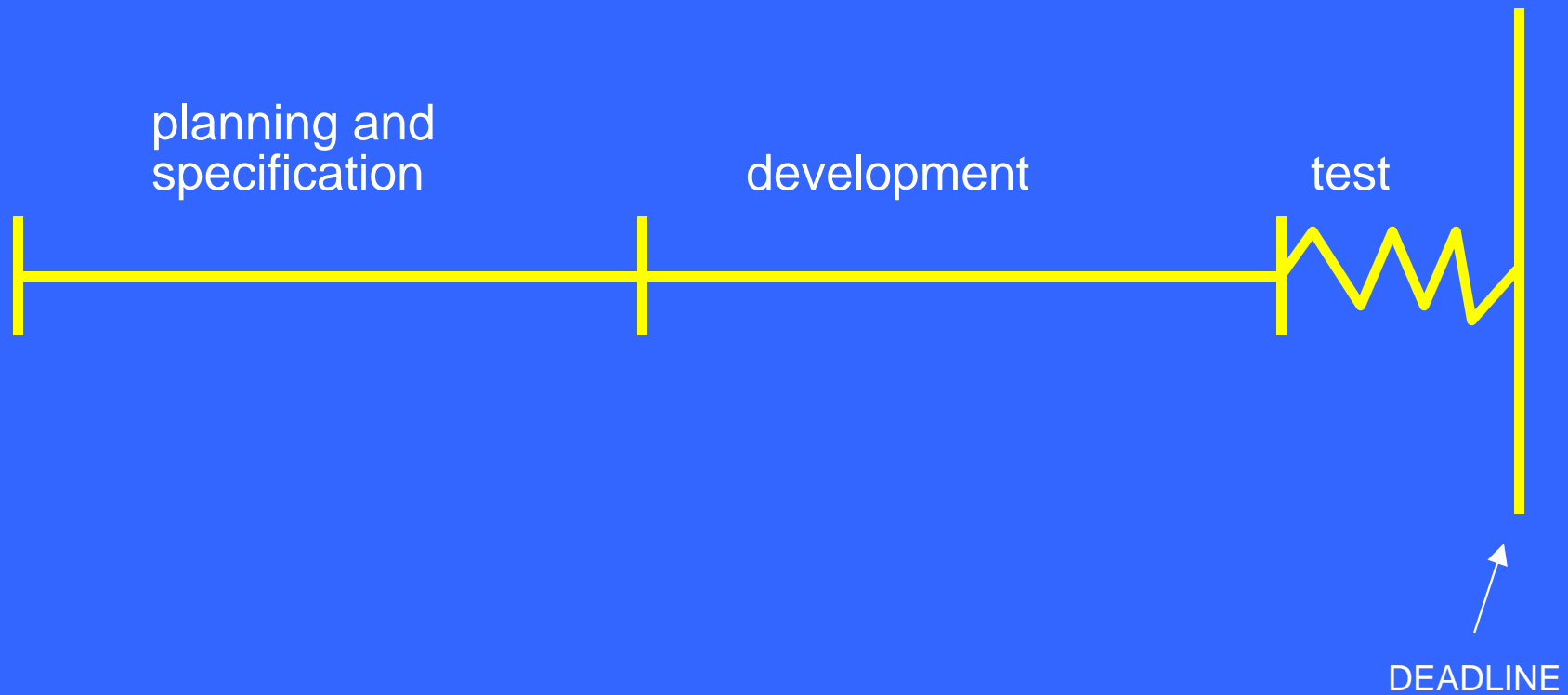
TICK

BOMB

- faults are risks until they are found
- finding them later will cost you more



but testing often gets under pressure



quality and testing

- quality is a general property
- testing is an activity (you can choose not to do it)

quality and testing

- testing is an important instrument to establish the quality
- but you must be able to establish the quality of the testing

common experienced problems with testing

- costly
- time consuming
- boring to do
- difficult to manage:
 - what is the progress
 - what is the quality
- the proper resources (users, specialists) are not available when needed
- often neglected
- automated scripts hard to maintain
- ...

Strategic Context



separation of test development and test execution

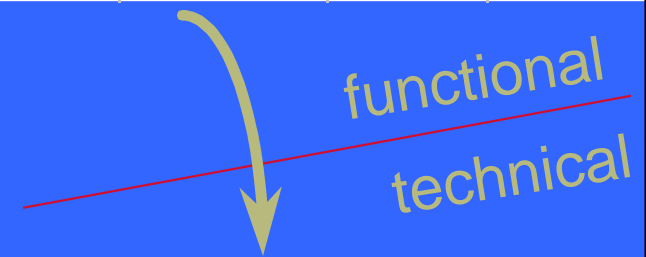
- test development aimed at the production of “clusters”

- input and expected results
- test language with “action words”
- in spreadsheets

A	B	C	D
...			
<i>transfer</i>	Houston	Black	\$210
<i>check balance</i>	Black	\$210	
...			

- automatic execution by a “navigation script”

- written in the script language of the cast tool
- general part: the engine
- specific part: the action words

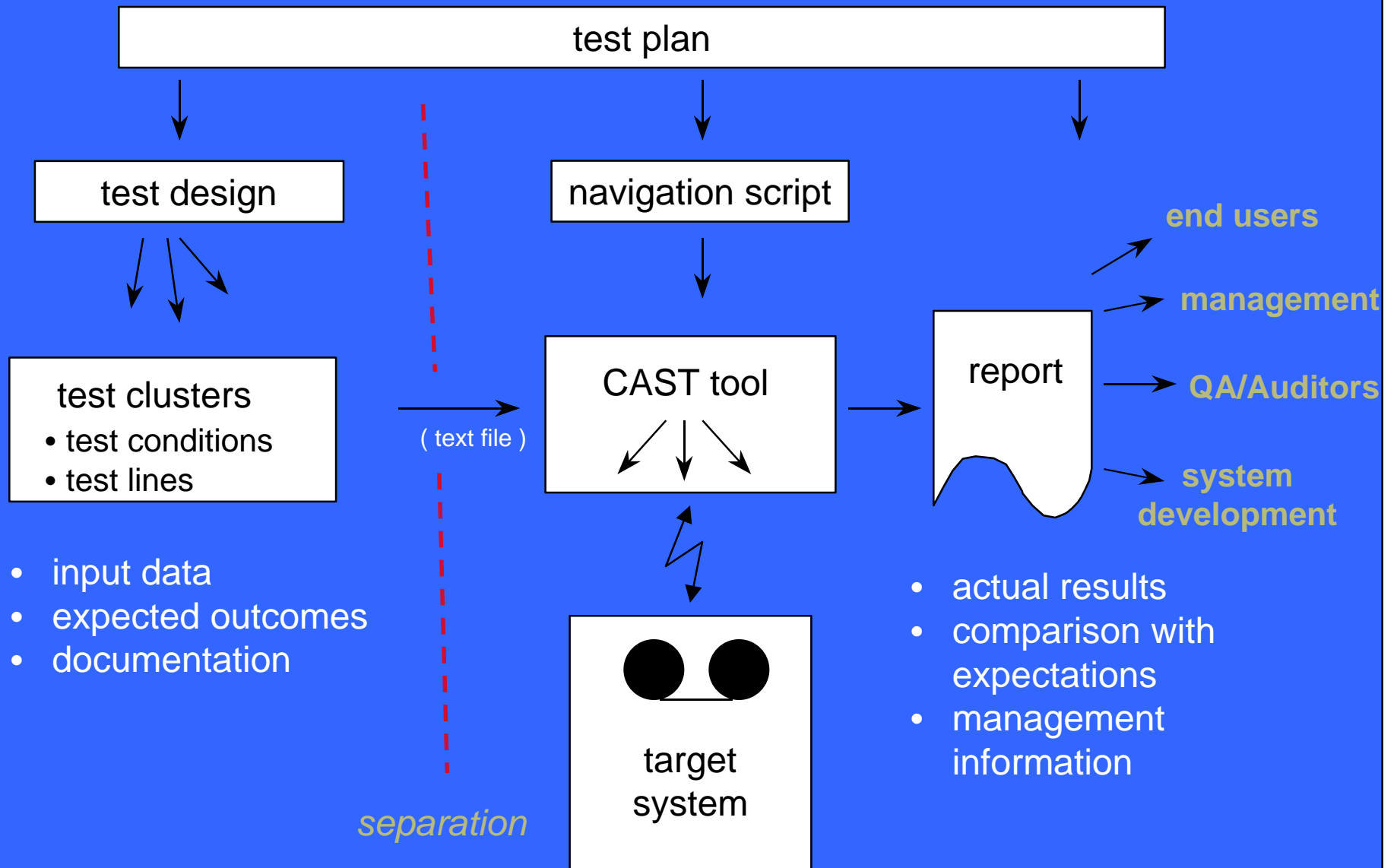


```
case action of
  "transfer": ...
  "check balance": ...
end case
```

organisation

organisation

automation



action words

test data

example of a cluster

<i>cluster</i>	EXAMPLE OF A TEST CLUSTER			
<i>version</i>	1.0			
<i>author</i>	Hans Buwalda			
<i>section</i>	1. Entering clients and balances			
	last name	first name	account nr	balance
<i>enter client</i>	Green	John	458473948	1500
<i>enter client</i>	Wood	Anna	422087596	2100
<i>section</i>	2. Money Transfers			
	from	to	sum	
<i>transfer</i>	458473948	422087596	500	
<i>transfer</i>	422087596	785793025	1201	
<i>section</i>	3. Checking names and numbers			
	account nr	last name	first name	
<i>check name</i>	458473948	Green	John	
<i>check name</i>	422087596	Wood	Anna	
	account nr	sum		
<i>check balance</i>	458473948	1000		
<i>check balance</i>	422087596	1399		

documentary

input

expected output

example of a cluster level report (1)

```
=====
cluster name       :      EXAMPLE OF A CLUSTER
cluster version   :      1.0
cluster author    :      Hans Buwalda

script name       :      Example Navigation Script
script version    :      1.0
script release date :      February 1997

run date and time :      3-03-97 13:39:16
=====
```

```
-----
SECTION 1 - Relation management
-----
```

```
1 ( 6 ): enter client   Green  John   458473948   1500
2 ( 7 ): enter client   Wood   Anna   422087596   2100
```

example of a cluster level report (2)

11 (20):	check name	422087596	Wood	Anna
12 (23):	check balance	458473948	1000	
13 (24):	check balance	422087596	1399	
->FAILED:			1400	

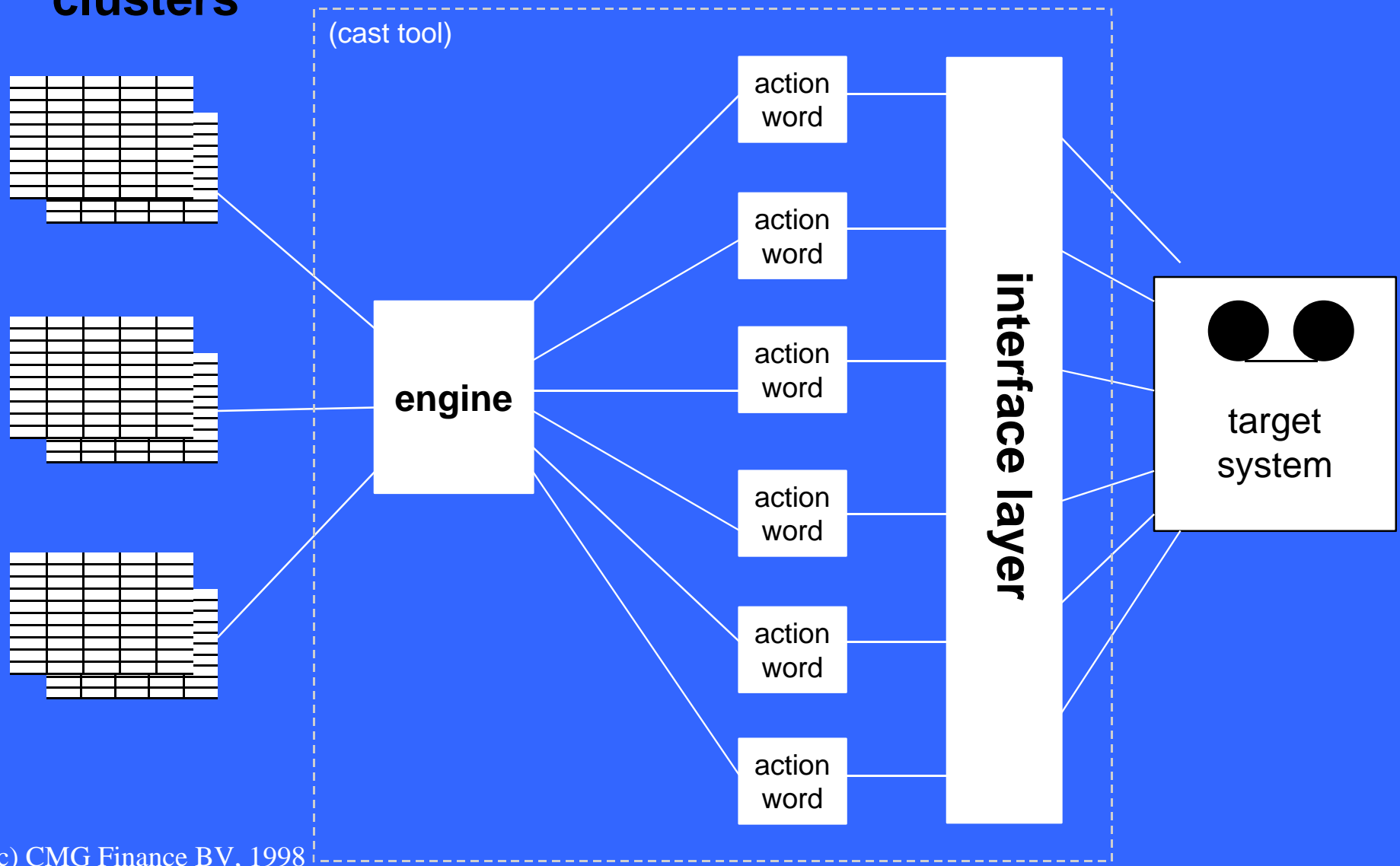
```
=====
end of cluster      :      EXAMPLE OF A CLUSTER
finished at        :      3-03-97 13:39:26
time used          :      15

number of cluster lines :      26
number of test lines   :      13
number of checks      :      10
number passed         :      9
number failed         :      1
percentage passed     :      90 %
```

```
failed at test lines (see above report):
13
=====
```


lay out of the navigation

clusters



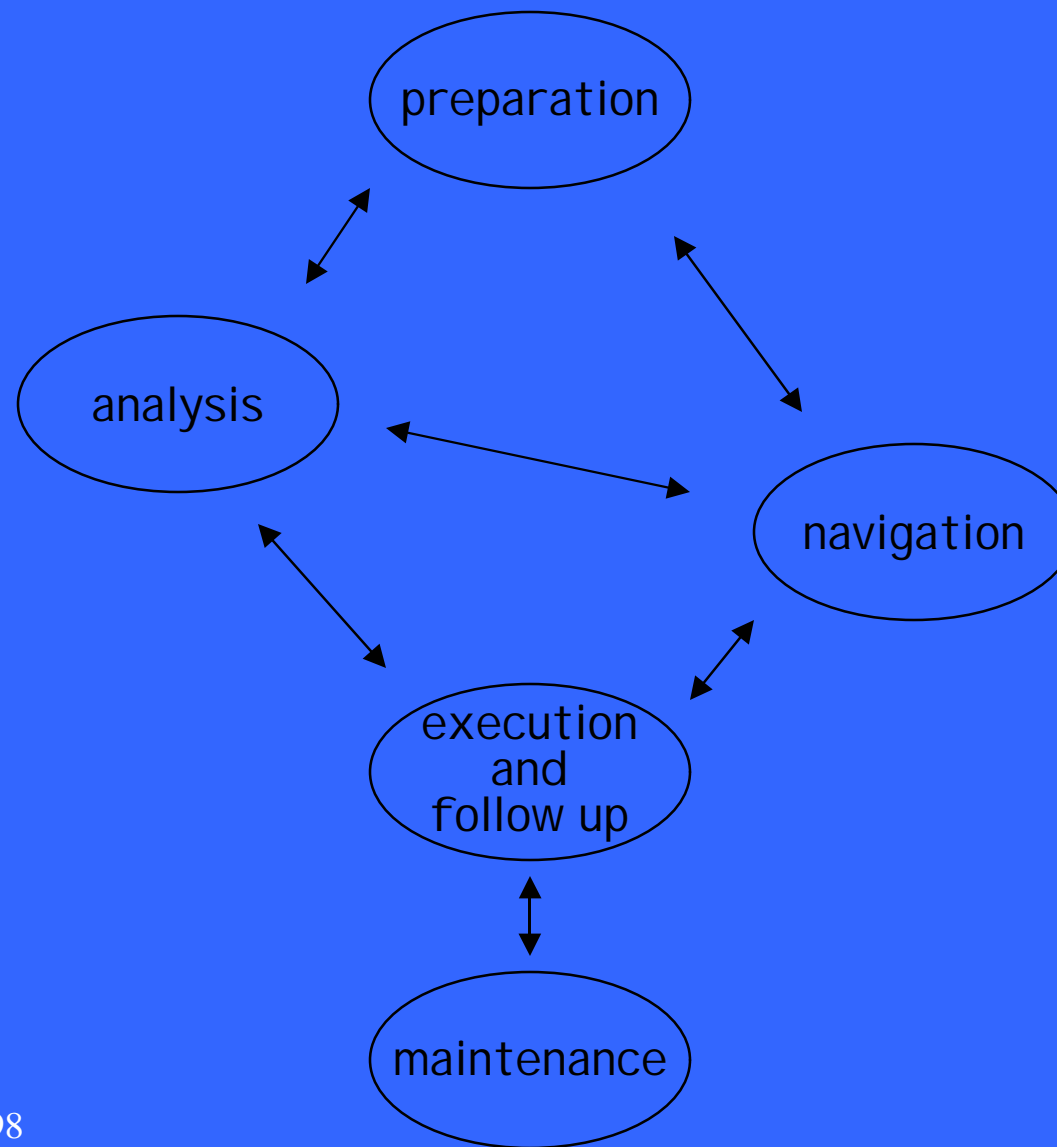
potential advantages of the approach

- less sensitive to target system changes
- better accessible tests
- test development better plannable
- less costs, especially for repeated testing
- higher motivation participants
- better organisational embedding possible:
 - clearer separation of tasks
 - tangible products

the method does not rely on a specific tool

- Winrunner/XRunner
- QA Run
- Hiperstation
- MS/Visual Test
- SQA Teamtest
- ATF
- Autotester
- FEPI
- ...

Activities at Project Level

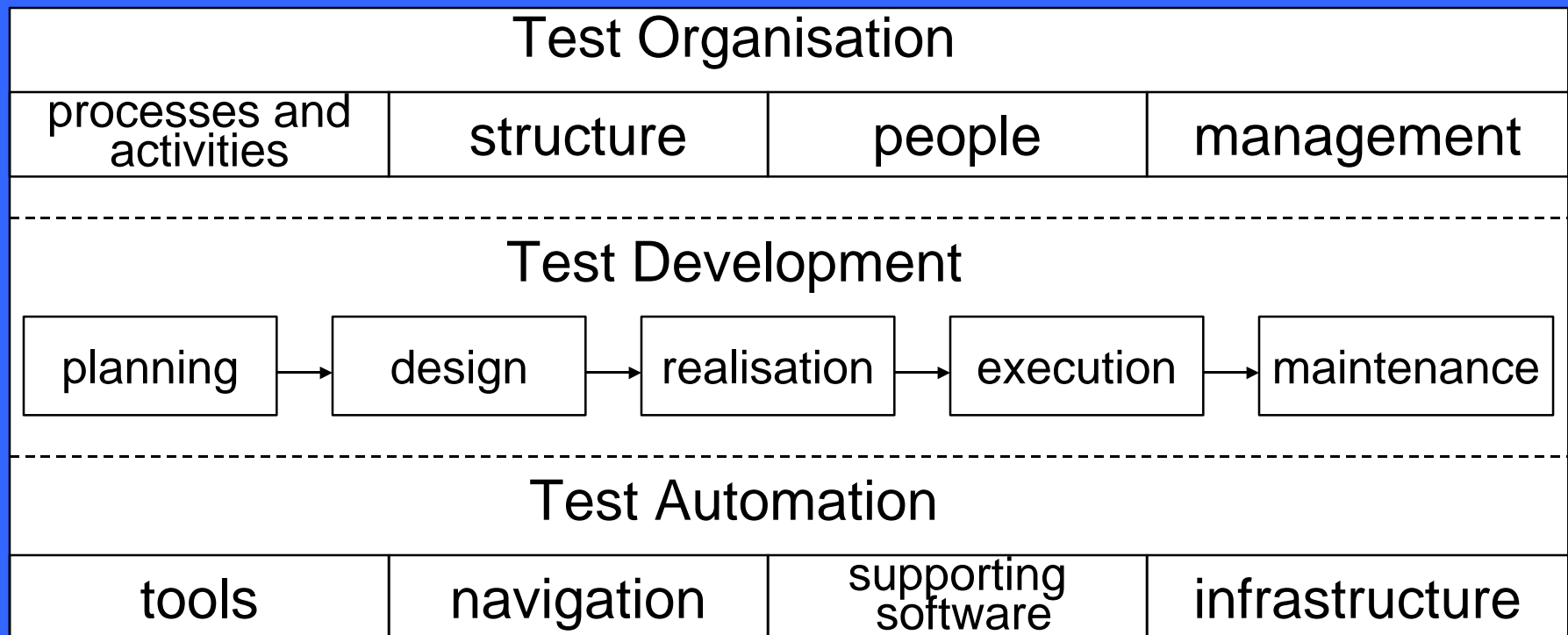


Activities at Organisation Level

- one or more pilots
- training and handbooks
- resourcing (pooling, hiring)
- auditing and reviewing
- r&d
- development of common products
- ...

Implementation

- three dimensions for successful testing:
 - organisation (fitting)
 - test development (structuring)
 - test automation (tooling)



some special applications

- performance and multi user testing
- testing of batch systems
- testing of large scale conversion like year 2000 and euro
- regression testing
- test generation
- test result analysis
- fault tracking
- interface testing

TestFrame

Testing with Action Words

Hans Buwalda

CMG Finance BV
PO Box 133, 1180 AC Amstelveen
the Netherlands
Tel: (31) (20) 50 33 000
Fax: (31) (20) 50 33 022
Email: hans.buwalda@cmg.nl

Abstract

Testing of information systems is a major concern for many organizations. It is often experienced as costly, time consuming, boring to do and difficult to manage. But testing is unavoidable to avoid even greater problems in production.

Automation of the test process with test tools is drawing a lot of attention in the market as a possible relief. These tools are known as CAST: Computer Aided Software Testing. But without a proper approach things will get worse instead of better.

In this paper a method is lined out, working with *action words*, which takes the automation of testing a step further than the commonly used record & playback approach. It is a method for test development, test automation and test organization. Tests are divided into clusters. Every cluster is developed and automated separately.

First the existing approaches for test automation are described, including some of the pitfalls they present. Then the new method is described in general. Next the effect of the method on test automation is described. Finally the application of the method for specific tests is outlined, like batch and performance tests.

1. INTRODUCTION

Automated testing is an area which is getting increasing attention in the industry. In essence it is not a new technique. Programmers have created solutions, ad hoc or structural, to test their products automatically as long as there are computers. Already for many years there are products, on platforms like mainframes, Unix systems, and PCs, which assist in the automation of testing.

The reason for the recent attention is twofold. Firstly client server applications with graphical users interfaces are very complex and contain many aspects that can and should be tested. For example the objects of one client screen of medium complexity contain together typically 2 to 3 thousand different properties, like colors, x and y pixels co-ordinates, visibility and accessibility or underlying database events. There is no way that such amounts of (meta) data can be tested manually.

Secondly PC's are becoming more and more powerful and taking over the role of terminals, getting connected with mini and mainframe platforms using terminal emulators or client server like applications, apart from the applications that run on the PC platforms itself. The PC therefore is becoming an ideal platform for Cast tools, even the applications under test run elsewhere.

An increasing amount of good test products has become available. The core of most of these tools is aimed on test execution. The number of tools for test generation is limited, because this usually does not lead to good tests. Practically every tool set contains modules for test planning, test management and bug tracking. Other features, like performance testing, complete the picture.

In essence two techniques are available for specifying the tests:

- ***Record and playback***

Test actions are carried out by hand and recorded by a test tool, invisible for the tester and the application under test. The record process can be interrupted to introduce checks, specifying that whatever is on the screen or part of the screen at a certain point during the recording should also be there at the same point during the playback.

- ***Test programming***

Most tools contain a script language, which can be used like a normal third or fourth generation programming language, extended with special features for testing like functions for simulating mouse and keyboard events and capturing screen data. The script language can be used to program tests. One of the ways to do this is to take the recorded scripts and embedding them in a loop, which is reading records from a data file. This approach is often referred to as "data driven record and playback".

Both methods, if used prudently, can lead to improvements of the test process, especially when tests are to be repeated more than once. Still pitfalls can arise. In essence the record and playback is in fact automation of the existing manual process. A good comparison is that of the first cars, which looked like carriages with a motor attached to the place of the horses. It is a

first step but not an optimal design. The best use for record & playback is for small ad hoc tests that have to be repeated only a few times. Test programming is an effective doubling of the programming effort, the system design has to be repeated in the test design. The logic of the test is hidden in the script code of a test tool. In both approaches there is a lack of flexibility: maintenance on the underlying system, like changing a menu structure or the location of a result on a screen, means an substantial effort to keep the tests running.

A second potential problem with existing approaches for test automation is accessibility of both the tests and the test results. Because actions and data are mixed and represented in a technical form, it is difficult to understand what exactly is tested, especially for a non-technical person like an end-user or an auditor. It is therefore hard to get a commitment for a system using automated tests.

More practical considerations are the difficulties to start early with the test preparations, one has to wait until there is a working system, and, for the record & playback method, the applicability on only on-line systems.

2. DESCRIPTION OF THE ACTION WORD APPROACH

The tests are not registered in the test tool, neither as record playback scripts, nor as test programs. Instead the tests are put separately in spreadsheets. These spreadsheets are called *test clusters*. To implement this approach a *test language* is introduced, specifying actions to be taken and data to take that action with. The data can be either test input or (expected) test outcomes. The actions are specified as *action words*, short commands to the test tool.

An example:

	last	first	date of birth	
<i>enter client</i>	Buwalda	Hans	2-jun-57	...
...				
<i>check age</i>	40			

Annotations for the table:

- A dotted line points from the text "this row will be skipped when the test is executed" to the first row of the table.
- A dotted line points from the text "input data" to the "..." cell in the first row.
- A dotted line points from the text "action words" to the "*enter client*" and "*check age*" cells.
- A dotted line points from the text "expected result" to the "40" cell in the third row.

This example describes a part of a functional acceptance test for an imaginary client management system. The action words used are “enter client” and “check age”. The first one enters some client data, the second one specifies an expected outcome, in this case the age.

The action words are usually specific for the application that is tested. So there will be other action words for a stock trading system and for a mortgage system. The first line in this example contains column headers, meant to enhance the readability. Because there is no action word in front of them, they are skipped during test execution. Lay out properties like the italic printing of the action words have also no effect on the test itself.

In order to process the tests from the test cluster into the test tool, the clusters are first exported to a text file. This is done in the form of “tab delimited ASCII”, meaning that the fields

in the spreadsheet are separated by tab characters. This is a standard export format of popular spreadsheet programs.

The test automation is regarded as a separate activity, apart from the test design. To interpret and execute the commands in the test cluster a special script is written, so called *navigation script*. This navigation script is usually written in the script language of a standard test tool. Most test tools in the market have a script language build in powerful enough to make implementation of a navigation script possible.

The navigation script consists of several components, some of which are general, others must be specifically written for the application that is being tested. The general functions read the test lines from the tab separated ASCII text file, which was exported from the test cluster. The lines are interpreted one by one. The first field, the action word, is used to call a function for that action word. The action word functions are specific to the application.

In the above mentioned example there will be specific procedures for the action words used. The procedure for “enter client” will go to a client screen and fill all fields in the right order. This is not necessarily the order used in the test cluster. Even the number of fields is not necessarily the same. The navigation procedure can replace not specified fields with relevant default values. After entering the fields the navigation will close the entry screen by using enter or any other means necessary in the system under test.

The procedure for second action word in the example, “check age”, will follow a similar path. It will select a screen where the value for the age can be found and capture this value from the screen. Next it will compare this value to the expected value which was specified, in this case 39. The result, “pass” or “fail”, will be written into the *test report*. If it is not possible to capture a value from the screen other means can be used to obtain like SQL queries or internal API calls.

The report for our example will look something like this:

```
.....  
cluster:          example  
version:         1.0  
date:           December 1st, 1997  
.....  
.....  
21      enter client Buwalda  Hans      2-jun-57  ...  
.....  
35      check age 39  
        check of type:      age  
        expected value:    39  
        recorded value:    38  
        result:            FAILED  
.....  
.....  
number of checks: 257  
number passed:   252      percentage passed: 98%  
number failed:   5        lines: 10, 35, 52, 134, 201  
.....
```

report header, containing general info

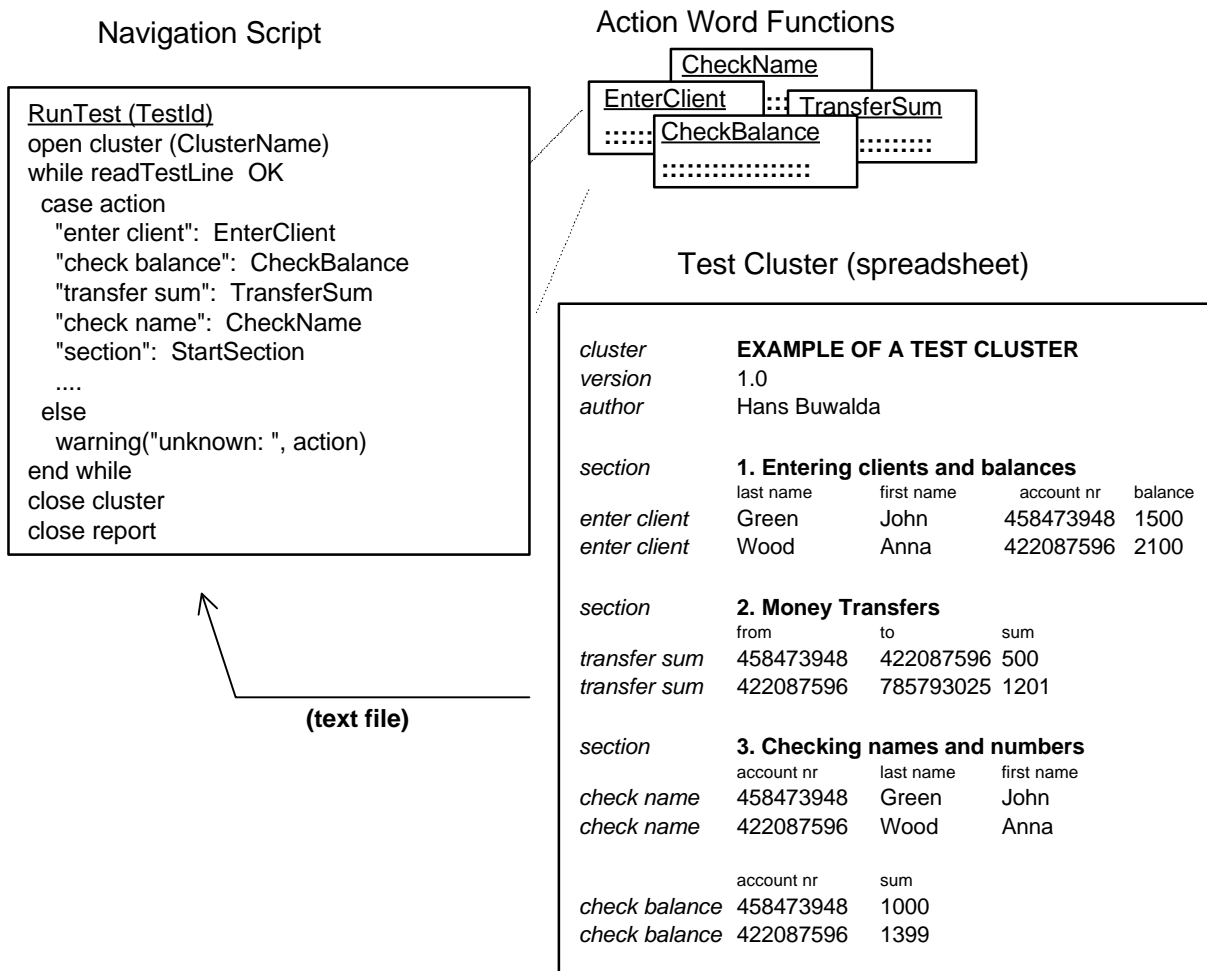
detailed results of input and checks (on-line number in the test cluster)

summary of the test run

The report consists of a header, a detail part and a summary. What is exactly listed in the report depends on the project. In our example the test has failed. The fail is marked, with the expected and recorded value, and in the summary the fail is marked again with a line number so that it can be easily found back for analysis.

Test results reported as “failed” are input for further analysis. They do not necessarily mean that the system is wrong. Like with other test methods also the tests themselves can contain errors. The fails must be regarded as signals.

Using a second example, a test on small banking system called Minibank, the cluster, navigation script and report are related as can be seen in the following picture.



The lines in the cluster are interpreted one by one by the navigation script. Every action word is carried out by a action word function in the navigation script.

The test report, produced under control of the navigation script, contains besides the comparison results other data relevant to assess the test. It will usually contain a print of the test lines interpreted, version information in the header and a summary of the test results. It is also pos-

sible to report the test results in an alternative form, for example to collect them into a database or into a specialized system for fault tracking.

In the above example the navigation was displayed as one loop reading the test lines and interpreting them. In practice a navigation script consists of several layers, dealing with tasks like interpretation of test lines, execution of action words, reporting and interfacing with the target system (for example through a GUI interface or a terminal emulator).

3. APPLICABILITY ON OTHER TESTS

The example in the last paragraph dealt mainly with *acceptance testing*. The method is also applicable for a number of other kinds of tests.

Tests earlier in the project, like the *module test*, the *system test* and *integration test*, can be automated with the described method as well. There are two considerations to make. Firstly the tests usually have similar input cases, but want to test also intermediate results, for example the contents of a database not accessible for an end user. Introducing additional action words for that purpose can do this. The navigation script will then access the relevant data using an appropriate means like a terminal emulator. It is even thinkable to access a debug tool, in which case expected values can be specified for variables within a program.

A second consideration is the moment in the project that the tests are necessary. Especially an early program test comes at a time when there are no implemented action words yet. Also it can be too much work to implement an action for every program in a system. In that case it is possible to use so-called “low level action words”. These words, like “push button” or “select menu item”, have the relevant button to push or menu item to select as an argument in the cluster. Although this has a great disadvantage of being more sensitive to system changes it can be a solution for tests that are only used once.

Apart from on-line systems also *batch systems* can be tested. This is a main difference with record & playback testing. The test clusters for a batch system are similar to those for an on-line system. They start with a number of test lines with input data. The action words depict in which tables or files the test data should go before the batch is started. For large records only those fields, which are relevant for the test, are specified, the action word will fill in the rest of the fields with relevant defaults. The input lines are followed by a special action word, like "start overnight", that will trigger the start of the batch process that has to be tested. Then the output lines are specified, with the action words specifying which tables or reports contain the data which to compare it with.

The processing of batch test can still be done with a test tool on a PC. The test tool will, under control of the navigation script, route the input data in the correct tables or files. This can either be done by using on-line screens (for example with the use of a 3270 emulator) or collecting in a local file which is uploaded. The test tool, acting as a “director” of the process, will start the batch process and, after it has detected that the process is ready, will test the outcomes. This can be done by either accessing the outcome record by record or by collecting the expected values in reference files and comparing these with a matching program either local or on the mainframe. There are also test tools available on the mainframe itself, which can be equally well used with the method.

A special application of the method is for testing *performance*. In these tests the focus is on the behavior of a system under a specified load. Figures that are measured are typically response times and the usage of system components. Those values can be compared to specified demands. This can be done in the test cluster introducing action words like "response time 10", meaning that a time of 10 seconds is the maximum allowed for the system to respond since the latest system entry. The test fails if the actual time exceeds these 10 seconds, in which case the actual time is reported.

Similar solutions are devised for load generation. A typical form is "generate load 10 order-entry", meaning that a cluster "order-entry" should be executed simulating 10 simultaneous users. This can be done on physical machines or as processes in a multi-tasking environment. The action word "generate load" does not do much more than starting the 10 processes. Within the earlier mentioned cluster "order-entry" other performance issues are specified with proper action words, for instance measuring response times and synchronization with other processes.

The method was developed in 1994 and used in many projects since then. In present days a large part of the work with the method is in *millennium projects* and the *euro currency conversion*. The method can be used to generate test cases efficiently, but also to automate tasks like the "time travel" which is necessary for many of the millennium tests.

4. EFFECTS OF THE METHOD

The method described in the previous paragraphs is used in a growing number of large and small projects, both for on-line and batch systems. The method has a number of effects on the test automation process.

The first effect is on flexibility. A test set (test clusters and navigation script) is much less sensitive to maintenance than for example record & playback scripts. When a change is made in the underlying system, this will usually lead to a change in the tests but this change is limited. Most of the times a change has only consequences for the navigation. For example a change in the menu structure of a system or the place of an output result means a change to the relevant action word procedures. By changing only a limited number of action word procedure a much larger set of tests will run again on the new version of the system.

When there is a change in the business processes implemented by the system, the change will effect the test cluster. For example a change in the tax laws will have an effect on a payroll system and therefore on the clusters testing that payroll system. But these changes are logical, a direct consequence of the functional change and therefore usually very straightforward to implement.

A second effect is on the accessibility of the test clusters and the reports. Although they are direct input for an automated test process the test clusters can be designed in a way that also non-IT people can understand them. It is even possible to let end users directly prepare the test cases in the spreadsheets.

A next effect concerns the planning. The preparation of the test clusters can start in an early stage in a project, without knowing all functional details of the system. Especially the business oriented test cases for functions like payrolls, life insurance's or mortgage calculations can be produced when a system is still in the definition phase. The navigation script can be developed when a system is in its detail design and early building phases.

There is an effect in cost effectiveness. The implementation of the navigation script is an extra activity in a project. This activity must be compared with the manual testing effort for the first test. An important difference in this respect with the record & playback method is that also the first run of the tests is automatic, there is no recording phase. Usually the cost for the navigation script is in the same order as the saving on the first test run. Of course the larger gain can be expected for repeated tests, either tests of new releases in the project or later in the maintenance phase of the system.

Like other automated test methods there is an implicit effect on quality. Because testing is done automatically it is very persistent. Also it is possible to always run all tests, catching the notorious unexpected faults resulting from maintenance in other parts of a system.

The method gives a basis to organize and manage the test process better. Products like the test clusters and the report are tangible. It is easy to show what has been tested and what the results are. Also the method gives a proper separation of tasks:

- test analysis, the process of developing the tests and assessing their results
- test navigation, the actual automation of the tests, which in practice turns out to be a different area, with different skills needed (people doing this are called the “navigators”)
- test organization, at project level dealing with topics like who is doing what at which time and above project level co-ordination of competence and resources and embedding of tasks like cluster and script maintenance in the organization

Although almost all tests are automated this is not an essential component of the method. There have been projects where tests were developed in the form of cluster with action words, but not automated. In stead they were executed by hand. Test clusters have turned out to be a efficient way of documenting tests with or without the automation by a test tool and a navigation script. The decision of automating can even be made for every cluster separately.

Last but not least in the projects so far an increase in *motivation* for testing has been observed. Reasons for that are that the dull parts of the work are done by the machine and that the separation of test design and navigation make it possible for people to work on the job they like most and they can do best, either the more business oriented test design or the more technically oriented navigation.

5. OTHER USES THAN TESTING

Although most projects in which the method is applied deal with automated testing, this is not essential. There are already uses for other purposes and more uses are imaginable in the future.

To understand this it is necessary to look closer at the automation part of the method. The essence is that the action words form a highly adaptable alternative interface to a system, one could refer to it as a “level of indirection”, a popular concept in programming. Using the action words pre-defined commands can be given to a system. If the interface of a system changes, it is enough to change the navigation level instead of having to change the command lines.

A typical alternative application of this principle is *initial loading*. Clusters can be specified with initial values for tables, system variables, user authorizations and other data that are necessary for initializing a system in an environment. This can be a test environment, a training environment or the actual production environment.

A similar approach is possible for *data conversion*. Data can be extracted from existing systems and entered using the action words, which already exist for the testing. In this way the necessity of developing specialized conversion functions can be avoided. This is especially applicable for data with no high volume like a table with currency codes.

6. CONCLUSION

In this paper a method is described for automated testing of systems. The design of the tests is strictly separated from the development of the script to automate their execution. The main benefits of the approach are flexibility, manageability, cost effectiveness, quality and motivation.

This paper has only given a first introduction of the method. There are many aspects to deal with in the actual implementation of the method. It is therefore recommended to start on a small scale. Also it is important to notice that the method, although potentially useful, is not a magic potion. Testing is a complex difficult activity that should never be underestimated, with or without method outlined in this paper.

Software Defect Analysis: Real World Testing

A Simple Model for Test Process Defect

Software Quality Engineer
Hewlett Packard Company
Boise, Idaho LaserJet Division
Quality Engineering Department

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Introduction

- Turning hindsight into foresight.
 - Drives need for Failure Analysis (defect categorization).
Helps create plan for Software Development Process Improvement on next project.

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Introduction

- Model 1 - Software Development Defect Categorization
 - A “common” practice.
 - The model proposed in this paper extends the usefulness of software development defect categorization to Software Testing.

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Introduction

- Model 2 - Software Testing Process Defect Categorization
 - Not a “common” practice.
 - This paper introduces a model for categorizing software testing

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Actions

- Present and discuss the models.
- Apply the models to software testing projects.
- Provide input to the author.

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Benefits

- Use Software Development defect trends to help focus Software Testing.
- Determine testing specific defect patterns. Take steps to

©Hewlett Packard Company, 1998

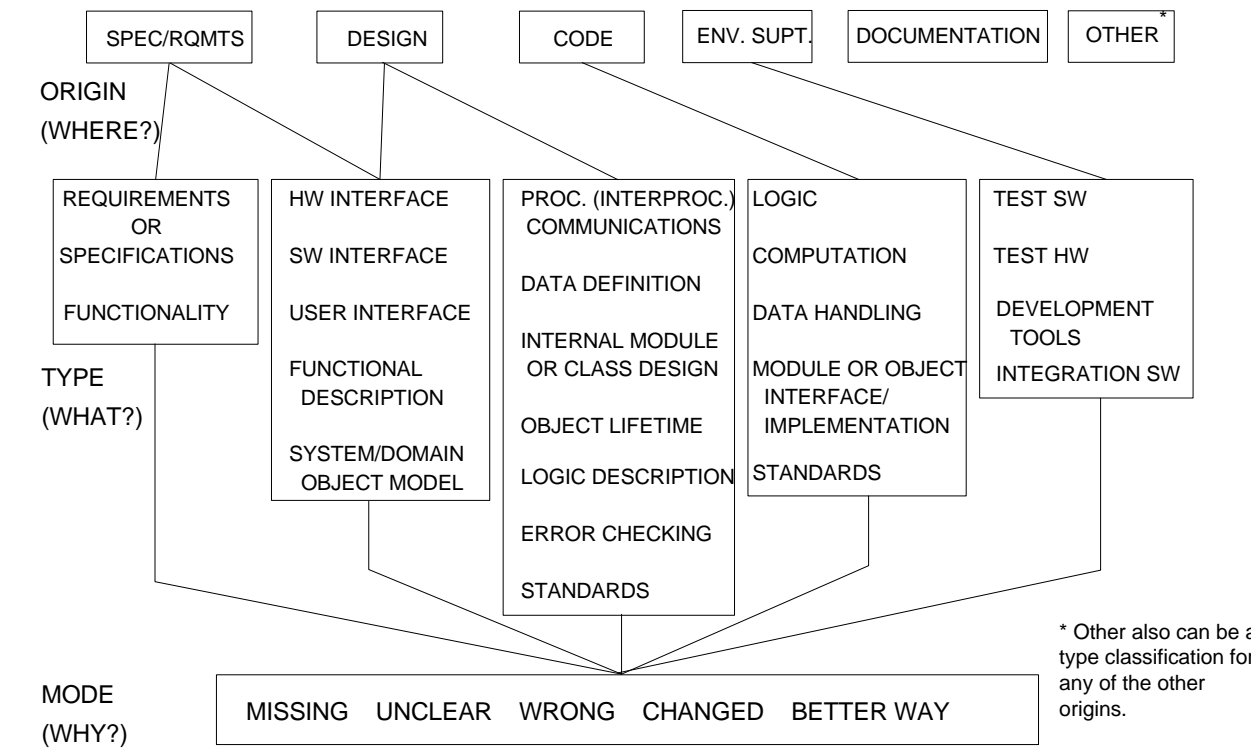
Quality Week Europe, 1998

Part 1 - What is Discussed

- Current HP Model. Results from
- Introduction of Model 1- Software Testing Focus Based on Defect Trends.
- Application of Model 1 to a product.

Current HP Software Defect Categorization Model

CATEGORIZATION OF SOURCES OF SOFTWARE DEFECTS

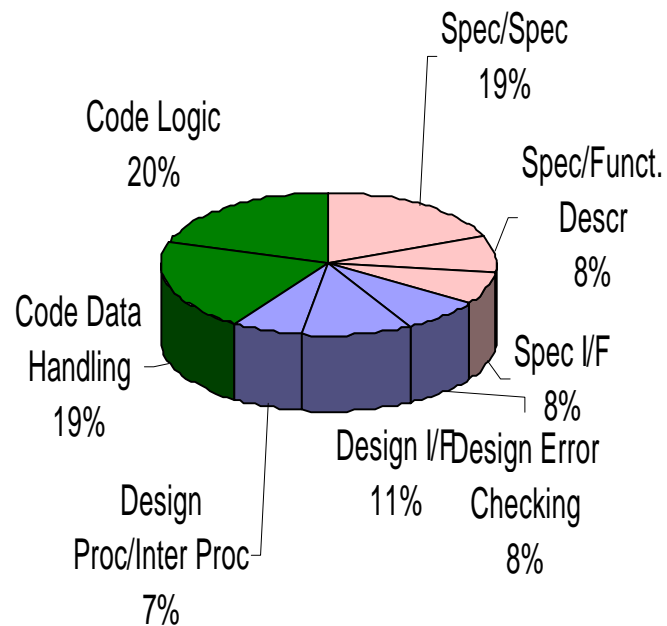


© 1992 Prentice-Hall

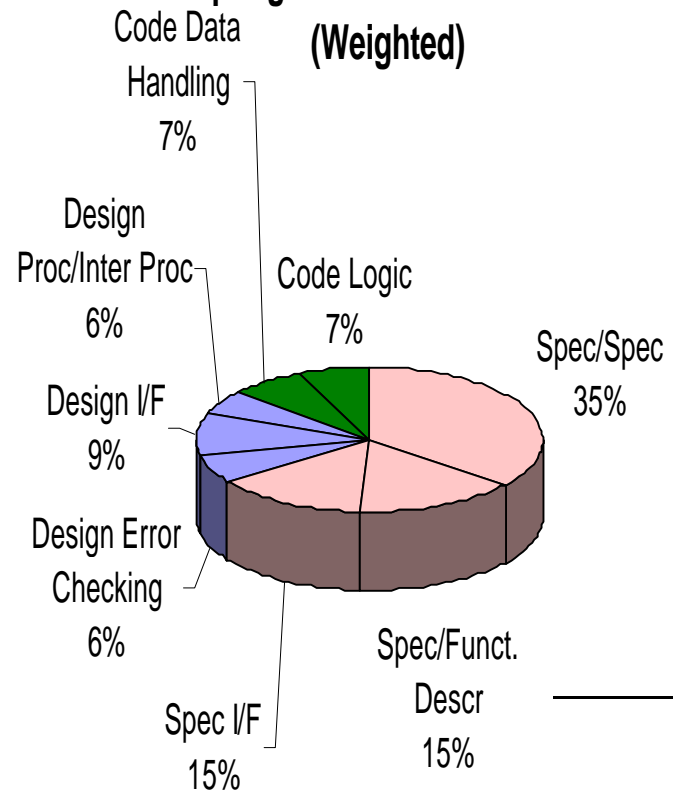


Results based on HP Model

**Top Eight Sources of Defects
(Actual)**

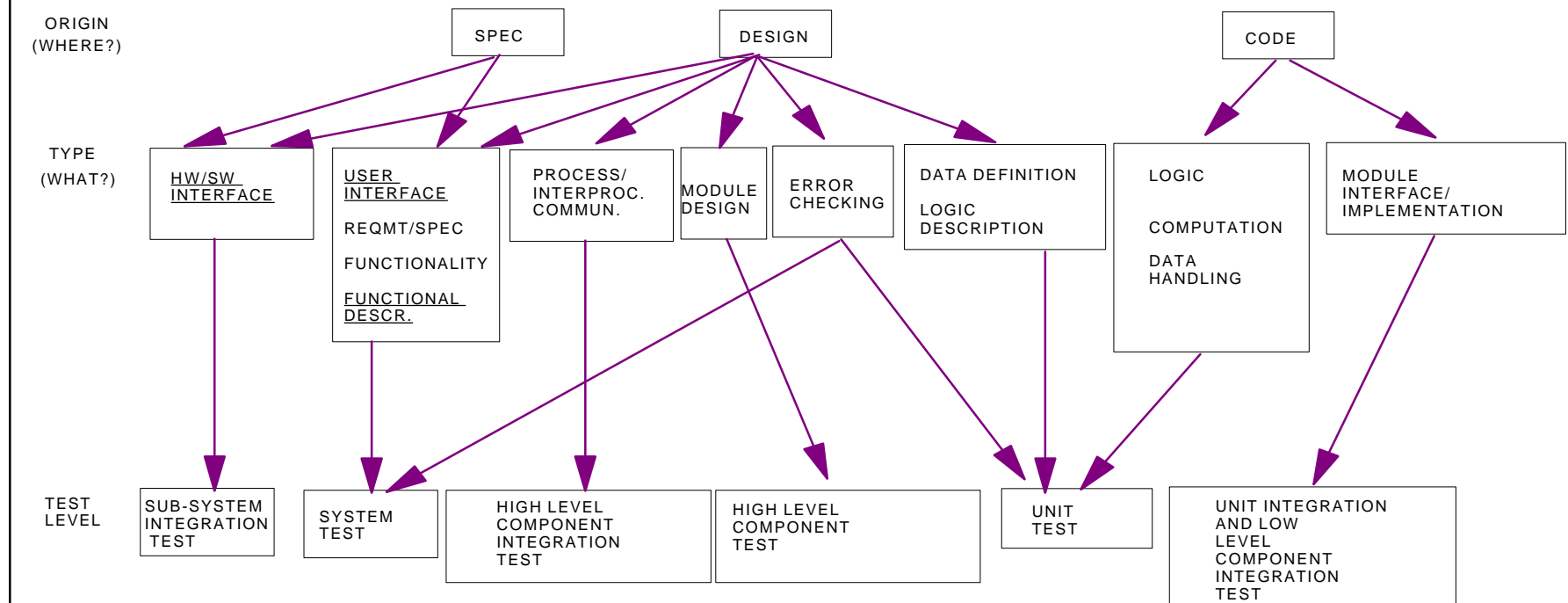


**Top Eight Sources of Defects
(Weighted)**



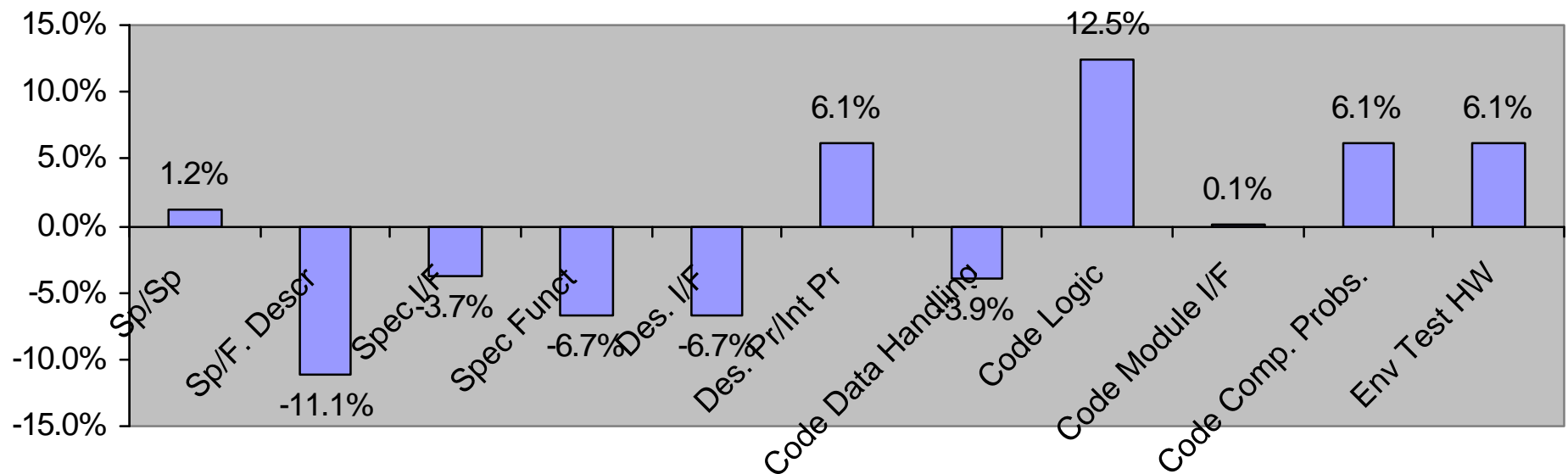
Model 1 - From Defect Trends to Software Testing Implications

Software Testing Focus Based on Sources of Defects



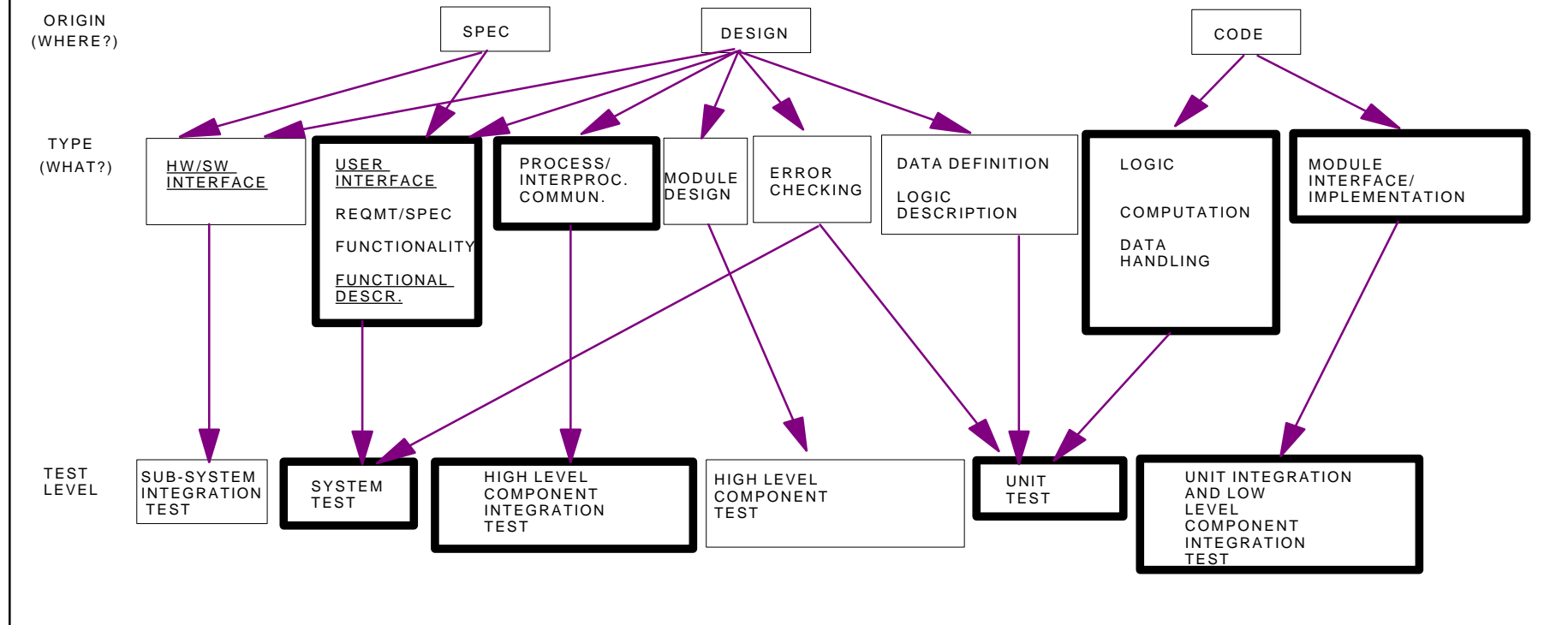
Applying Model 1 to a Product

**Percent Defect Changes
One Project Compared to its Follow-on**



Applying Model 1 to a Product

Software Testing Focus Based on Sources of Defects



Part 2 - What is Discussed

- Software Testing Defect Areas.
- Introduction of Model 2 - Software Testing Defect Categorization.
- What should be in place to apply Model 2 to a product.
- Test Defect Weighting.

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Software Testing Defect Areas

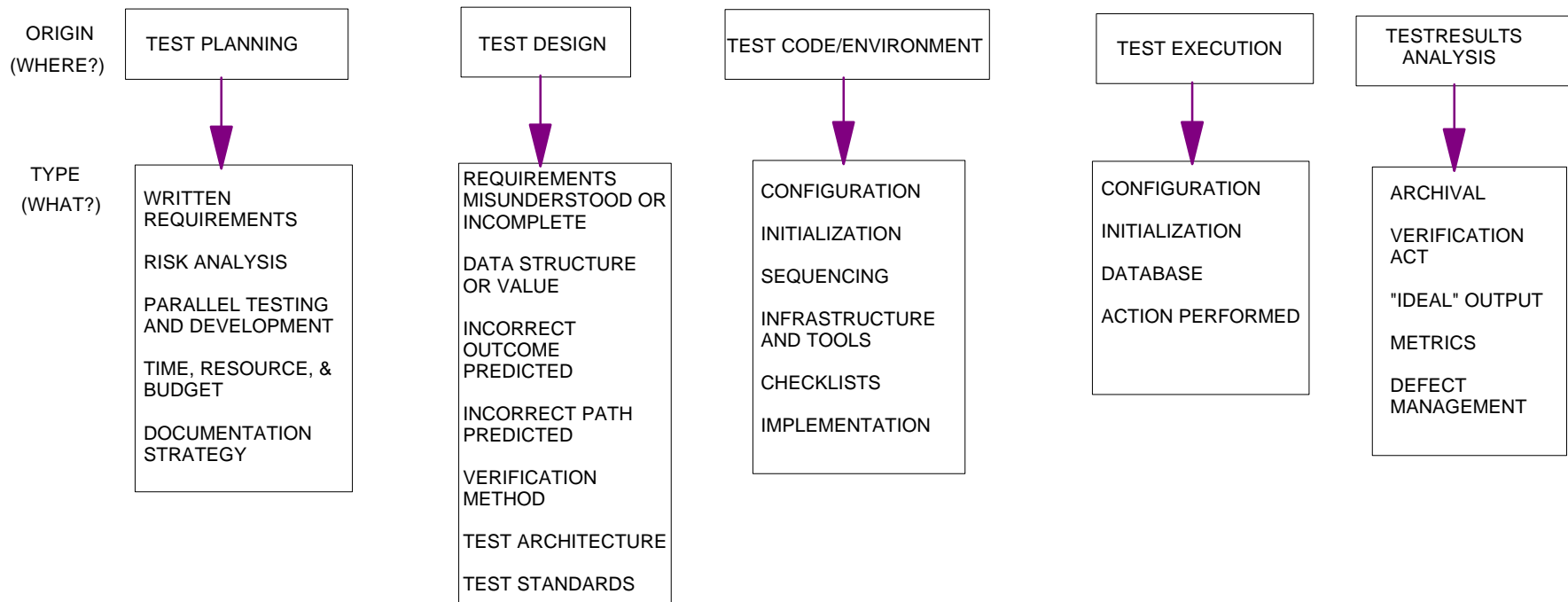
- Test Process Defects - Errors in strategy and/or procedures.
- Test Management Defects - Policies and/or resources not in
- Test Product Defects - Errors resulting in a change to the test

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Model 2 - Classifying Software

Software Testing Defect Categorization

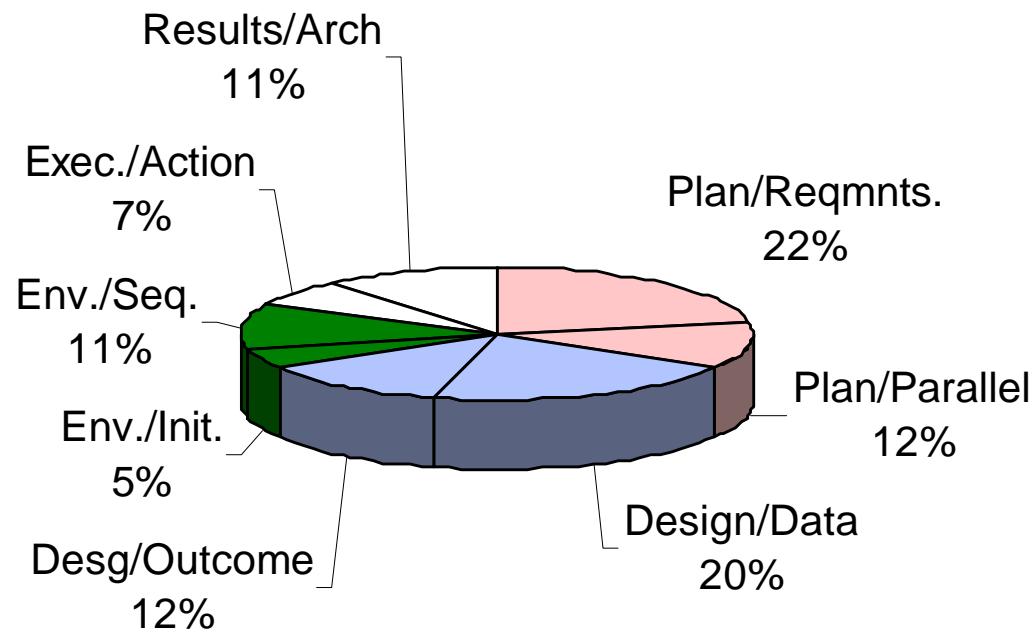


Necessary Elements to apply Model 2

- Distinction between Software Testing and Development Defects.
- Test Engineers log Testing Defects as such.
- Test Team commits to applying model after every project iteration.

Example Results

Top Eight Sources of Software Testing Defects



Test Defect Weighting

- All testing defects not created
- Test Planning and Design defects more difficult to detect
- Weighting factors vary from one testing project to the next.

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Next Steps

- Apply models to HP projects.
- Collect suggestions from experiences others have in using the models.
- Update the models based on input.

©Hewlett Packard Company, 1998

Quality Week Europe, 1998

Software Defect Analysis: Real World Testing Implications & A Simple Model for Test Process Defect Analysis

Abstract

This paper will present two models. The first model extends the current Hewlett Packard model for the “Categorization of Sources of Software Defects”. This expansion has to do with focusing software testing efforts based on software development defect trends. The second model allows software test professionals to categorize software testing defects. It is hoped that through the use of these two models, software test professionals will have the ability to better analyze software product and testing defects, enabling them to implement action items that will improve software testing.

Introduction

Any observer of human behavior will acknowledge that hindsight is almost always better than foresight. The challenge with software testing is to adapt the hindsight learned on one project into foresight on the next. Results from the categorization and analysis of software development defects after a project is finished are often used as a guide to implement software development process improvements on the next project. What is often ignored is the benefit that software development defect analysis has in relation to software testing. Since one purpose of software testing is to find bugs, examining software development defect patterns and trends, and then testing in those areas, will help focus the efforts of software test engineers. The first model introduced in this paper enable mapping software development defect trends to specific areas of software testing.

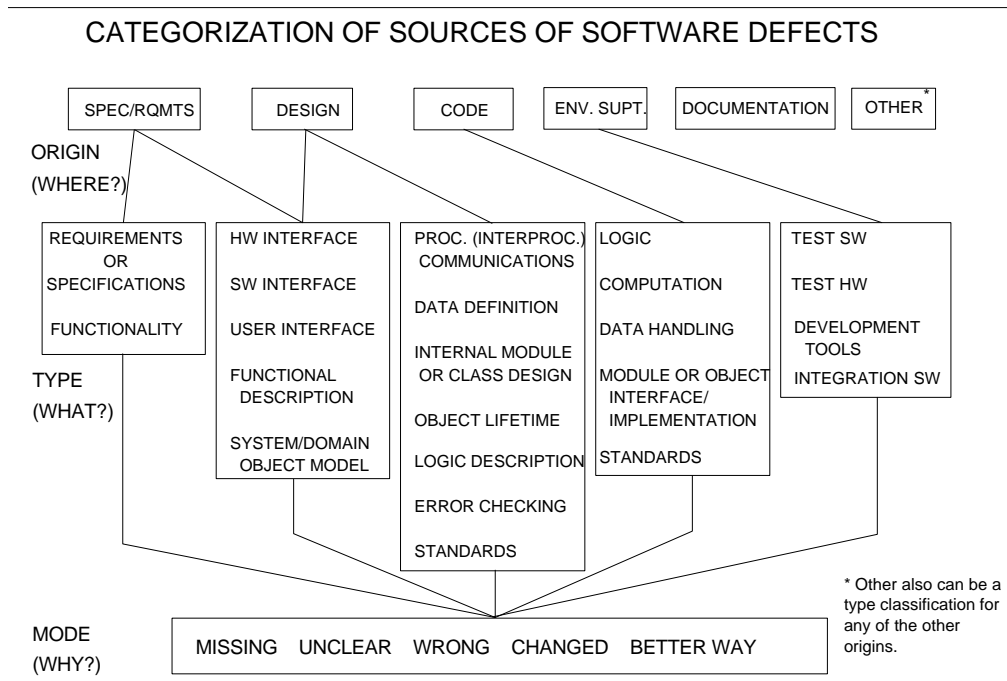
The analysis of defect trends in software development is a common practice in the software industry. What is equally important, but sometimes overlooked, is the examination of software testing defect trends. In contrast to multiple models available for categorizing software development defects, there are only a few models available, and proven useful, for categorizing software test defects. The second part of this paper will introduce a model, and the elements that should exist in a software testing organization, to categorize software testing defects.

Real World Testing Implications of SW Development Defect Analysis Data

This section of the paper will:

1. Introduce the Hewlett Packard model for categorizing defects.
2. Present defect categorization data from seven Hewlett Packard projects.
3. Explain the first model entitled “Software Testing Focus Based on Sources of Defects”.
4. Apply the model to a software development project and its follow-on.

The following model (Figure 1-1), developed by the Hewlett Packard Software Metrics Council in 1986¹, was used to categorize the defects represented in the pie charts below (Figures 1-2 & 1-3).

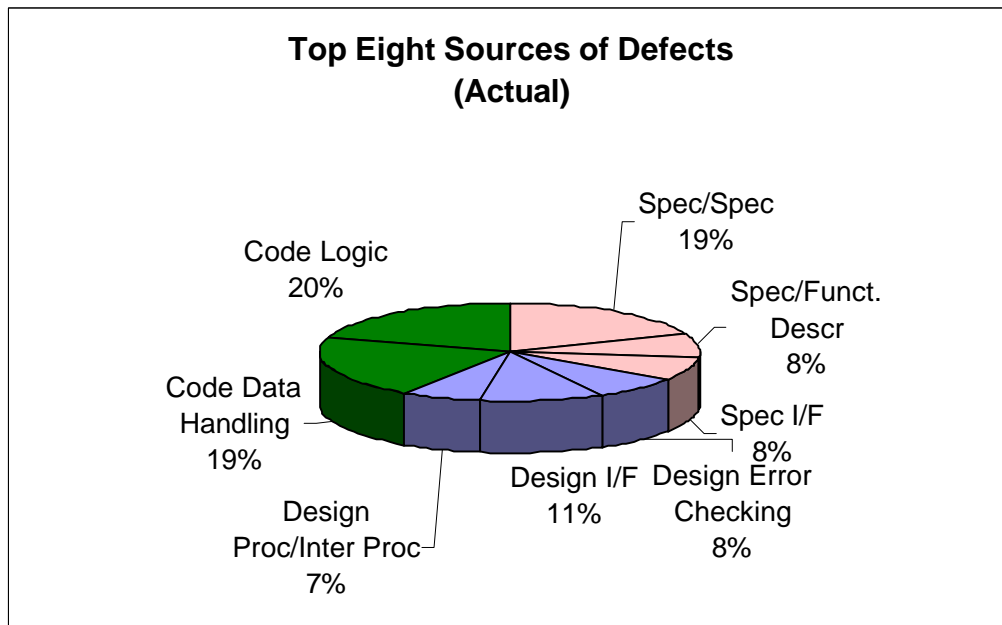


© 1992 Prentice-Hall

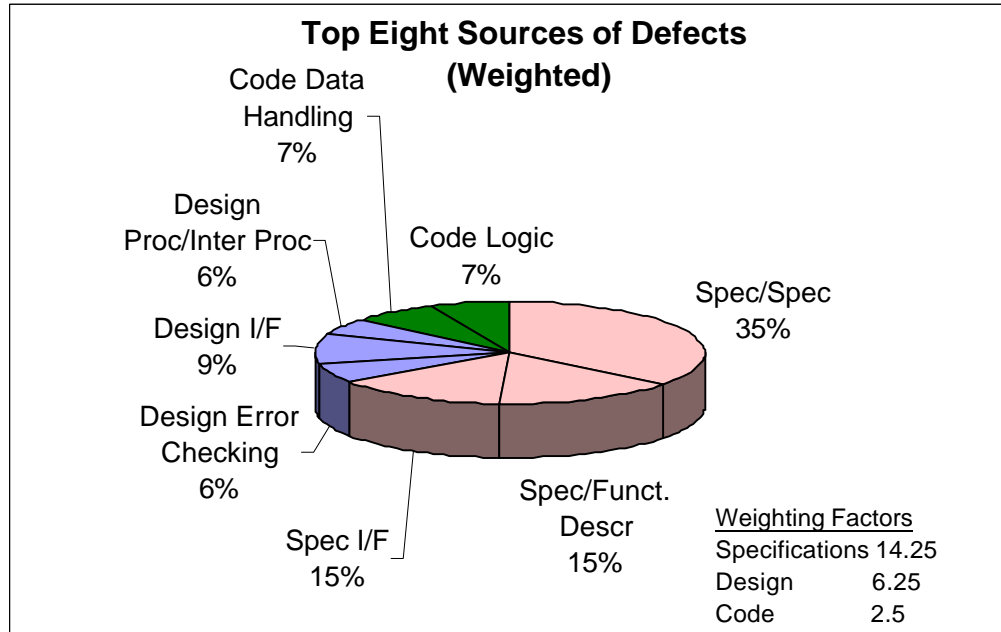


(FIGURE 1-1)

Figures 1-2 & 1-3 show the average, actual and weighted, defect trends from seven Hewlett Packard projects.



(FIGURE 1-2)



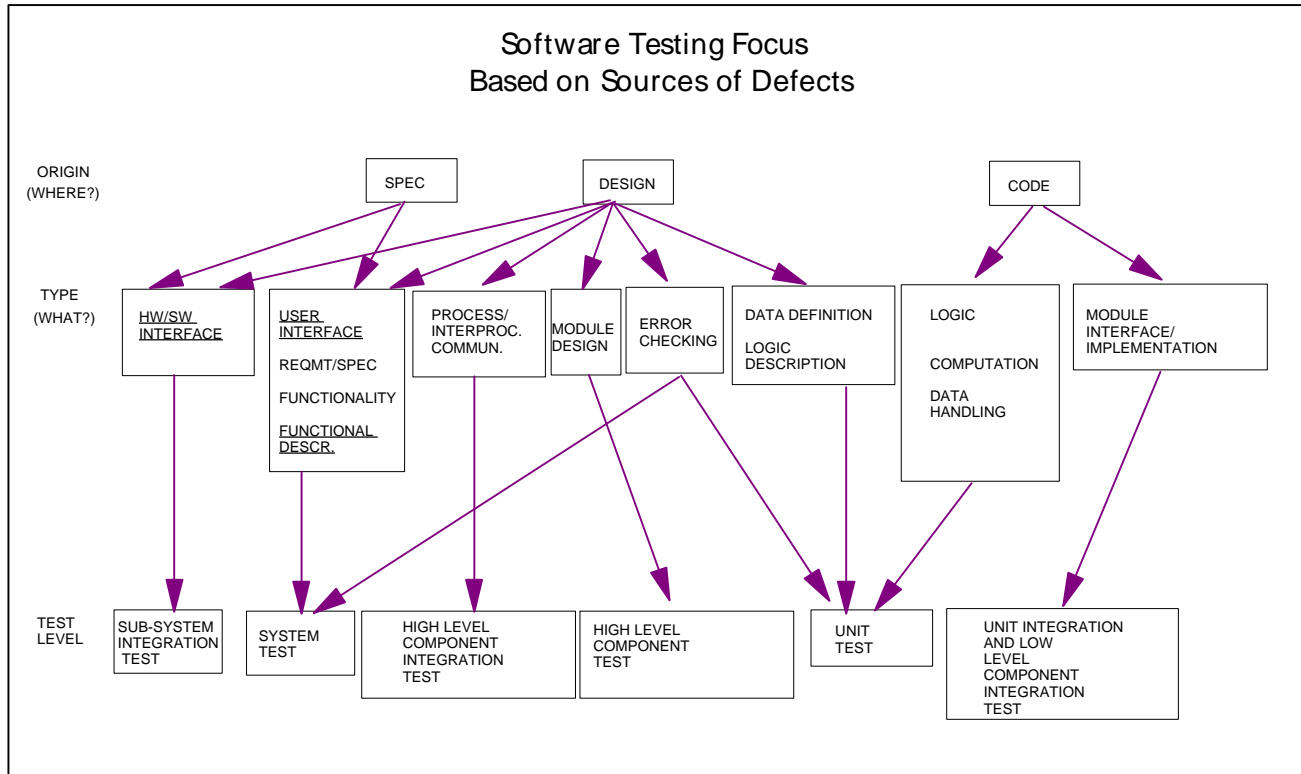
(FIGURE 1-3)

Brief Explanation of Figure 1,2, &3

Since the development of the model in Figure 1-1, it has been successfully used on many Hewlett Packard projects in multiple divisions. One of the first questions asked is why a weighted calculation is needed. The idea behind a weighted calculation is that all defects are not created equal. In other words, it will cost a development team more to fix a defect whose origin is specifications than it will to fix a defect whose origin is code. The main reason for this is the number of work products that will need to be altered to fix a specification defect as compared to a code defect. For example, a developer will need to alter the specification, change the design, and modify the code in order to fix a specification defect. On the other hand, a coding defect only requires a change to one work product, the source code. It is acknowledged that the weighting factors listed in Figure 1-2 may vary from one software development project to another.² The important fact to remember is that for every software development project, weighting factors exist.

Testing Implications (The Model)

The model proposed below (Figure 1-4) will map defect categories, based on their definition, to what appears to be the most appropriate software testing levels³. Based on experience using the model presented in Figure 1-1, most software development defects cluster in Specifications, Design, or Code. The model will focus on mapping the defect Types within these three Origins to particular software testing levels. A software test engineer should be very concerned about focusing testing in the areas where bugs cluster. This model will help with that responsibility. With the model, it is also possible to reverse engineer a defect categorized in a particular area and hypothesize as to what type of testing could have found the bug.



(FIGURE 1-4)

Industry Test Levels Applicable to the Model

The following explanations are applicable only to the software testing levels as contained in Figure 1-4.

Unit Testing – Testing the smallest piece of software that can be independently tested (i.e., compiled or assembled, loaded, and tested). Usually the work of one programmer consisting of a few hundred lines of source code. For a printer, an example of a unit might be the function that sets the paper's Orientation.

Unit / Low Level Component Integration Testing – Testing aimed at exposing interface and interaction defects between otherwise correct and unit tested components. For a printer, examples of low level components might be all the “Page Setup” settings (e.g. Paper Size, Paper Source, Numbers of Copies, and Orientation). An integration test at this level would test the interactions between each of the “Page Setup” settings.

High Level Component Test - Testing components near the top of the call tree (a graphical representation of the calling structure of components). For a printer, the Software Printer Driver itself is an example of a high level component.

High Level Component Integration Test – Testing aimed at exposing interface and interaction defects between otherwise correct and tested high level components. For a printer, this might

include testing the interactions between the SW Driver and other high level components such as the Installer and Uninstaller.

Sub-system Integration Test – Testing aimed at exposing interface and interaction defects between otherwise correct and tested subsystems. For a printer, this might include testing the interactions between the printer Firmware, Software, and Hardware.

System Test

Testing aimed at how the customer will use the product. For a printer, this includes the printer plus 3rd party applications and Operating Systems, but does not include packaging, learning products, etc. (Solution Testing).

Defect Types Suggest Testing Focus Areas

Specification Defect Types

If defect categorization reveals a large percentage of specifications/requirements defects, it would make sense to consider increasing the amount and/or types of System Testing. Specification Defect Types seem to fit best into the System Testing category because the majority of them are related to the customer or user of a product. In the Hewlett Packard defect categorization model it states that specification defects are “A mistake in the definition of the customer/target needs for a system or system performance.” Many of the specification defects found have to do with missing, incorrect, or added functionality. It is likely, however, that these types of bugs are best revealed with thorough System Testing that takes into account areas such as timing, specification domain testing (to test functionality), transaction flow (how a software program behaves), feature interaction, configuration, boundary configuration, hardware configuration, performance, recovery, security, etc. Testing these areas most closely emulates how the customer will use the product.

Concentrations of specification defects in the “Hardware and Software Interface” areas do not indicate an increased emphasis in System Testing. For a printer at least, Hardware and Software are subsystems. Therefore, significant percentages in these defect type areas attest to the need for increased emphasis on Subsystem Integration Testing.

Design Defect Types

The mapping of Design Defect Types to Testing Levels is quite dispersed. The “User Interface” Design Type is best tested by System Testing. As stated in the definition of this defect type these are “problems with incorrect design of how the product will interact with its environment and/or users”. Since the focus here is on the user or customer, a large percentage of defects of this type indicate that the focus of System Testing should increase.

“Process/Inter-Process Communication” design defects are “incorrect interfaces and communications between processes within the product.” This suggests a correlation to “High Level Component Integration” testing, where the interactions between major processes within a product are tested. An increased focus on various interface-testing techniques would be appropriate. For example, Call-Pair and Domain testing techniques. Call-Pair testing concentrates on exercising the interfaces between major processes within a product. Domain testing examines the extreme values of one or more input variables to verify correct interfaces and communication between High Level Components.

Concentrations of design defects in the “Hardware and Software Interface” indicate an increased emphasis on Subsystem Integration Testing.

The definition for the “Module Design” defect type states “Problems with the control (logic) flow and execution within processes”. A “process” most closely parallels a high level component in the test levels described above. High Level Component testing is aimed primarily at major processes within a product. As such, a concentration of “Module Design” defects should trigger an examination of whether or not a tester should look more seriously at testing High Level Components. High Level Component testing should occur in the most isolated context possible. Possible techniques might be private branch component testing, automated regression test suites, and increasing the ratio of dirty to clean tests.

“Data Definition”, “Logic Description”, and “Error Checking” Design Defect Types primarily suggest a map to various “Unit Testing” techniques. The primary reason for this is that each of these defect types is related to the internal structure and data of the program. Software Engineers and Developers are responsible for most Unit Testing since they are also the most familiar with the inner workings and layout of the software product under test.

The Design defect types of “Data Definition”, are defined as the “incorrect design of the data structures to be used in the module/product”, and “Logic Description”, defined as “incorrect data used in conveying the intended algorithm or logic flow”. Based on these descriptions, these types of defects are most effectively tested using Data-Flow techniques. This is a structural testing practice, where what happens to data objects (sequences of events that may alter the status of data objects) are taken into account in determining what paths to test.

Significant percentages of the “Error Checking” design defects indicate that more emphasis be placed on Unit Testing. Specifically, the programmer must thoroughly test whether or not the functions/units written perform in the way expected. This includes whether or not units handle errors properly. If the “Error Checking” defects are more specific to the specifications or requirements, it is likely that a greater emphasis might be given to State and Transition testing at the System Test level. An accurate state diagram should detail expected inputs and outputs specifying software behavior such as error conditions at higher levels.

Code Defect Types

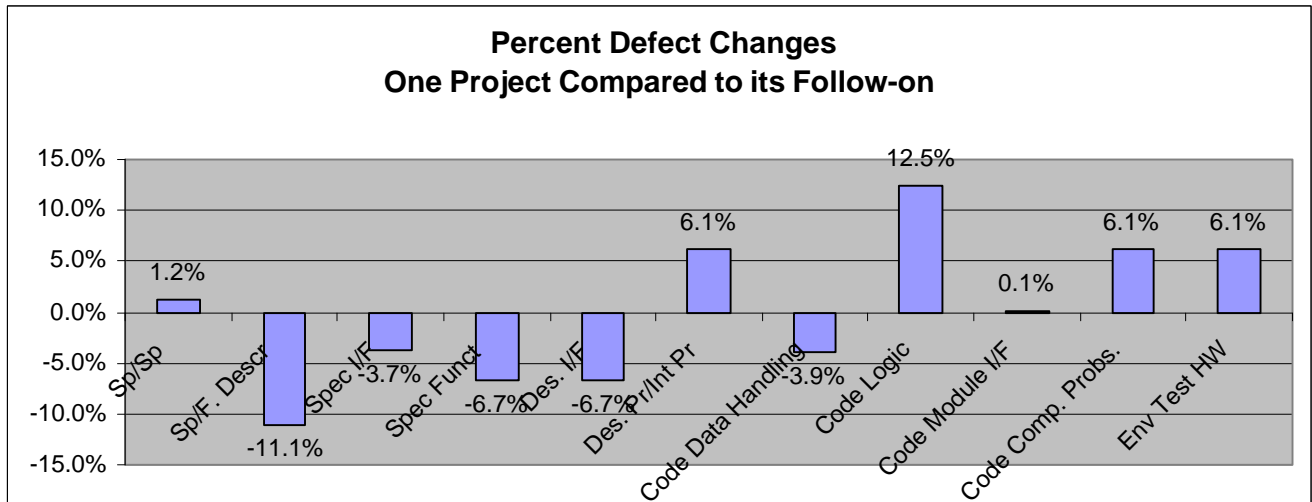
Based on the definition of “Logic” Code Defects, which is “forgotten cases or steps, duplicate logic, extreme conditions neglected, unnecessary function, or misinterpretation errors”, a majority of the defects are likely to be found with thorough Logic-Based Unit Testing techniques.

Similar to the Design defect types “Data Definition” and “Logic Description”, Code “Data Handling” and “Computation Problem” defects may most effectively be tested using Data-Flow testing at the Unit Test level. Both of these types of defects have to do with how the data is interpreted and calculated in the code.

Significant percentages of the “Module Interface/Implementation” defect type indicate weaknesses in the interfaces between Units and Low Level Components. These types of defects suggest an increased focus on interface testing techniques such as Call-Pair and Domain Testing.

Application of Model to a Project

The model described contains ways that Software Testing personnel can examine Software Defect Categorization data and trends. The model, and the explanations of various types of testing that may be appropriate based on concentrations of various types of software defects, is a tool to help the software test engineer. If used properly, the software test engineer can better determine, based on where bugs have clustered in the recent past, where to focus testing in the immediate future. With example defect categorization data, this paper will now explore how this model may be applied to a project.



(FIGURE 1-5)

The data shown in Figure 1-5 represents increases or decreases in the percentage of particular defect types from one project compared to its follow-on:

- Specifications Defects – decreased 20.3%
- Design Defects – decreased .6%
- Code Defects – increased 14.8%

From this information, it appears that process improvements aimed at decreasing the percentage of specification defects, such as a specification inspection program (early testing), are in fact improving the percentage of defects attributable to this Origin. The one defect type within this origin that is still increasing is “Requirements or Specifications”. The project has recently engaged in a Requirements Engineering activity aimed at clarifying and improving requirements. It is hoped that these new requirements will enhance the System Testing efforts already initiated for this product line.

There is not a significant difference in the percentage of defects categorized as Design. The example project did make effort to create and inspect interface documents and this probably why “Design Interface” defects did not appear in the top percentages of the follow-on product. Instead of “Design Interface” defects, the follow-on project had defects attributable to “Design Process/Inter-Process” defects. These types of defects suggest increased emphasis on High Level Component Call Pair and Domain Testing.

Coding defects increased 14.8% from one project to the next. For both projects, there are significant percentages of defects types occurring in the “Data Handling”, “Logic”, and “Module Interface / Implementation” areas. In the follow-on project, there is also the appearance of “Computation Problems” as a code defect type category. According to the model, “Module Interface / Implementation” defects suggest more focus be placed in Unit / Low Level Component Integration Testing. Interface testing techniques such as Call Pair and Domain Testing should be considered at this level. “Data Handling” and “Computation Problems” defects suggest difficulty with manipulating data at the Unit Test Level. Increased Data-Flow testing should help reveal additional weaknesses in these areas. Code “Logic” defects indicates that an increased emphasis should take place in considering the logic flow of a program. Logic-Based Testing techniques, such as testing code using Boolean Algebra, are important in discovering improvement opportunities for “Code Logic” defects.

Each software development project will have different defect trends. As such, the focus for testing can and should be different from project to project. The explanation above illustrates how any Software Development Project can interpret defect categorization data collected from their own project to improve testing efforts. By focusing testing efforts based on these results a testing organization can:

1. Better plan testing efforts based on real project defect data.
2. Focus testing on areas where the bugs are most likely to be hiding.
3. Assuming the current product is based on reused and/or leveraged code from a previous product, the current product can take advantage of incremental improvement. This is done by adjusting the testing focus for the current project based on defect trends from the most recently finished project.

Much work has taken place in the Software industry to categorize and define defect types for Software Development. What has received less emphasis is software testing. The paper will now propose a model to help software testing professionals analyze software testing defect trends.

A Simple Model for Software Testing Defect Analysis

This section of the paper will:

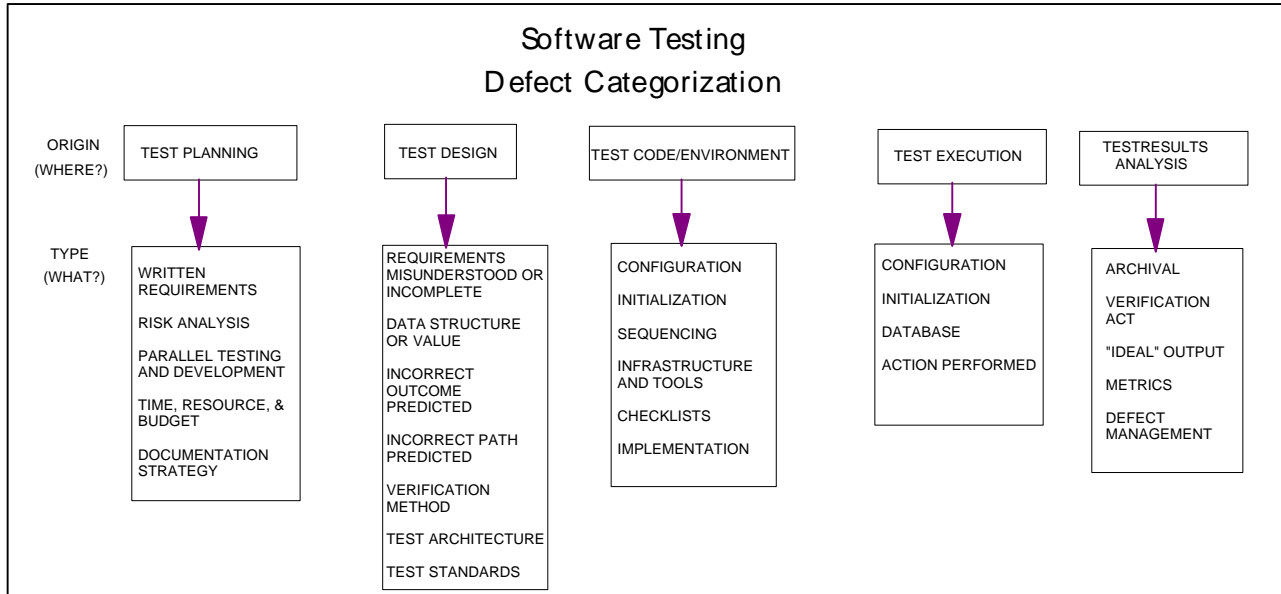
1. Introduce and define the second model entitled “Software Testing Defect Categorization”.
2. Discuss the environment that must exist for the model to be used successfully.
3. Provide an example of data that could be generated by applying the model to a project.

Although there are some industry views on this subject, there is much to be done to make Software Test defect logging and analysis a common practice. One of the reasons the Hewlett Packard model for categorizing software development defects has been successful is because the “Type” portion of the model has few enough types that engineers have little trouble categorizing their defects. These types are based on causes of defects, rather than any possible reason for failure. The model proposed will keep number of test defect “Types” manageable.

There are at least three areas of Software Testing defects. These are testing process, testing management, and test product defects. Test Process defects are errors in the strategy and procedures necessary for successful testing. When the resources and policies necessary for successful testing are not in place, this is considered a Test Management defect. Errors that result

in changes to the test itself are Test Product defects. The categorization model described in Figure 1-6 contains defect types that can be applied to each of these three areas.

Portions of the model described below are based on “Test Definition or Execution Bugs” as defined by Dr. Boris Beizer⁴.



(FIGURE 1-6)

Test Planning

Written Requirements

There are always testing requirements. The question becomes whether or not they are recorded somewhere. This type of defect indicates that an initial set of documented testing requirements does not exist or are of poor quality.

Risk Analysis

There are risks with every testing endeavor. Defects of this type have to do with not engaging formal risk analysis as part of test planning. Some risks may be testing new technology, initializing new test infrastructure and/or tools, the outsource of testing, lack of resources at particular times in the testing life cycle, and testing over multiple protocols/languages, etc. Prioritizing these risks, communicating contingency plans, and identifying other holes and gaps in the testing strategy are essential to success.

Parallel Testing and Development

At certain times in the product lifecycle, cooperation between the software developer and software tester is necessary. When initial requirements are generated, the tester and the developer should communicate and agree that the requirements are testable and can be implemented (coded). At that point, the developer creates the product and the tester develops tests. Once the tests are applied to the product and defects occur, communication is again necessary between the tester and developer. The defect must be analyzed to determine if it is a Testing or a Development defect. A breakdown in the communication between the tester and developer at the point of initial requirements and when defects are encountered constitute this type of defect.

Time, Resource, & Budget

Management support of a testing project is essential. This type of defect occurs when there is little or no management support and a realistic testing schedule with sufficient time, resource, and budget, is not developed.

Documentation Strategy

It is important to decide up front what documents are essential to the success of the testing process. At a minimum, there should be written test requirements (mentioned above) and a thorough test plan. As with any project, plans and estimates always change based on reality. A document change management strategy to review and approve/disapprove modifications to the testing requirements and test plan must exist. This change management strategy must be agreed to by both development and testing personnel. If testing requirements and plans change without any agreed on process, this type of defect occurs.

Test Design

Requirements Misunderstood or Incomplete

The test and the component under test are mismatched because the test designer did not understand the requirements. This becomes apparent when the test is applied to the product. A defect of this type may also occur because the requirements for the test are not documented thoroughly. At a minimum, test design should document the configuration, start situation, the actions to be performed by the test, and the expected outcome.

Data Structure or Value

Data objects used in tests or their values are wrong. Defects of this type are caused by incorrectly designing data structures or by assigning wrong values to data within a test.

Incorrect Outcome Predicted

Predicted outcome of a test does not match required or actual outcome.

Incorrect Path Predicted

Outcome is correct but was achieved by the wrong predicted path. The test is only coincidentally correct.

Verification Method

The method by which the outcome will be verified is incorrect or impossible.

Test Architecture

The style or method for constructing the test is missing or in error. Also, exactly what is being tested is not well documented or understood.

Test Standards

The test design does not comply with locally accepted design standards.

Test Code/Environment

Configuration

The hardware and/or software configuration and/or environment specified for the test is wrong. The configuration portion of the test design is documented but incorrect. Whatever environment

the test was supposed to run in is not correct. Configuration takes into account many factors such as operating systems, applications, peripherals, networking, etc.

Initialization

The specified initial conditions for the test are wrong. The start situation of the test design is documented but not correct. Examples might include the value data variables and function parameters are set to when a test begins.

Sequencing

The sequence in which tests are to be executed, relative to other tests, or to test initialization, is wrong.

Infrastructure and Tools

The test suite management and execution tools, and the infrastructure necessary to utilize them properly, are not set up correctly or do not exist.

Checklists

The absence of exit and entry criteria, release notes, procedures to obtain code, resources needed to successfully test, and other checklists necessary to set up the environment prior to Test Execution constitute a defect of this type. Emphasis on exactly what is received from the development community, is it ready to test and how this is determined, are factors that should be considered when evaluating the test environment.

Implementation

There is an error in the code necessary to put in effect or carry out the test. This could be an inaccuracy in the test harness or any other instrument directly related to completing the test.

Test Execution

Configuration

The configuration and/or environment specified for the test was not used during the run. A valid environment was specified but, because of an error in the execution of the test, that environment is not the environment where the test is run.

Initialization

The tested component is not initialized to the right state or value. The start situation of the test design is correctly documented but an error in executing the test causes the initialization to be wrong. Examples might include when data variables and function parameters are set to incorrect values when a test begins execution.

Database

The database used to support the test is wrong or data is incorrectly entered.

Action Performed

This may be as simple as a keystroke or button hit error. However, this type of defect may also be more complicated than simple input. For example, if for some reason when the test is executed, it does not run according to its design, due to factors like network traffic, machine downtime, or failure of tools the test is dependent on.

Test Results Analysis

Archival

The results of testing are not archived electronically or in hard copy form. It is dangerous not to keep a record of the testing that has occurred. For the final test run, it is often necessary to archive results to meet legal requirements. Very often, follow-on projects will need to refer to the results of testing from their predecessor(s).

Verification Act

The act of verifying the outcome was incorrectly executed. The defined method for verifying the outcome is correct but performed erroneously.

“Ideal” Output

This type of defect occurs when no process exists to update the “ideal” outcome of tests. Many tests cannot be termed a success unless the results of test execution are compared to what is thought to be the ideal outcome of the test. If the results of test execution match the “ideal” output, then the test is called a success. What is often discovered, when the same test is run on a product and its follow-on, is that the “ideal” outcome for a test needs to be amended.

Metrics

Nothing is in place to measure the status of testing processes and products. Some measurement programs focus on the status of the product under test, rather than testing processes and products. It is important to measure the testing process in terms of effectiveness and efficiency. Some important factors to measure are defects found by tests, various types of coverage, the degree of automation, estimated as compared to actual costs, ability to meet schedule, and test center throughput.

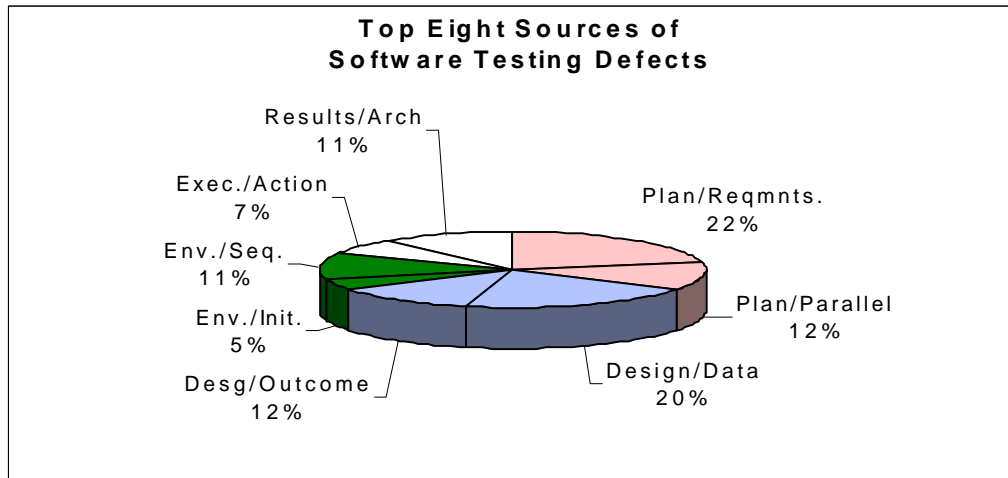
Defect Management

Defects must be managed by the responsible testing organization. This type of defect applies when the defect management system is exclusively focused on the development product and no effort is made to align the defects being found to the tests that found them.

Application of the Model

In order to apply the “Software Testing Defect Categorization” model successfully, the following must be in place:

1. There is a distinction in the defect management system between a product defect and a test defect. This implies that both defect types are logged in a defect management system.
 2. Test Engineers must log testing defects as they occur.
 3. The testing team must be committed to categorizing a subset of defects after each test project.
- As with any Failure Analysis and Process Improvement effort, analyzing trends once does little or no good. To gain the full benefit, the team must commit to categorizing defects after each project, implementing process improvements on the subsequent project, and then reevaluating defect trends. This will help to measure whether or not the process improvements resulted in the betterment of the software testing. Let's assume, in this fictitious example, that the Testing defect trends for a project looked like the following:



(FIGURE 1-7)

If this were data from a real project, the test team would need to decide which areas to focus on for Testing Improvement. Some of the more obvious areas where defects are occurring are in Test Planning and Design. If the team decided to limit its process improvement focus on Design, the next step would be to consider root causes as to why a significant percentage of defects are attributed to Test Design. Improvement strategies specific to the project should then become apparent. Project specific improvement strategies would then be implemented on the next project. When the next project is finished, the defect trends would be examined again to determine if the testing improvements did in fact make a difference. With this type of incremental improvement, the advantages of software testing defect categorization and root cause analysis become evident.

Test Defect Weighting

As mentioned previously, in reference to software development defects, all defects are not created equal. This is also true for defects attributed to software testing. Defects that are categorized as Test Planning or Test Design defects are more expensive to understand and correct than Test Code/Environment, Test Execution, and Test Results Analysis defects. For example, initializing a critical data variable to an incorrect value is not as difficult to detect or fix as if the test design never accounts for testing the variable. It is acknowledged that weighting factors may vary from one software testing project to another. The important fact to remember is that for every software testing project, weighting factors exist.

Conclusion

Software Development Engineers have been analyzing defect trends, and making improvements based on the data, for many years. It is time that Software Test Engineers do the same. This paper has introduced two models. The first model that can help start the process of interpreting software development defect trends and, based on those tendencies, determine what areas of testing might be a appropriate for focus in the immediate future. The second model introduces a way to categorize software testing defects, enabling better understanding of what improvements might be possible in the project's testing lifecycle. Both of the models allow Software Testing Organizations to benefit hindsight. This hindsight is gained by possessing data in relation to what the defect

trends are for both Software Development and Testing. This hindsight can be turned into the foresight necessary to target and improve Software Testing efforts and practices.

Acknowledgements

Thanks go to my immediate manager, Len Schroath, and my functional manager, Anne Vermilion. Thanks for believing in the people that work for you.

I would like to thank my colleagues Susan Davis, Mike Dunlap, and Felix Silva for providing valuable comments and suggestions in relation to the contents of this paper.

I would also like to thank Bob Grady, author of three books and numerous articles, for introducing me to Software Failure and Root Cause Analysis. Without learning what I did from his experience and expertise, this paper would not have been possible.

References

1. Grady, Robert B., "Practical Software Metrics for Project Management and Process Improvement". Prentice Hall, Inc., (1992), pp. 127-128, 223-227.
2. Boehm, B., "Software Engineering Economics". Englewood Cliffs, NJ: Prentice-Hall, Inc., (1981), p. 40.
3. Beizer, B., "Software Testing Techniques". Boris Beizer, (1990). Printed by Van Nostrand Reinhold, NY. Software Testing Levels, as contained in (Figure 1-4) and the explanation following the diagram, are based in part on definitions in various sections of "Software Testing Techniques".
4. Beizer, B., "Software Testing Techniques". Boris Beizer, (1990). Printed by Van Nostrand Reinhold, NY, pp. 475-476. The defect types as contained in (Figure 1-6) and the explanation following the diagram are based in part on Bug Statistics and Taxonomy as defined in "Software Testing Techniques".

A simple model to predict how many more failures will appear in testing

A. Bertolino, E. Marchetti

- Introduction
- Bemar model
- Case studies
- Future work

Static models of software defects

Software metrics are used to estimate the number of defects in the software

General form

$$y = f(x_1, x_2, \dots, x_n)$$

y is a defect metric, such as:

- number of changes required in the design
- number of errors
- number of program changes

x_1, \dots, x_n can be:

- product-related
- process-related

Akiyama's study

Metrics used (product-related)

- program size in lines of code (S_s)
- count of decisions (DE)

Total number of defects

$$d_{tot} = 4.86 + 0.018S_s$$

$$d_{tot} = -1.14 + 0.2DE$$

Dynamic models of software defects

- The software system is considered as a “black box”
- Tests are developed using the operational profile
- The reliability is estimated without considering the complexity of the program
- Test are developed for:
 - increasing the reliability
 - estimating the reliability

Well known examples are:

- Musa model for the time between failures data
- Yamada S-shaped model for the failures per time period data
- Goel/Okumoto model for both

Bemar model: motivation

- Sometime identifying an operational profile is quite difficult and expensive
- Operational testing is applied to the whole system, or to big-size portions of it
- Operational testing can only start when the software configuration and behavior are fairly stable
- In general, commonly used debug test methods do not exhibit a regular trend in reliability
- Oftentimes for a software producer modifying the test process is not easy

Bemar model: purpose

- To predict the number of remaining failures when:
 - ◊ the operational profile can't be identified
 - ◊ the test involved single modules or small pieces
 - ◊ the test process is in the early phases, which are functional and deterministic
 - ◊ the rate of detection of failures remains quite stable
- To provide the software producer with a method that:
 - ◊ can be applied without any change to the test process
 - ◊ establishes the effectiveness of the tests performed so far
 - ◊ establishes a stop criterion for the testing

Bemar model: application

- Collect the data during the test
- Establish a test interval (TI) length
- Group failure data into test intervals
- Apply the Bemar method

Cai's method

Assumptions

- Modules are randomly divided in part 0 and part 1
- There are $N=N_0+N_1$ remaining defects in the software:
 - N_0 in part 0
 - N_1 in part 1

Use

- Perform code review on a randomly chosen module
- Establish to which part (0 or 1) the module that shows a defect belongs
- Use the number of defects discovered to predict how many defects should be detected during the phase of dynamic testing in each part (N_0 , N_1)

Model rationale

Intuition

- The number of failures f can be estimated by:

$$f = n * t$$

- n the total number of tests executed
- t is the failure detection rate

Problems

- A distribution should be used instead of a known failure detection rate
- The empirical distribution for t can only be identified after tests are completed

Solution

- A Bayesian approach to derive the distribution of t from the observation of test results.

Bayesian approach

- "subjective" interpretation of probability
- allows consistent deductions from probability statements, and inference from observation
- given *prior* probabilities and new observation, derives updated *posterior* probability:

$$\text{P}(\text{conjecture} \mid \text{observation}) = \frac{\text{P}(\text{observation} \mid \text{conjecture}) \text{P}(\text{conjecture})}{\text{P}(\text{observation})}$$

posterior probability (points to the left side of the equation)

likelihood (points to $\text{P}(\text{observation} \mid \text{conjecture})$)

prior probability (points to $\text{P}(\text{conjecture})$)

To predict the failures

A random variable T , that takes values in $[1..M]$, is defined as the distance between two successive failed test intervals

First step

Predict the number of failed test intervals N_{FTI}

Second step

Predict the number of expected failures N_F

First step

- Establish a prior distribution for T
- Derive a posterior distribution for T (after a given number k of test intervals):

$$P(T = i | F_k) = \frac{p_T(i) \cdot \left(\frac{1}{i}\right)^f \left(1 - \frac{1}{i}\right)^{k-f}}{\sum_{j=1}^M p_T(j) \cdot \left(\frac{1}{j}\right)^f \left(1 - \frac{1}{j}\right)^{k-f}}$$

- Predict the number of failed test intervals

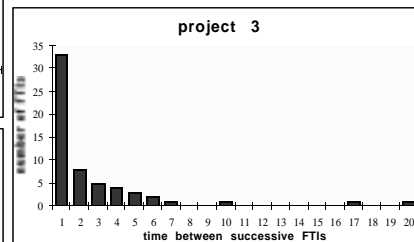
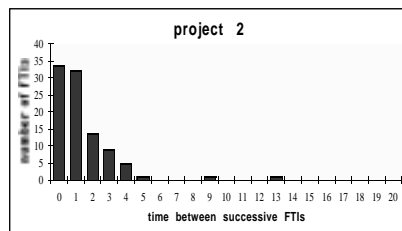
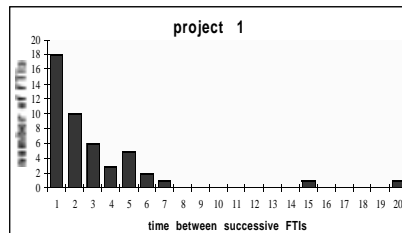
$$N_{FTI,k} = \frac{NTI}{E_k[T]}$$

Second step

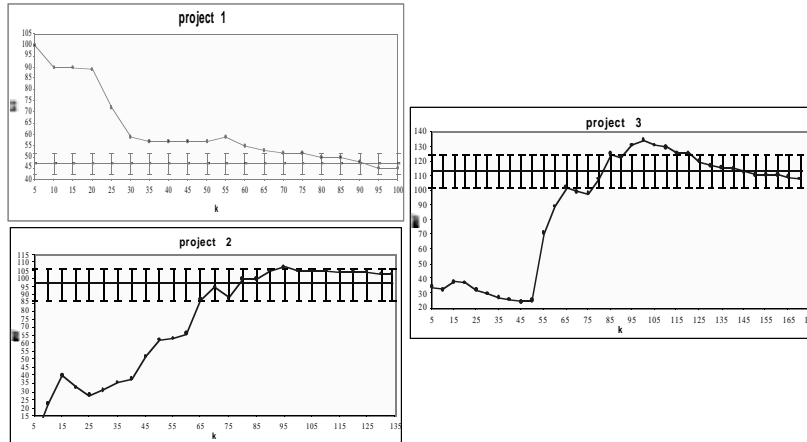
- Derive the mean number of failures observed within a failed test interval: $E_k[F]$
- Predict the number of failures that the product will show at the end of the test

$$N_{F,k} = N_{FTI,k} \cdot E_k[F]$$

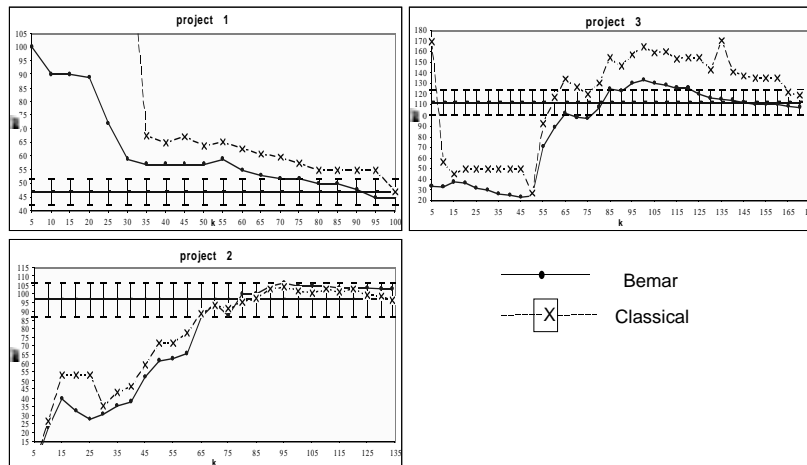
Sets of failure data



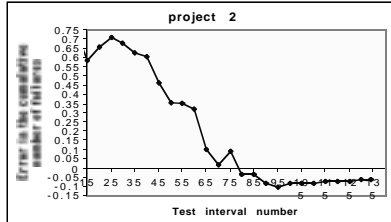
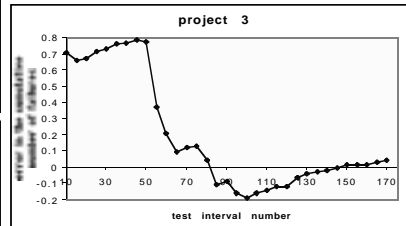
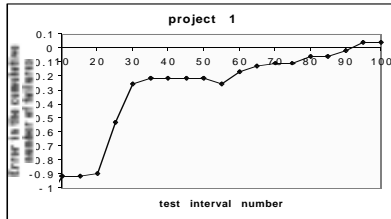
Model predictions



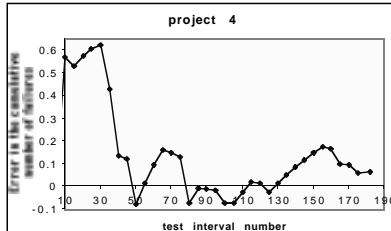
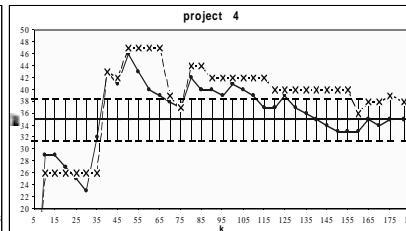
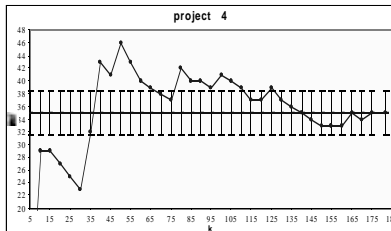
Comparison with a "classical" approach



Relative error

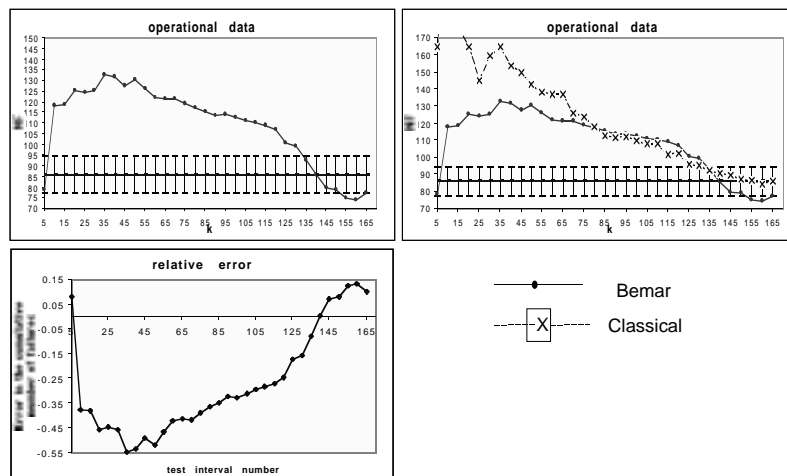


Application to a set of operational data



—●— Bemar
 - - - X - - - Classical

Application to a published set of data



Future work

- Find more efficacious ways to express prior knowledge
- Quantify in advance the goodness of Bemar prediction
- Validate the Bemar method with more data
- Use of Bemar in combination with a reliability growth model

A simple model to predict how many more failures will appear in testing

A. Bertolino, E. Marchetti

Istituto di Elaborazione della Informazione, CNR, Pisa, Italy

bertolino@iei.pi.cnr.it, e.marchetti@iei.pi.cnr.it

Abstract

This paper deals with dynamic models to evaluate how many more failures will be observed in future tests, based on the failures observed so far. The assessment of reliability through testing is now one of the most mature fields in software engineering. There exist tens of reliability growth models, and several tools for applying them. The major assumptions of these models are that test cases are randomly drawn from the operational profile, and that as defects are found and removed, reliability will exhibit an increasing trend. Both assumptions are hardly satisfied in the first stages of the testing process or for the testing of small modules. Besides, there are not reasons why commonly used test methods at this time, such as specification-based testing or branch coverage, should exhibit a regular trend in reliability.

These are the motivations for the work reported here. A dynamic model is introduced that can be applied to predict the number of remaining failures in early test phases. It is called the Bemar model. The Bemar model is quite general and makes no assumption on as to how tests are selected. The most attractive feature is indeed the simplicity of the model: testers have just to collect the detection rates of failures, i.e., the intervals between subsequent failures. No estimation of parameters of the product or of the development process is required.

Keywords: Bayesian approach, defect count models, functional testing, number of expected failures

1. Introduction

In spite of great advances in the software engineering field since the complaints about a software crisis began to spread in the mid-seventies, the state of practice in software development is still such that producing defect-free code remains wishful thinking. On the contrary, coping with software failures, during development and after release, is among the hardest tasks of managers, while testing, debugging and maintenance activities still consume the largest part of development effort and resources. For these reasons, methods to estimate the defect contents of software are of great interest for managers and testers.

Researchers have devoted much attention to this problem and have proposed many models to quantify faults and failures. It is important to distinguish between two different approaches that have been taken. One approach consists of looking at properties of the present or past products, and/or at parameters of the development process and then, using these observations, trying to make a guess of the total number of defects, or faults, in the current product. A different approach

is instead to observe defects, or, more properly, failures, as they show up in testing. Based on the observed behavior, one then uses statistical inference procedures to predict the number or the time of failures expected in future tests or in operation.

Depending on which of the two approaches is followed, defect counting models have been categorized as static or dynamic, respectively [Conte et al., 1986, Chapter 7]. However, the fact that static and dynamic models assess two different entities, namely defects in the code the first and failures to be observed the second, must be underscored.

Static models are very attractive to managers, because they provide "numbers", which the managers are eager of, very early in advance in comparison with dynamic models. The latter can only be used late in the life cycle, i.e., in the testing phases, when it may be too late to efficaciously re-direct development efforts. In fact, static defect models are used to identify more risky modules and consequently re-allocate testing resources or modify design. In addition, static models claim to estimate the total number of defects. As by testing we find and fix failures, then, static models would provide a prediction on how many defects are left in the code, which may seem a very attractive measure at first glance.

On the other hand, a defect can be more or less disturbing depending on whether, and how much frequently, it will eventually show up to the final user (and depending of course on the seriousness of its consequences). Indeed, in many or in few, some defects will inevitably escape testing and debugging. So, in the end, the real important measure to decide whether a product can be released is software reliability; i.e., the number of *failures*, and not of remaining defects, must be estimated. Until they do not cause failures, remaining defects do not trouble neither customers nor producers.

The right position is that static and dynamic models are both useful, but for different objectives. In the front-end phases of the life cycle, managers should use static models to apportion risk among modules and to allocate development time and resources. In the final stages of development, instead, they should use dynamic models in order to evaluate how much disturbing are the defects that are inevitably left, and to decide whether the product is ready for delivery.

This paper deals with dynamic models to evaluate how many more failures are expected to be observed in future tests, based on the failures observed so far. The assessment of reliability through testing is now one of the most mature fields in software engineering [Lyu, 1996]. There now exist tens of reliability growth models, and several tools for applying them, in combination with rather sophisticated techniques to evaluate the accuracy of the measures given by the models, and to select the most appropriate model for a specific data set.

Existing models, though, all share the underlying assumption that the test cases are randomly drawn from the operational profile, and that as defects are found and removed, reliability will exhibit an increasing trend. Both assumptions are hardly satisfied in the first stages of the testing process. Industrial test processes commonly undergo several subsequent steps, identified with differing terms, from unit to subsystem, and to system testing. Operational testing can only start when the software configuration and behavior are fairly stable, and is applied to the whole system, or to big-size portions of it. For the testing of single modules, or of small subsystems,

identifying an operational profile is quite difficult and expensive, and perhaps not sensible at all. Besides, there are not reasons why commonly used test methods at this time, e.g., branch coverage, should consistently exhibit a regular growth in reliability.

These are the underlying motivations for the work reported here. We introduce a dynamic model, called the Bemar model, that can be applied to predict the expected number of remaining failures in early test phases. The Bemar model is quite general and makes no assumption on as to how test are selected. The most attractive feature is indeed the simplicity of the model. It only requires to collect the intervals of time between subsequent failures. No estimation of parameters of the product or of the development process is needed.

In the next section the underlying intuitive model is described; the mathematical formulation is provided in Section 3. The model has been applied to some real world data; the results are presented in Section 4. Although the data available are too poor to validate the model, these first results look promising. This work is still in a preliminary phase; we briefly outline future directions in the Conclusions.

2. Model Rationale

In measurement, one tries to map observations of the empirical world to mathematical entities that can be formally manipulated. Models are defined trying to capture one's intuition and understanding of the real world; indeed, "intuition is the starting point for all measurement" [Fenton and Pfleeger,1997]. In this section we present the intuition underlying the Bemar model.

The stimulus for this work came from the analysis of the test results collected over several projects by a software producer, namely Ericsson Telecomunicazioni S.p.A. in Rome. This producer routinely logs for each product the failures observed since early test phases until beta testing, and is interested in finding more effective ways to use these data for project management and product control. So far, these data are used to derive measures of fault density, that is the ratio between the cumulative number of failures observed in a given time period and the product size, expressed in lines of code.

With regard to the results from beta testing, which is operational, standard approaches for reliability estimates and predictions can be applied. In [Bertolino et al., 1998], we describe a first case study conducted at the same producer, aimed at experiencing the use of software reliability engineering techniques. But, reliability growth models could not be applied to the early test phases, for the reasons we explained in the introduction.

It must be made clear beforehand that it is not the case that this producer is looking for new testing methods to be applied that would facilitate failure predictions (as could be for instance the case if fault seeding approaches were applied). On the contrary, this producer has a well established and formalized test process, and is looking for efficient metrics that can be applied to the data collected. It is plausible to assume that to a certain extent this proviso would be the same for many other producers.

We surveyed the literature in search for a dynamic model that could be applied to the test outputs from the early test phases; reliability growth models could not be used, as earlier explained. An

interesting finding of this survey was [Cai, 98]. Cai has proposed a model to predict the remaining number of defects in the code based on the failures that are observed in testing, which is in a sense a hybrid approach between static and dynamic models. Since the assumptions underlying Cai's model reasonably held for the projects of this producer, the model was applied to the data available, in order to see if the estimation of defects provided by the model was conclusive for our situation, but with negative results.

We investigated on the reasons why Cai's method, which reportedly worked well on his data, did not function on our data. One of the findings was that Cai's model does not consider the time occurrence of failures. Intuitively, Cai's model is similar to fault seeding methods, but instead of considering the proportion between seeded faults and unknown faults, Cai divides the software under test into two parts, and uses the relative occurrence of (real) faults in either parts. The model is thus only concerned with the number of faults and possibly with how these are distributed among the modules of a system, but not with the time of their detection.

In our opinion, the rate of failure discovery is a fundamental parameter, and should be included in the model. In simple words, the scenario we have in mind is that n failures are detected after d days of testing, and that we want to estimate how many more failures we expect to find in the next d' days, if we continue to test in the same way. We reasonably think that the prediction should be different if the failures are uniformly distributed over the d days, or if instead all the failures are, say, discovered in the first day of testing, and then the remaining $(d - 1)$ days exhibit no failures.

We have consequently defined a new dynamic model taking into account the time distribution of failure discoveries. The intuition behind this new model is very simple: assuming that we can know a priori, or somehow estimate, the rate of failure findings over the sequence of executed tests, say t , then if by n we denote the total number of tests to be executed, quite obviously the expected number of failures f would be estimated by:

$$(1) f = n * t$$

Of course this formula is rather naive and cannot be used in practice in this simplistic form, because the rate of failure detection in testing can never be established with certainty; it is rather a random variable, for which a distribution should be identified. For each new product under test, the empirical distribution of the failure detection rate can only be precisely drawn only after the testing is completed. However, if we could assume that, after having observed the test results for some time it stabilizes (i.e., it can be used as an approximation of the real, so far unknown distribution, to predict future behavior), then a formula generalizing Eq. (1) could be used. This is the underlying intuition of the Bemar model. To derive the distribution of the failure detection rate from the observation of test results, Bemar uses a Bayesian approach. This is described in the next section.

According to its justification, we expect that Bemar performs better when the rate of detection of failures in testing remains more or less stable. This is in contrast with the assumption underlying reliability growth models. In fact, the Bemar model should be applied to early test phases, and in general to all those situations in which failures are found with some regularity, and remains valid

only for limited periods, i.e., till the point in which the rate of occurrence of failures starts to decrease as an effect of having removed a high number of faults.

In other words, the Bemar model performs well as long as reliability growth models cannot yet be applied. It is foreseeable that the Bemar model and a reliability growth model can be used in complementary way. How these could be combined will be object of future investigation.

3. Description of the Bemar Model

Before presenting the definitions and formulas adopted in the model, the typology of data available is described.

The software producer provided us with sets of failure data collected over several projects during the phase of subsystem testing. The test cases are deterministically chosen by examining the functional specifications and altogether before test execution starts (which means that the number of tests to be executed is decided in advance). The tests are not executed continuously, but only during the working days (i.e., five days in a week) and 8 hours per day. For each project, the information registered consists of the start and end dates of the test phase, and of the calendar day (but not the day time) of discovery of each failure. Test execution (CPU) times were not recorded.

Based on the coarse granularity of available data, we decided to group failure data into test intervals (TIs). A TI could be as long as a day, a week, or any other length (for instance measured in seconds), depending on the global duration of the testing, the precision of the data available and the amount of observed failures.

A TI in which at least a failure is observed is called a *failed test interval* (FTI), otherwise it is said a successful TI. Note that, anyhow small a test interval is chosen, until this remains larger than a single test there will always be a chance to observe more than one failure within a failed test interval. Hence, we predict the expected number of failures in two steps: first we predict N_{FTI} , i.e., the number of FTIs is estimated. Then, from this number, we derive the number of failures N_F .

In the first step, to estimate N_{FTI} , we define the distance between two subsequent FTIs as a random variable T , that can take discrete values within an interval $[1, M]$ (where M is a maximum fixed value). Precisely, for each i within $[1, M]$, the associated probability mass function (pmf), $p_T(i) = P(T=i)$, gives the probability that the next failure will be observed after i TIs (i.e., $i - 1$ successful TIs are observed and then the i th TI is a FTI).

Denoting by NTI the total number of test intervals to be performed, and with $E[T] = \sum_{i=1}^M p_T(i) \cdot i$, it can be shown that the following formula holds¹:

$$(2) \quad NTI = N_{FTI} \cdot E[T]$$

¹ Actually, this formula holds precisely if it can be assumed that the last test interval is a failed one. Otherwise, the left-hand side should be decreased by the number of test intervals occurring between the last FTI and the last test interval. This adjustment will be neglected in the paper.

Since for each project the number of test intervals can be easily derived (remember that the functional test cases are preselected in advance), Equation (2) above can be solved for N_{FTI} , yielding:

$$(3) \quad N_{FTI} = \frac{NTI}{E[T]}$$

We need now a procedure to derive $E[T]$. Looking at the data available, we see that the failures are variously distributed over the whole test period and it is not generally the case that towards the end of the functional test period less failures are observed than at the beginning (as it is expected in operational testing). In particular, the data do not show any consistent reliability increasing trend, appearance which was confirmed by the Laplace test [Kanoun et al., 1997] conducted over all the sets of data. In Figure 1, we show for instance the failure data relative to one of the products analyzed.

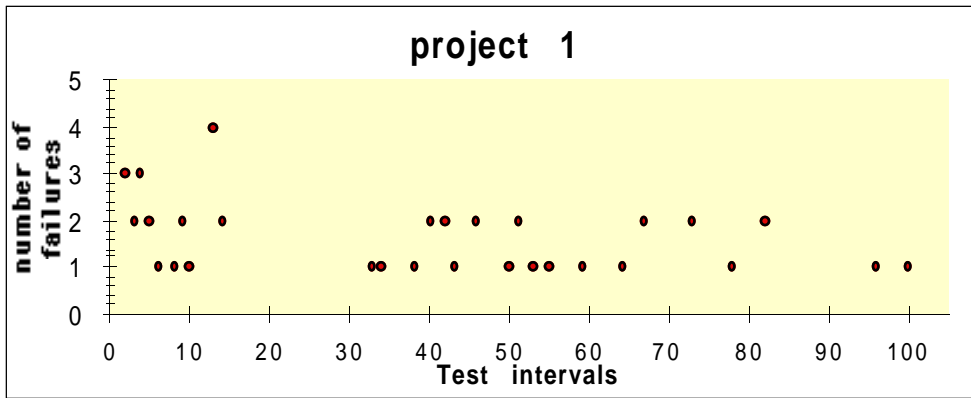


Figure 1: Failure data for a product

To develop a prediction procedure which is sensible, and correlated with the behavior of a given product under test, it is desirable to use the test results collected as functional test proceeds to adjust an initial estimate of the pmf. Hence, we chose to adopt a Bayesian approach.

In the Bayesian framework, probabilities are meant to describe an observer subjective knowledge of yet-unknown events. This knowledge evolves as events are observed. In this context, the pmf of T $p_T(i)$ is taken as the *prior* knowledge about the behavior of a product under test. I.e., $p_T(i)$ is taken to model a tester's subjective belief about the rate of failure detection *before* some evidence (the test results) about the product under test is observed. During the performance of the functional testing, the realization of a sequence of test intervals with and without failures is observed. Thanks to this evidence, the tester's knowledge about this product evolves and a new distribution for the pmf of T , called the *posterior* distribution, can be derived. Denoting by F_k the sequence of observed outcomes (failed/successful) for the first k TIs, the posterior distribution $p'_{T,k}(i)$ then gives $P(T=i | F_k)$, i.e., it is the update of $p_T(i)$ after having observed the sequence F_k . Applying Bayes' formula we have:

$$(4) p'_{T,k}(i) = (P(T = i | F_k)) = \frac{P_{prior}(T = i)P(F_k | T = i)}{\sum_{j=1}^M P(F_k | T = j)P_{prior}(T = j)}$$

in which the term $P(F_k|T=i)$ is usually called a *likelihood function*. To derive it, we can consider that, if $T=i$, then the probability of observing a failure in the next test interval is $1/i$, i.e.:

$$(5) P(F_1|T = i) = \begin{cases} \frac{1}{i} & \text{if } F_1 \text{ is failed} \\ (1 - \frac{1}{i}) & \text{if } F_1 \text{ is successful} \end{cases}$$

Substituting this in formula 4, and iterating the same reasoning also to the subsequent test intervals, we finally obtain²:

$$(6) p'_{T,k}(i) = \frac{p_T(i) \cdot \left(\frac{1}{i}\right)^f \left(1 - \frac{1}{i}\right)^{k-f}}{\sum_{j=1}^M p_T(j) \cdot \left(\frac{1}{j}\right)^f \left(1 - \frac{1}{j}\right)^{k-f}}$$

which gives the posterior pmf for the random variable T, after observing k test intervals, out of which f were failed.

In general, deriving a prior distribution for the probability of interest is a difficult task, which also generates some perplexity towards the usefulness of Bayesian inference methods. In our case, the form of $p_T(i)$ can be derived on the basis of data available from similar products. In general, some suitable representation of ignorance is often adopted, like for instance the uniform distribution, though actually absolute ignorance can never be assumed.

By using the posterior pmf provided by formula (6) to derive $E[T]$, by (3) we can then derive $N_{FTI,k}$, i.e., the number of FTIs expected after N_{TI} test intervals, using the test information collected during the first k test intervals.

From $N_{FTI,k}$ the total number of failures N_F needs now to be estimated. This clearly depends on how many failures on average are observed within a FTI. We can again define a random variable F to represents the number of failures observed within a FTI, and then derive N_F from N_{FTI} , with $N_F = N_{FTI} \cdot E[F]$.

We derive an empirical pmf for F by considering the results from the first k TIs. In particular, by analyzing the sets of failure data available, a maximum number of failures per FTI, MF , can be fixed. From the distribution of the number of failures within a failed test interval, we are able to calculate the expectation $E_k[F] = \sum_{i=1}^{MF} P_k(F = i) \cdot i$.

²In the generalization of this formula from the case $k=1$ to larger values of k , we have in reality used some relaxed assumptions, which could raise some objection to its validity from a purely theoretical viewpoint. In future work we will fix these problems. However, on the set of data available, this formula performed better than other theoretically stronger models.

Therefore, after having observed k FTIs, the number of failures that a product will show at the end of the functional test is:

$$(7) N_{F,k} = N_{FTI,k} \cdot E_k[F]$$

The formulas (3) and (7) are to be used incrementally during functional test, i.e., considering each time a greater value for k , and adjusting the pmfs involved correspondingly. In this way, the prediction about the total number of failures for a product as testing proceeds will be more and more precise.

4. Application

The Bemar method has been experimented on the failure data relative to the functional test phase of several products; we have also tried it on some operational test results (for which we expect the model is not working as well as for functional testing). We briefly present the results in sections 4.1 and 4.2, respectively.

4.1 Functional testing

Before applying the Bemar model to the data relative to functional testing, we investigated ways to derive a suitable prior distribution for T .

About these data we knew that the products performed similar functionalities, they had been tested by the same producer and with the same methodology. It was plausible to expect that the test results could exhibit a similar behavior, which would be a useful fact to derive a prior pmf for T .

More in general, it is probable that a software producer has collected similar information about the functional tests developed in the past. In the case that the products exhibit a similar behavior, the information collected (in particular the mean and the variance) can be useful to establish a proper prior pmf of T for the next product that the producer will test.

First of all, analyzing the failure data we noticed that the distance between subsequent FTIs was not greater than 20 and that the maximum number of failures per FTI was 6. Therefore we considered that the variable T could take discrete values within the interval $[1,20]$ and we put $MF=6$.

Then for each project we derived a histogram of the time distance (measured in elapsed test intervals) occurring between two subsequent FTIs. In Figure 2 we report the histogram corresponding to the product shown in figure 1. Analyzing the histograms for this and all the other sets of data available, a certain regularity in the failure behavior under functional testing was in fact noted. This observation would sustain the hypothesis that a general distribution for the distance between two successive FTIs for the functional test process of this producer can be identified.

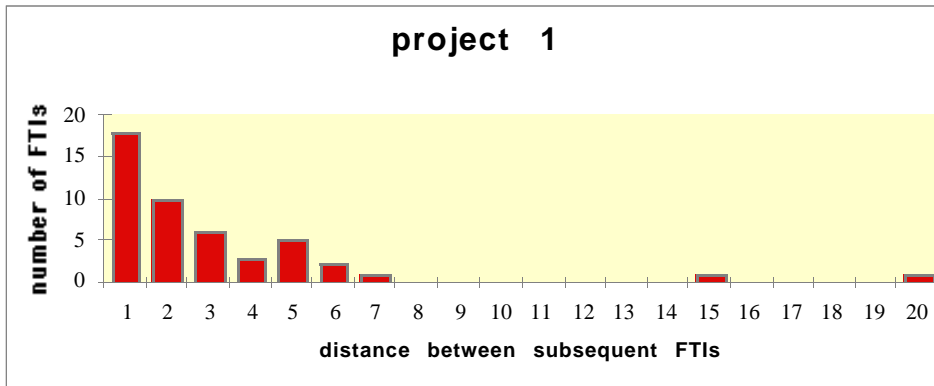


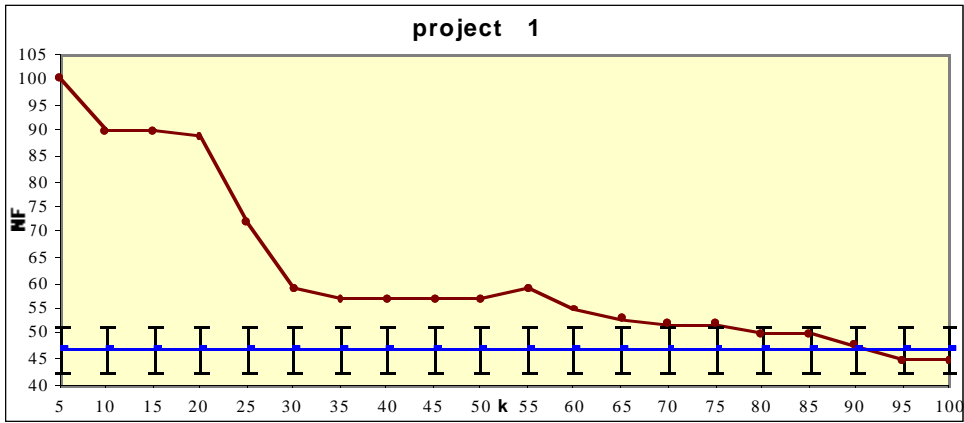
Figure 2: Histogram for the random variable T

In particular, after some analysis, we decided to approximate the prior pmf of T with a *normal truncated distribution*. We derived the normal curve with mean and variance equal to the sample mean and variance, and truncated it between 1 and 20. Since the data we have are grouped within intervals, we then approximated this continuous distribution with a discrete one.

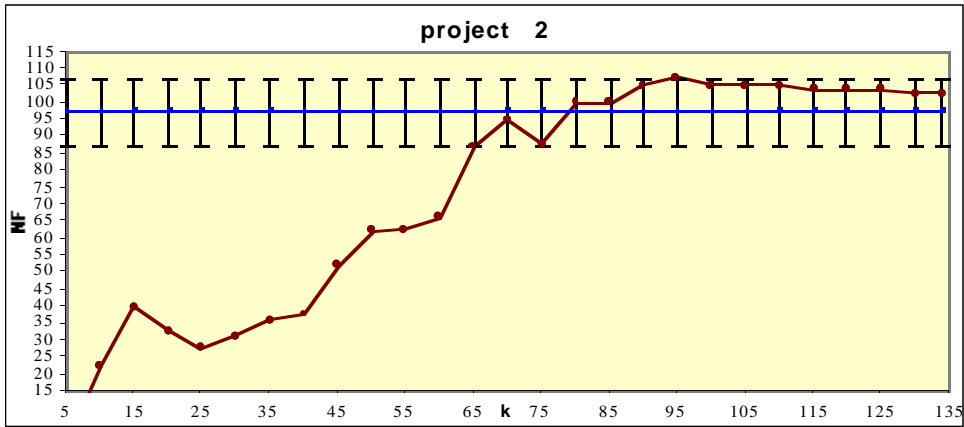
The approach we followed to verify the model was the following. Considering the whole series of test outputs of a product, an intermediate test interval TI_k is taken as the current point. From this point, the cumulative number of failures that will be observed for the whole testing period is estimated applying the Bemar model. For the prediction, therefore, we use the failure data collected from the beginning of the functional test up to the selected point TI_k .

This computation is repeated for progressively longer test intervals (i.e., for greater values of k), for instance after the first 5 TIs, after the first 10 TIs, 15, and so on. In fact, since a Bayesian inference procedure is used, the prediction is progressively updated considering each time a greater amount of collected data.

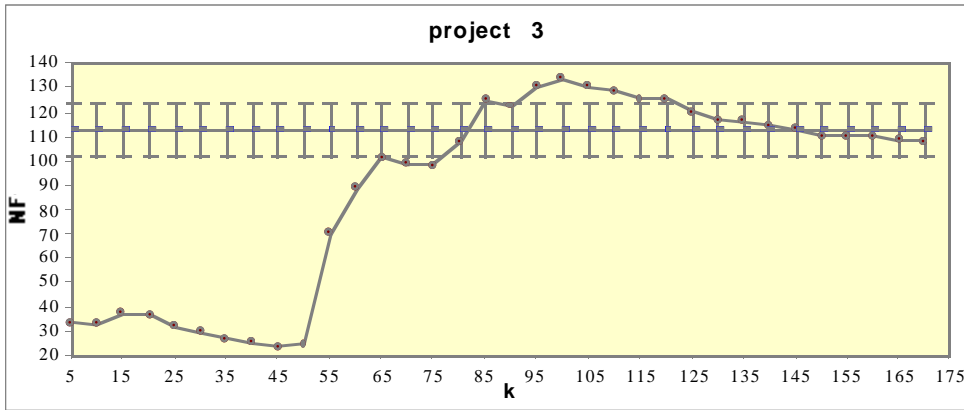
In Figure 3 the results obtained applying the Bemar method to some of the sets of data available are shown. In these figures on the horizontal axis we put the number of test intervals, k , considered to make the prediction, and on the vertical axis the cumulative number of failures, NF , predicted at the end of functional test. The dotted line represents the cumulative number of failures predicted at the end of the functional test using as prior pmf of T the normal truncated distribution. The effect of improvement of the prediction as more test outputs are observed is clearly visible. To compare the results predicted with the real ones, in the figures we drew the actual number of failures counted at the end of the testing (the horizontal line). The strip around the horizontal line marked with vertical segments signs the zone where the relative error of the estimation is below 10%.



(a)



(b)



(c)

Figure 3: Predictions with the Bemar model

In general, for all the case studies considered, we could observe that the model starts with very high errors, but after about a half of the test period, the prediction becomes quite good. We are currently studying other ways to derive a prior pmf for a specific producer from the test result observed in earlier projects. We expect that a prior pmf which fits better to the test process under investigation should converge more quickly to a valid prediction.

4.2 Application to operational test data

We are interested in discovering if and how the Bemar model can be applied as a complementary approach to reliability growth models or in those situations in which the failure data relative to operational testing do not show a reliability increasing trend. For this reason, we also tried our model on some operational test results collected by the same producer during beta testing.

The problem in applying the Bemar model to this kind of data was that operational test results collected previously on similar projects were not available. Therefore we could not apply the criteria described in the previous section for the selection of a prior pmf of T . We hence decided to adopt a uniform prior distribution.

For the rest, the approach to apply the Bemar model to the data collected during the operational phase is the same of that described in Section 4.1: we took an intermediate test interval k as the current point of the operational test, and from this point we predicted the expected final number of failures. This computation has been repeated taking progressively longer periods. We report the results in the figure below.

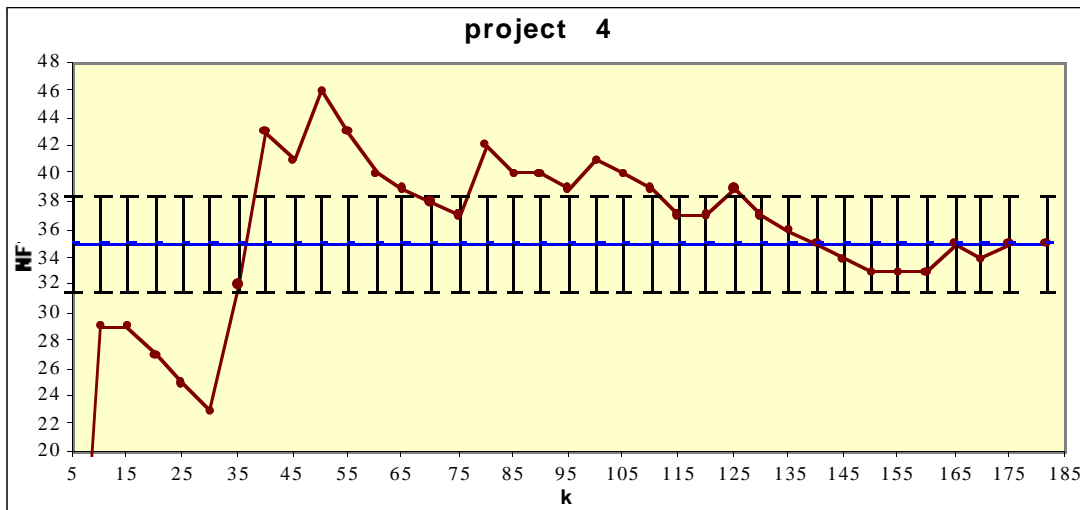


Figure 4: Prediction of the Bemar model for beta testing

The performance of the model becomes acceptable after 110 TIs, over a comprehensive period of 180 TIs. We must add that attempts to apply standard reliability growth models to these same data were not successful; the problem was that the reliability did not regularly increase, as required by those models.

On the contrary, we expect a worse performance of Bemar over data that exhibit consistent reliability growth. We have tried the model on a set of data taken from the literature (Abdel-Ghaly, 1986). These data are reported as execution times in seconds between successive failures. To apply our model, we have grouped the failure data into test intervals of 600 seconds.

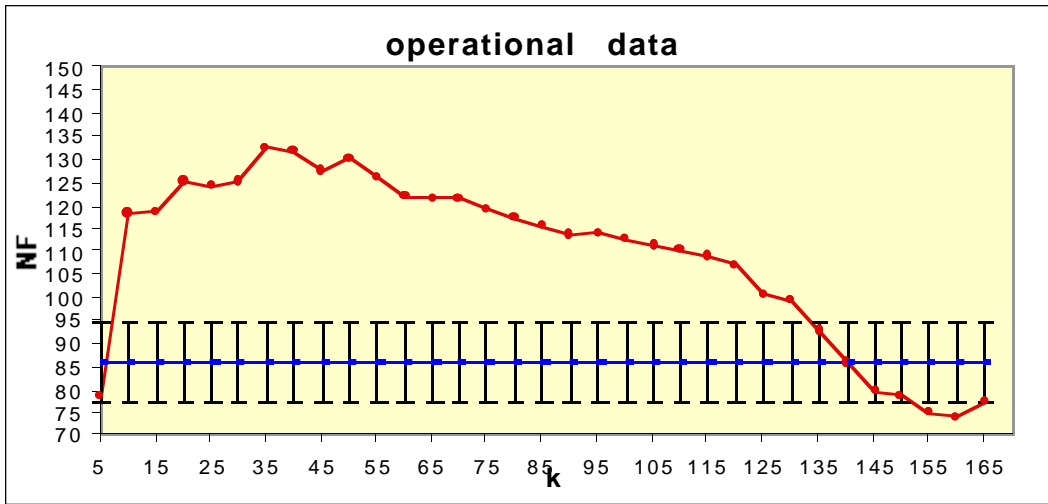


Figure 5: Prediction of the Bemar model for operational testing

5. Conclusions

This work is still in a preliminary stage. We are investigating dynamic models for monitoring and controlling the test process based on observed test results. In this paper we have briefly presented the motivations, the formulation and a few applications of a new model that can be applied to failure data to predict the expected number of failures in future tests. The model is still incomplete, and needs further validation on more data. In particular, the formula used to make the prediction needs to be augmented with some method to estimate in advance the error bound. For the time being, we have evaluated the relative error against known results, and the model performance looks encouraging.

This model assumes that the detected failures are distributed over the whole test period, and that reliability does not exhibit a regular trend. This could be the case for the early test phases, when many failures still remain, and standard reliability growth models cannot yet be applied. In this sense, we believe that this model works in complementary way with reliability growth models, and in fact we intend to investigate an approach to use both models in combination.

Acknowledgements

We thank Emilia Peciola and Gaetano Lombardi of the Research&Development Division of Ericsson Telecomunicazioni S.p.A. in Rome, for providing us with valuable data and information, as well as for useful discussions and interest during the development of this work.

References

A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, Evaluation of Competing Software Reliability Predictions, *IEEE Tr. On Software Eng.*, Vol. SE-12, No. 9, Sept. 1996, pp. 950-967.

A. Bertolino, G. Lombardi, E. Marchetti, E. Peciola, "Introducing a Reliability Measurement Program into an Industrial Context", *Proc. of ESCOM-ENCRESS 98*, Rome, May 27-29 1998, pp. 277-286.

K. Y. Cai "On Estimating the Number of Defects Remaining in the Software", *J. System Software*, Vol. 40, No. 2, pp. 93-114, February 1998.

S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*, The Benjamin/Cummings Publishing Co., Menlo Park, Ca, 1986.

N. E. Fenton and S. L. Pfleeger, *Software Metrics A Rigorous and Practical Approach*, 2nd Ed., Int. Thomson Comp. Press, 1997.

K. Kanoun, M. Kaaniche, and J. P. Laprie, "Qualitative and Quantitative Reliability Assessment", *IEEE Software*, Vol. 14, No. 2, Mach 1997.

M. R. Lyu (Ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

Impact of Sequence-Based Specification on Statistical Software Testing

Stacy Prowell

Q-Labs[®]

5516 Lonas Rd, Suite 110, Knoxville, TN 37909, USA
<http://www.q-labs.com/>
Email: Stacy.Prowell@Q-Labs.com

Primary Benefits

Sequence-based specification contributes to the understanding of:

- Precise test boundary
- Complete input domain
- Valid input sequencing
- Intended software function

Sequence-based specification also helps enforce an external, user-centered view of software function.

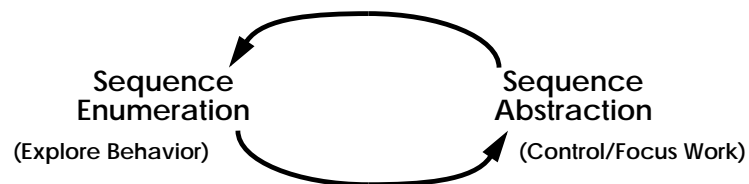
Sequence-Based Specification

Sequence-based specification is a black box specification method.

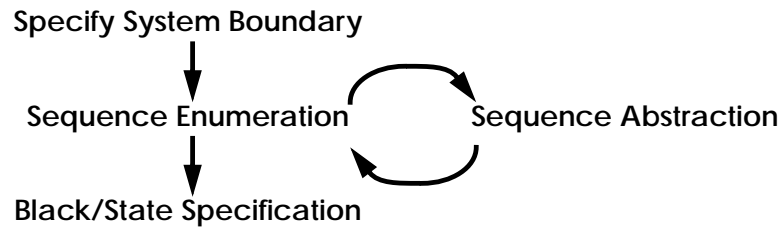


The specification gives an external, implementation-independent “user’s view” of software function.

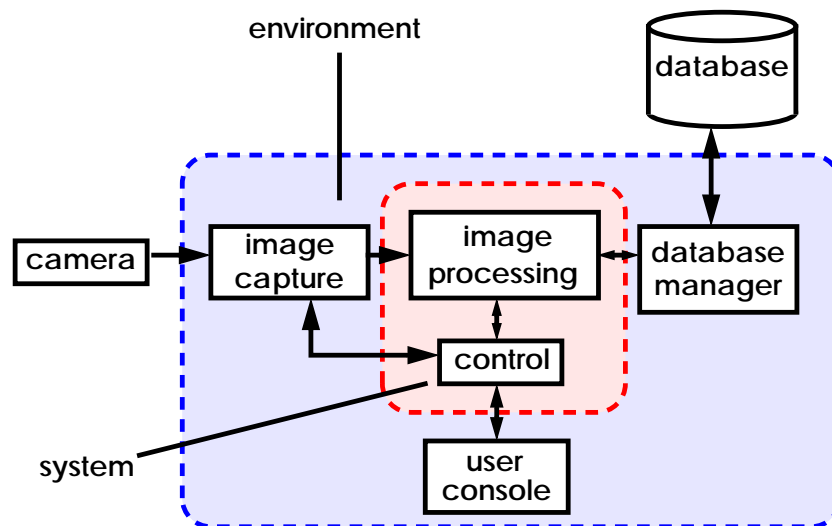
Primary Techniques



Sequence-Based Specification Process



Specify Software Interfaces



Black Box Specification

Issues

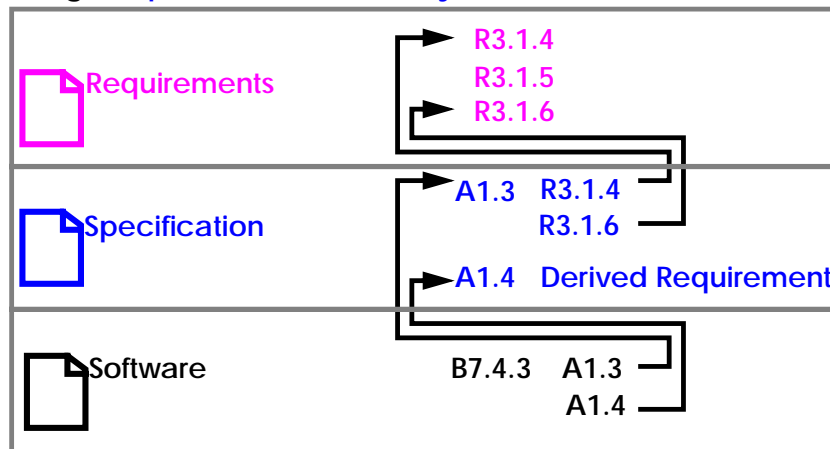
- 1 Software need not generate a response for every stimulus.
clock pulses
- 2 Some sequences may violate the definition of the system or environment, or may violate known design assumptions.
stimuli before invocation
events from deactivated hardware

Solution

R contains two special values: 0 and ω , the **null response** and **illegal**, respectively. These two values allow the black box to be a total function.

Requirements Traceability

The **correctness** of the black box function must be shown through **requirements traceability**.



Sequence Enumeration

Basic Idea: Enumerate stimulus histories in order by length. For each history, give the response.

λ	0
T	t
F	ω
L	ω
TT	ω
TF	f
TL	r
FT	ω
FF	ω
FL	ω
LT	ω
LF	ω
LL	ω

Illegal Prefix

Whenever a sequence has an illegal prefix, the sequence is illegal. Do not expand illegal sequences in the enumeration.

λ	0
T	t
F	ω
L	ω
TT	ω
TF	f
TL	r
TFT	ω
TFF	f
TFL	a
TLT	t
TLF	ω
TLL	ω

Equivalent Sequences

Note that the sequences λ and TL both leave the plane in the initial condition. Thus the future behavior of the system is independent of which of these sequences occurred.

We say two sequences u and v are **equivalent** if and only if, when extended by non-empty histories, the responses always match.

$u \equiv v$ if and only if $\forall (w \in S^*), w \neq \lambda \Rightarrow BB(uw) = BB(vw)$

Basic Idea: When enumerating, note equivalences to previous sequences in the enumeration, as with $\lambda \equiv TL$. We say TL is **reduced** to λ .

Since the sequences agree on future behavior, only extend one. Do not extend sequences which have been reduced.

Complete Enumeration

Noting reductions in the enumeration, we obtain the following.

λ	0
T	t
F	ω
L	ω
TT	ω
TF	f
TL	r / $\equiv \lambda$
TFT	ω
TFF	f / $\equiv TF$
TFL	a / $\equiv \lambda$

There are no sequences left to extend; the enumeration is **complete**, and gives a response for any sequence in S^* .

Example Enumeration

Sequence	Response	Equivalence	Trace	Notes
IN MG HR IN	INA	IN	3.5	
IN MG HR HR	ERR	IN MG HR	4.3.5, 6.1, 6.4	4.3.5 If the SOS receives a command (other than IN) from the GCS during processing of a previous command, a protocol error shall be generated and processing of the previous command shall continue.
IN MG HR OTE	HF	IN MG HR ASN	4.3.3, 4.3.7	4.3.7 The outcome of an HT shall not affect subsequent functionality.

Canonical Sequences

One is only allowed to reduce to previous histories in the enumeration. Every sequence in an enumeration will be equivalent to an unreduced sequence.

There will be one unreduced sequence, called the **canonical sequence**, for each equivalence class of the equivalence relation discovered on S^* .

Abstraction

Basic Idea: **Abstractions** may be used to omit or hide details about histories, resulting in shorter histories.

Formally, a **sequence abstraction** is a mapping ϕ from X^* (**atomic** sequences) to Y^* (**abstract** sequences), such that

- sequences get no longer ($|\phi(u)| \leq |u|$)
- the stimulus ordering is preserved ($\phi(u)$ is a prefix of $\phi(uv)$)

Example

Two stimuli:

$R(n)$ = User requests a connection

$A(n)$ = Connection request accepted

A system n is "connected" if there has been a request followed by an accept.

Example

Define an abstract stimulus $C(n)$ formally as follows.

$$p_C(\lambda, n) = \text{false}$$

$$p_C(ux, n) = \begin{array}{l} \text{true if } x = A(n) \text{ and } p_C(u, n) = \text{false and} \\ \text{\ } u \text{ contains an } R(n) \end{array}$$

$$p_C(ux, n) = \text{false otherwise}$$

The atomic sequence

$R(7) R(5) A(3) A(5) R(4) R(9) A(4) A(7)$

is mapped to abstract sequence

$C(5) C(4) C(7)$.

Working with Abstractions

Basic Idea: Enumerate with the abstract stimuli.

One enumerates to discover system behavior. Thus, one may not know enough about an abstraction to specify it formally.

In this case, give an initial, informal definition of the abstraction and enumerate using the abstract stimuli.

Based on the behavior discovered, create a formal definition of the abstraction.

Writing the Specification and Transformation to State Machine

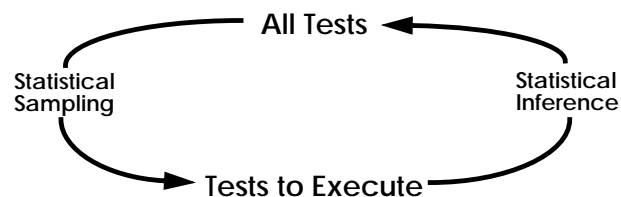
A state machine can be generated from the enumerations, specification functions, and abstraction definitions—the **enumeration state machine**.

The enumeration state machine gives intended software response for a known state and input.

States are distinguished by future behavior (responses).

Statistical Software Testing

Statistical software testing is the application of statistical science to software testing problems.



Statistical testing must be a well-defined procedure, performed under specified operating conditions.

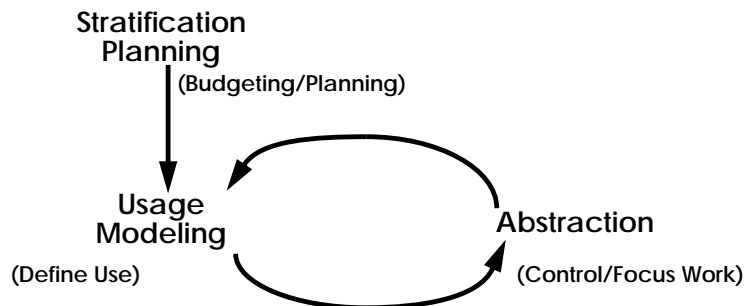
Statistical Software Testing

Statistical software testing is primarily a black box testing method.

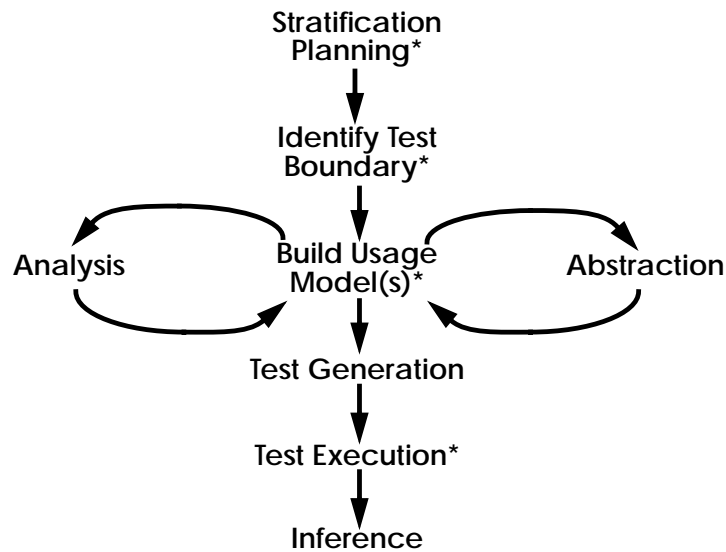


A usage specification gives an external, "user's view" of software use.

Primary Techniques



Statistical Software Testing Process



23 (37)

Statistical Software Testing

Implementation knowledge may lead to bias in testing.

Focus on **states of use**, not software states.

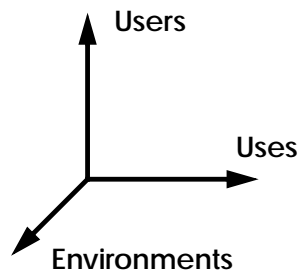
External view in testing is aided by external specification.

Requirements traceability of specification supports

- Evaluating requirements coverage for testing
- Evaluating testability of requirements
- Generating non-random tests for particular requirements

24 (37)

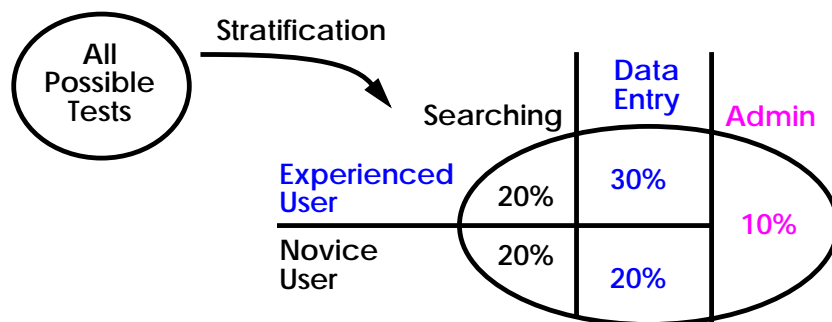
Stratification Planning



A **stratum** is a combination of user, use, and environment.

- Definition of use must include initial and final states.
- Final states must be verifiable.

Stratification Planning



Initial state: software uninvoked
 Final state: software terminated

What are all the conditions under which the software should terminate? These conditions are given by the specification.

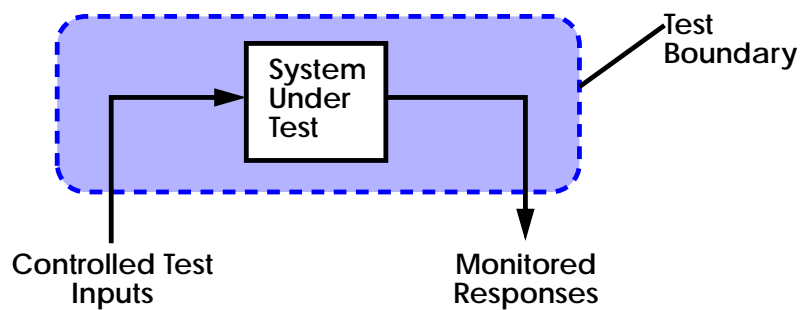
Stratification Planning

The complete and consistent nature of the specification reveals classes of use which might otherwise be missed or insufficiently tested:

- Error conditions
- Critical use
- Unexpected use

This analysis is based on user-perceived function and risk.

Identify Test Boundary



Identify and understand all interfaces to be cut during testing.

- Drive stimuli
- Monitor responses

Identify Test Boundary

Sequence-based specification's deterministic model helps avoid:

- Undocumented inputs
- Misunderstood interfaces

while providing:

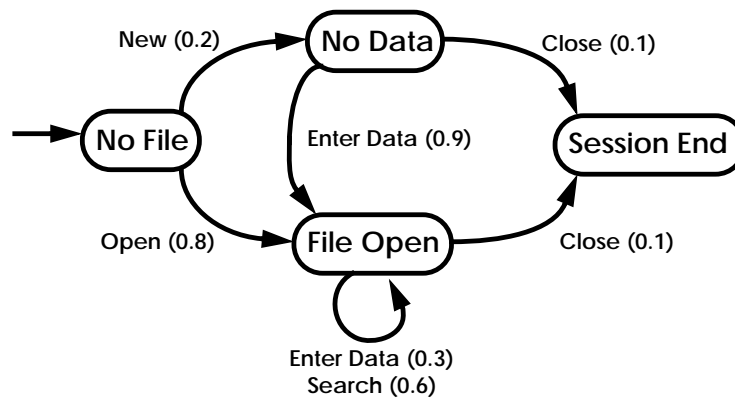
- Planning for test drivers / monitors
- Evaluation of interface testability
- Evaluation of requirements testability

Usage Modeling

Expected use can be modeled as a finite-state, discrete parameter Markov chain.

- Whittaker and Thomason (1993) proposed the Markov chain as the model for software use.
- Walton (1995) explored optimization of model probabilities given constraints.
- Gutjahr (1997) explored acceleration of testing by modifying the chain probabilities and weighting the results.

Usage Modeling



Usage Modeling

A usage model is essentially a state machine whose transitions have an associated probability distribution.

States are distinguishable by future use (stimuli)

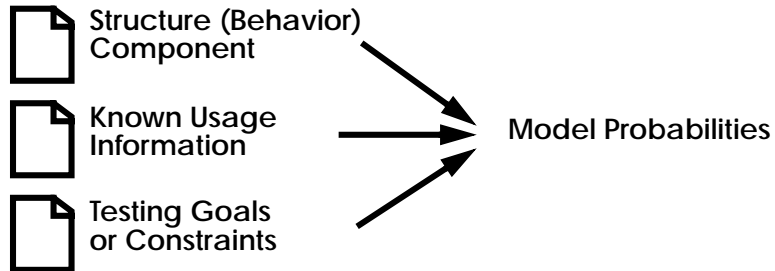
Different states from the specification state machine typically correspond to different states of use.

The specification state machine may be used as the “first cut” of a usage model. Testers proceed by:

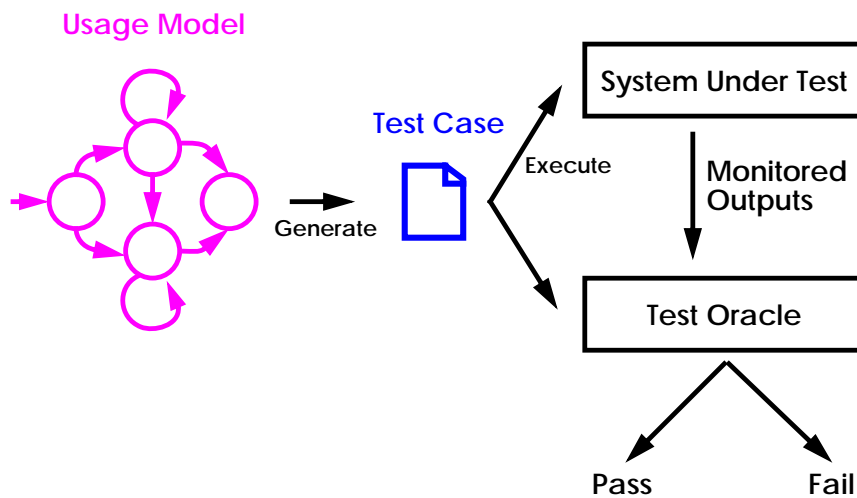
- Combining states to reduce model size
- Splitting states to reflect pure usage factors

This changes the focus from constructing the model to refining the model.

Usage Modeling



Test Execution



Test Oracle

The specification state machine can be used as the test oracle.

The derivation of the usage model from the specification state machine enforces the correspondence.

Two issues:

- **Abstraction:** The oracle may not compute all abstractions.
- **Boundary:** The test boundary may differ from the system boundary; some unavoidable non-determinism may be introduced.

Summary

The use of sequence-based specification has several impacts on a statistical software testing phase.

Direct benefits:

- Communication reduced by precise specification
- Requirements traceability simplifies test case crafting
- Precise system boundary reduces start-up time
- Reuse of specification state machine reduces model construction effort
- Reuse of specification state machine as test oracle reduces cost of automated testing

Summary

Indirect benefits:

- External focus reduces implementation bias
- Strict definition of interfaces improves testability analysis
- System understanding is improved for all practitioners

IMPACT OF SEQUENCE-BASED SOFTWARE SPECIFICATION ON STATISTICAL SOFTWARE TESTING

S. J. PROWELL

ABSTRACT. The combination of sequence-based software specification with statistical software testing yields direct benefits of reduced start-up and communication overhead and the potential for automated generation of initial usage models and test oracles. Indirect benefits include better developer and system engineer understanding of external usability issues and an emphasis on external events which supports evaluating testability of requirements. This paper introduces the sequence-based specification techniques of sequence enumeration and sequence abstraction, then proceeds to trace the impacts of sequence-based specification on the development of test plans, usage models, and testing oracles.

1. MOTIVATION

Statistical testing of software based on a usage model requires that test engineers precisely define the test boundary, extract the expected usage profile, legal inputs, potential outputs, valid sequencing of inputs, and expected software responses to given input sequences. Extracting these details requires considerable communication between testing team members and both software developers and systems engineers. Misunderstandings about fundamental details of the software's input space and intended function often result in significant revision of software test plans and usage models. Further, software developers and systems engineers may communicate specialized knowledge of the software's implementation which may

Key words and phrases. Specification, sequence-based specification, statistical testing, usage models.

Dr. Prowell is with Q-Labs, Inc., 5516 Lonas Road, Suite 110, Knoxville, TN 37909, USA. Email: Stacy.Prowell@Q-Labs.com.

bias software testing. Often such knowledge rests on the assumption that the software system is correctly implemented (for example, the claim that some behavior is “impossible” because the software is intended to make it so), and may therefore hinder the utility of the software testing phase.

The introduction of sequence-based software specification can directly reduce the start-up costs incurred during testing by requiring that the software’s input domain, valid input sequences, and intended response for each sequence be considered early. Though additional costs are incurred during the specification phase, all subsequent phases of software development and testing benefit from the early introduction of a detailed, external specification. This reduces start-up times and communication overhead for all phases by helping identify and confront potential risks early in the process, and pays dividends in software testing.

2. SEQUENCE-BASED SPECIFICATION

Sequence-based specification is a collection of techniques for rigorous development of an external, functional specification of a software system’s intended behavior, known as a *black box specification* [2]. The development of this black box specification is guided by the technique of *sequence enumeration*, which is the literal enumeration of sequences of inputs and the assignment of correct responses to each enumerated sequence. Sequence enumeration is controlled and focused through the development of *sequence abstractions*, which allow practitioners to change their view of the system’s inputs and outputs without losing the formal nature of the resulting specification.

Sequence-based specification proceeds as follows:

1. The system boundary is specified by defining all software interfaces as precisely as possible.
2. All direct inputs to the system (stimuli) are identified.
3. Sequences of stimuli are enumerated, and a response is given for each sequence. All choices are justified by tracing to either

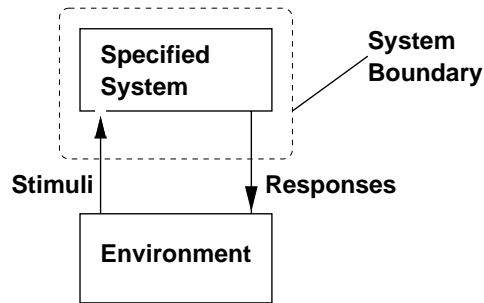


FIGURE 1. System boundary and environment

the software requirements, or by the introduction of a derived requirement.

4. Abstract stimuli are defined and introduced into the enumeration process as necessary to control growth, hide details, or refocus the enumeration activity.
5. From the enumerations, a black box specification is constructed and refined to the appropriate level of detail.

This section contains a short review of sequence-based specification. More detail may be found in [2], with full details and a case study in [3]. A more extensive case study will be available in [4]. Proofs of the theorems presented in this section can be found in [1].

2.1. Identification of the System Boundary. The system boundary lists all system interfaces, and should be documented in an interface specification. The collection of external entities with which the system communicates directly makes up the system's *environment*. Events which occur in the environment, and which can affect system behavior are *stimuli*. Events which occur in the system and which are observable from the environment are *responses*. Note that this definition of stimulus and response includes changes to monitored and controlled variables. The system boundary is depicted in fig. 1.

Work on a black box specification cannot begin until an initial list of stimuli and responses is in hand, though this list may be revised as work proceeds. In fact, the process of sequence enumeration, based as it is on a deterministic view of the system, may reveal problems in

the system boundary definition, resulting in revisions of the interface specification.

2.2. The Black Box Function. Let the set of all system stimuli be denoted S , and let the set of all system responses be denoted R . The black box specification denotes a complete function $BB : S^* \rightarrow R$, where S^* denotes all finite-length sequences of stimuli, which are interpreted as stimulus histories from left to right. Each sequence of stimuli is mapped to a unique value in R , which is the intended response to the most recent stimulus in the sequence. Software in operation might emit a response for every stimulus in a sequence, but the black box specification identifies only the last response.

Throughout the rest of this paper, let λ denote the empty sequence, and let sequence concatenation be denoted by juxtaposition, so the concatenation of $u \in S^*$ onto the left end of $v \in S^*$ is denoted uv .

The mapping rule of the specification requires that all sequences be assigned a unique value from R , but some sequences may not be physically realizable for software in operation because they violate the definition of the system. Consider a sequence which has several stimuli prior to the software being invoked for the first time. Such a sequence cannot be observed by the software. Such sequences are called *illegal*, since they violate the system definition. To account for these sequences, R is extended to include a special value ω , to which all illegal sequences are mapped. Software in operation need not generate an externally-visible response for every stimulus sequence, but the mapping rule requires a value from R . To accommodate these sequences, R is further extended to include a special "null response" denoted 0 and interpreted as "no externally-visible response."

2.3. The Sequence Enumeration Method. Sequence enumeration is a technique for discovering the black box function for a system

through the literal enumeration of all sequences of stimuli. In sequence enumeration, practitioners generate sequences from S^* in order by length, and then according to some fixed ordering among sequences of a particular length. As each sequence is generated, practitioners assign some value from R for the sequence, tracing the choice to software requirements or documenting derived requirements as necessary to justify their decisions. This systematic process increases developer understanding of the system's external behavior.

During sequence enumeration, practitioners may discover that all extensions of one sequence are mapped to the same response as the identical extensions of the other sequence. Formally, two sequences u and v are (*Mealy*) *equivalent*, denoted $u \equiv_{\rho_{Me}} v$ if and only if $\forall w \in S^*, w \neq \lambda, BB(uw) = BB(vw)$. If, in an enumeration, the current sequence u is equivalent to a previously-enumerated sequence v , then practitioners note both the value $BB(u)$ and the equivalence to previous sequence v . In this case, u is *reducible* to v . If there are no previous sequences to which u may be reduced, then u is *irreducible*.

Henceforth, the mapping of sequence u to response $r \in R$ in an enumeration will be denoted $u \mapsto r$. If u is found to be reducible to previous sequence v , then this is denoted $u \mapsto r / \equiv v$.

Theorem. [*Canonical Sequence*] Let $u \in S^*$ be a sequence. Then there exists a unique sequence (the "normal form") $v \in [u]_{\rho_{Me}}$ which is irreducible.

The unique normal forms for the reduction system defined by the equivalence ρ_{Me} are called *canonical sequences*. Since equivalence relations are transitive, and since every equivalence class has a canonical sequence, every reduction in an enumeration is required to be to an unreduced sequence.

Enumerations are generated by extending sequences of length n to obtain all sequences of length $n + 1$. If a sequence is illegal ($u \mapsto \omega$), then all sequences with prefix u are also illegal, and thus illegal sequences need not be extended in an enumeration. If a sequence is reduced to previous sequence v , then for any sequence uw ($w \neq \lambda$), $BB(uw) = BB(vw)$. Thus sequences which have been reduced need

not be extended in an enumeration. It follows that only unreduced, legal sequences need to be extended. If no sequences of some length n need to be extended, then the enumeration is *complete*.

Theorem. *A complete enumeration specifies a response for every sequence $u \in S^*$.*

A sequence enumeration thus defines a complete, consistent black box specification for a software system.

Every enumeration can be viewed as a Mealy machine by taking the canonical sequences as states. This leads to a direct conversion of the enumeration, and thus the black box, into a state machine.

2.4. Controlling Enumeration Through Sequence Abstractions. Sequence enumeration is made practical through the application of abstraction techniques, which can be used to control growth, defer details without losing them, and change views of the system. A sequence abstraction is a complete mapping $\phi : X^* \rightarrow Y^*$ from *atomic* (stimulus) sequences X^* to *abstract* (stimulus) sequences Y^* such that the mapping satisfies two properties:

- abstract sequences are never longer than corresponding atomic sequences: $\forall u \in X^*, |\phi(u)| \leq |u|$, and
- the order of the stimuli in the sequence is preserved: $\forall u, v \in X^*, \phi(u)$ is a prefix of $\phi(uv)$.

There are several common forms of abstractions. Some of these “abstraction patterns” are considered in [1] and [4]. The process by which one defines abstractions is beyond the scope of this paper. Fig. 2 illustrates the idea of abstractions. Here a sequence of request connection and accept connection request events, denoted $R(\lambda)$ and $A(\lambda)$, respectively, is transformed into a sequence of connect events, denoted $C(\lambda)$.

Abstractions may be used as follows, in conjunction with enumeration:

1. Enumerate sequences at the lowest reasonable level of abstraction.

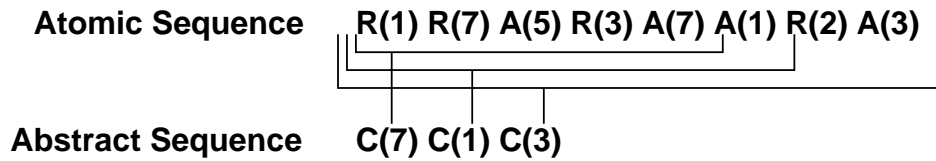


FIGURE 2. An abstract view of a sequence

2. If work stalls or is found to be unproductive, replace atomic stimuli with abstract stimuli to resolve the difficulty and restart enumeration with the abstract stimulus set.
3. Continue to invent abstract stimuli and enumerate until a complete enumeration is obtained, or until system behavior is sufficiently understood to write a complete, consistent black box specification at some level of abstraction.

3. STATISTICAL TESTING OF SOFTWARE AND THE IMPACT OF SEQUENCE BASED SPECIFICATION

Statistical testing refers to the application of statistical science to software testing, and may involve both random and non-random methods. The particular kind of statistical testing to be discussed here involves the generation of software test cases from a stochastic model representing the known or expected usage of a software system (a *usage model*). The stochastic model used will be a finite-state, discrete parameter, irreducible Markov chain. The application of such a model to the testing of software has been widely discussed (for example [5], [6], and [7]). For this reason, the presentation of statistical testing here will be terse.

As with the system specification, the intent is to have an external, black box view of the software to the extent possible. Knowledge of the internal implementation of a system is a source of bias in the statistical experiment, and can result in failure to test portions of the system, or in the direction of significant effort to seldom-used, non-critical functionality. Because the software specification is developed with an external, implementation-independent view, this focus on external events is easier to achieve. Additionally, it provides a means

for discussions between developers and test practitioners while reducing the risk of introducing an implementation bias.

3.1. Test Planning.

3.1.1. *Test Boundary.* Software testing can be viewed as a statistical experiment. One cannot run all possible tests, therefore only some sample of the potential tests will be run, and a conclusion about the reliability of the software in operation use must be derived from the software's performance on this sample. Treating testing as a statistical experiment implies that testing must be a well-defined procedure performed under specified operating conditions in order to generate useful, accurate conclusions.

The system under test must be understood and isolated by the definition of a *test boundary*. The test boundary consists of the precise definition of all interfaces with the system under test. This is similar to the system boundary, with the following exceptions: for every interface which is cut, all stimuli must be controllable, and all responses must be observable, throughout the test. For this reason, practitioners may choose a test boundary which includes some components which are in the system's environment. The test boundary is illustrated in fig. 3.

The definition of the test boundary, which includes the software's interfaces, is crucial to the rest of testing. Inputs to the system which are unknown to the test team may escape testing; vague or incorrect interface descriptions will result in long discussions with developers (who may argue for or against testing certain parts of the system, because of their prior assumptions about system reliability) and in wasted effort due to misinterpreting the interface specifications.

Sequence-based specification requires the development and maintenance of a complete system boundary definition. The process of enumeration and the deterministic nature of the specification force developers to insure that external information sources critical to the production of software responses are documented. This insures the

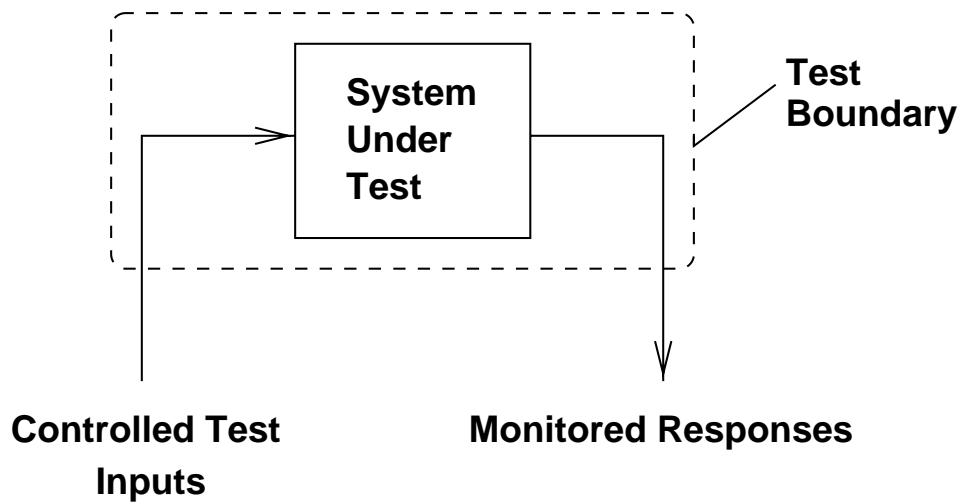


FIGURE 3. Test boundary

persistence of valid interface specifications which may be made available to test practitioners. Further, precise interface descriptions reveal what system stimuli and responses can be controlled and monitored effectively during a test. The availability of a precise interface description serves both to reduce communication overhead between test practitioners and developers, and focuses the discussions on the external system interfaces, thus reducing the chance of introducing an implementation bias into the testing phase.

The most significant advantage gained from such an interface specification is the ability to effectively plan for automation. Since interfaces are precisely specified, test drivers and monitors may be developed in parallel to the software development effort. In this way start-up time, and thus cost, for software testing is significantly reduced.

During test planning, one must also consider requirements coverage. Sequence-based specifications directly support requirements traceability through sequence enumeration. It is thus possible to identify all input sequences which implement a given requirement. This allows assessing whether the requirement is testable given the

external view and chosen test boundary. The ability to quickly transform a requirement into a collection of sequences also provides for the use of non-random, crafted test cases to address specific requirements.

3.1.2. *Test Stratification*. The testing effort may be divided and focused along three dimensions: types of users, types of uses, and environments of use. A combination of these three dimensions which is valid for software testing is called a *stratum*. Identification of test strata and the assignment of testing budget and effort to each identified stratum is *stratification planning*. The idea of test planning is illustrated in fig. 4.

For the purpose of stratification planning, a *user* is any source of stimuli to the system under test, and these may be deduced from the interface specification. A *use* is any single instance of software usage, and may be defined in terms of specific start and finish events, number of commands, duration of test, or simply by specifying starting and terminating conditions. A use always begins in some initial state (perhaps one of several), and ends in some defined final state (perhaps one of several). The initial and final states of any such use must be defined and *must be verifiable*. If a test is intended to leave the system under test in some particular state (such as terminated, with temporary files deleted) but instead leaves the system in some different state (unterminated, or with temporary files not deleted), then the test must be counted as a failure [8].

The identification of appropriate initial and final states is greatly aided by access to a consistent, complete specification. As a simple example, consider the definition of the initial state to be "uninvoked," and the definition of the final state to be "terminated." Under what conditions does the software terminate? This can be found immediately by examining the specification.

Examination of the specification will reveal important classes of use which might otherwise be missed in testing, or tested insufficiently. For example, it is common to discover previously-unconsidered

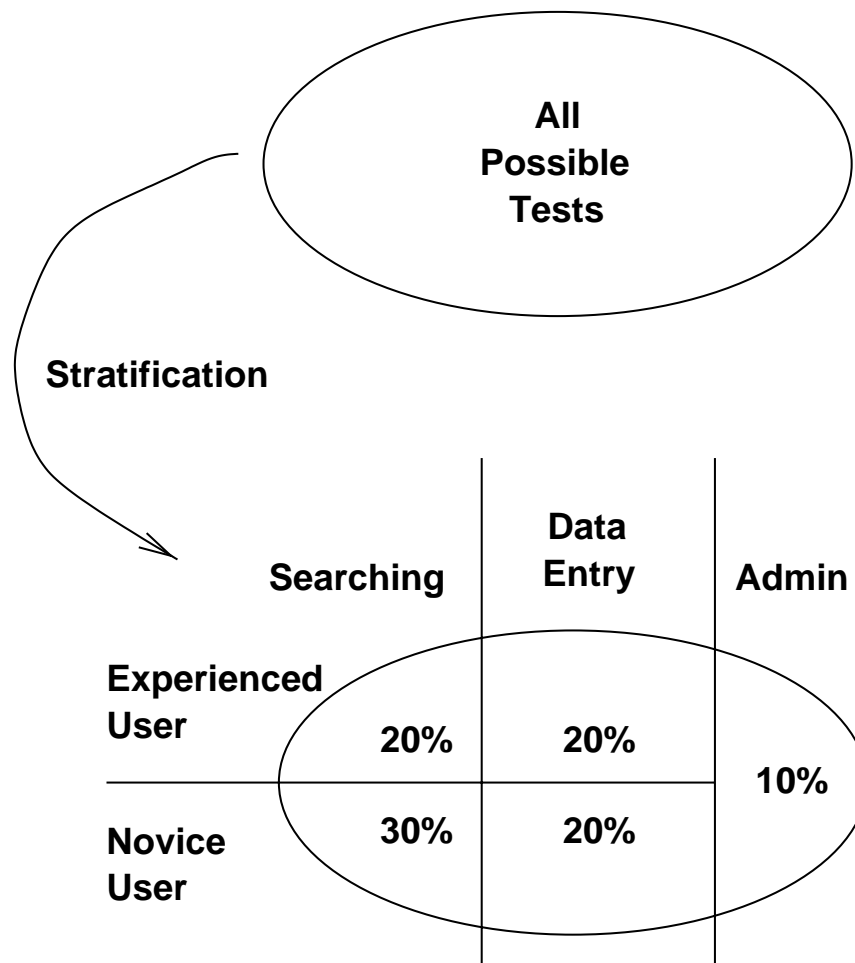


FIGURE 4. Stratification

critical behavior (such as error conditions) as one develops a sequence enumeration. The resulting specification is passed to testing, and additional testing effort can be directed to this behavior. This redirection of test effort occurs as the result of examining an external behavioral specification and not due to knowledge of internal implementation, and is thus based on user-perceived function and risk.

3.2. Modeling Expected Software Usage. Software use will be modeled as a finite-state, discrete parameter, irreducible Markov chain. Such a model is essentially a finite state machine whose transitions

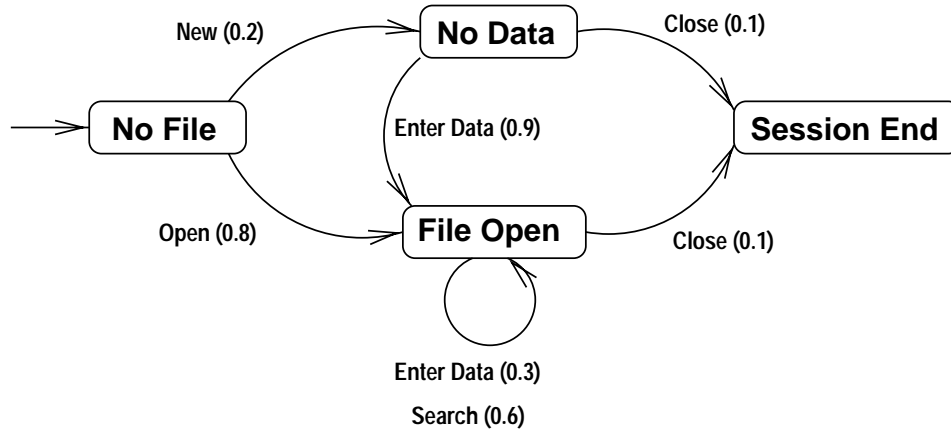


FIGURE 5. A Markov chain usage model

are labelled with stimuli and have associated probabilities. A graphical view of such a model is given in fig. 5, with initial state “No File.” States in this model correspond to *states of use*. Let any path in the chain which begins with the initial state be a *test trajectory*. For example, one test trajectory for the model in fig. 5 is “Open, Search, Search, Enter Data, Close” with probability 0.00864. Any test trajectory which terminates in a final state is therefore a *test case*, thus the trajectory just discussed is also a test case, since “Session End” is the final state. Let ρ_U be a relation on the set of test trajectories S^* , and let $p(u)$ denote the expected probability in use of test trajectory u . Two test trajectories u and v are *usage equivalent*, denoted $u \equiv_{\rho_U} v$ if extensions are always approximately equally likely. In short, the likelihood of future use is independent of which sequence has been observed.

Formally, choose some small $0 < \epsilon < 1$ and break the interval $[0, 1]$ into $[0, \epsilon), [\epsilon, 2\epsilon), \dots, [n\epsilon, 1]$. Then two values are ϵ -equivalent, written $p \approx_{\epsilon} q$ if and only if they fall in the same interval. Equivalence of test trajectories is given by: $u \equiv_{\rho_U} v$ if and only if $\forall w \in S^*, p(uw) \approx_{\epsilon} p(vw)$. Note that this relation is an equivalence relation, and we may use it to deduce a state space from S^* . Formally, a *state of use* is an element of S^* / \approx_{ϵ} .

This kind of trajectory equivalence is similar to the notion of sequence equivalence used in sequence-based specification. While there is no simple containment of one in the other, there is a weak relationship which can be exploited. A response $r \in R$ may change the potential input domain of the software (perhaps by opening or closing a window, or turning on or off some external hardware), and clearly two sequences which have differing future input domains are unlikely to be usage equivalent. Thus if two sequences are not Mealy equivalent, then they are unlikely to be usage equivalent. It follows that different states of the Mealy machine derived from an enumeration are likely to correspond to different states of use. Thus test practitioners may take the Mealy machine as the first cut of a usage model.

The ability to automatically generate an initial usage model from the software specification is a tremendous advantage to test practitioners. The development of a usage model requires considerable knowledge of both the software's legal sequences and generated responses, both of which are directly encoded in the Mealy machine. Practitioners may then modify the machine by combining states to reduce test overhead and model size, or by splitting states based on usage criteria not considered in the specification (such as time of day, type of usage, whether a file has just been printed, etc.). Thus the test practitioners are able to quickly shift their focus to optimizing their efforts to achieve test goals.

3.3. Execution of Tests and the Role of Oracles. The role of a test oracle is illustrated in fig. 6. The Mealy machine additionally encodes the associated software response, and thus can serve as a test oracle (up to the level of abstraction used in the specification). As sequences of inputs are generated for input to the software, one can use the Mealy machine to predict the appropriate response. This response may then be checked against the response generated by the software.

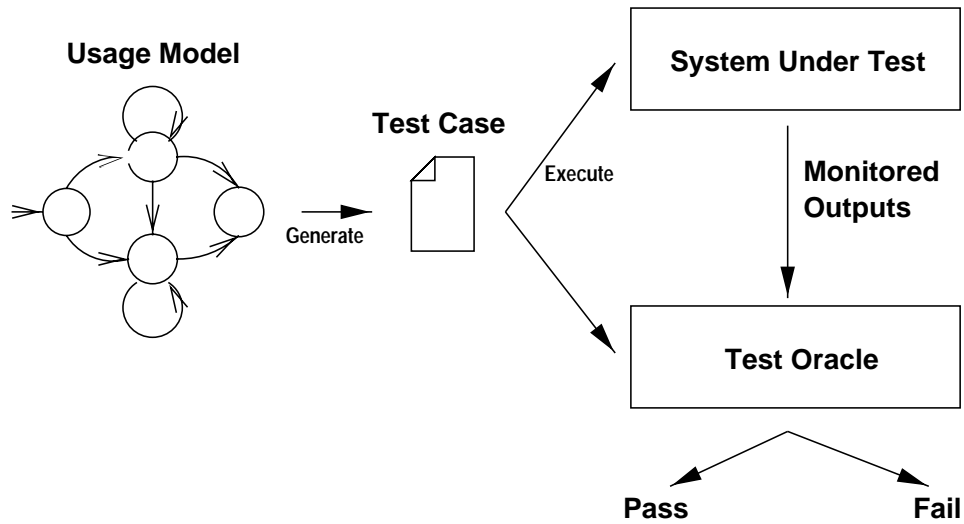


FIGURE 6. Test oracle

Two factors complicate the use of the Mealy machine as an oracle. First, abstractions may be used in the development of the enumeration, and thus in the Mealy machine. These can be removed through function composition, as described in [3]. One can implement the abstractions and, given the sequence of real inputs passed to the software, determine the appropriate abstract sequence. The Mealy machine will then reveal the abstract response, which can be mapped to the particular atomic response.

Second, and more significantly, the test boundary and the system boundary may differ. As a result, some software inputs needed to determine a particular response given the Mealy machine may be unavailable to the oracle. One solution is the following. Transitions in the Mealy machine which correspond to such unavailable stimuli are changed to λ -transitions (i.e., transitions which may be taken nondeterministically). Thus for any sequence executed against the software, the sequence prefix reveals that the software must be in one of a collection of states. The final stimulus is then used to determine all potential responses. Note that, because of the nondeterministic nature of this oracle, it is possible that failures will not be

detected, as pointed out in [9]. Note that, because some stimuli are lost, no other solution is possible.

4. CONCLUSIONS

Aside from the obvious advantages to test practitioners of a any software specification the development of a sequence-based specification has the following direct benefits to software testing:

- The precise nature of the specification reduces communication overhead by conveying the software's intended behavior unambiguously
- Requirements traceability, maintained throughout the sequence enumeration process, simplifies the construction of crafted test cases to address specific requirements by identifying the sequences which implement a particular requirement
- The precise system boundary definition reduces start-up time and provides early opportunities to assess and develop test drivers and scaffolding
- Using the Mealy machine from the enumeration as an initial Markov model structure reduces Model construction time
- The Mealy machine from the enumeration serves as a test oracle to support automated testing, reducing the overall cost of automating software testing

These benefits all serve to either reduce the effort required during software testing, or to improve management of the testing process.

Software testing benefits indirectly from the development of a sequence-based specification in the following ways:

- The focus on external events reduces the chance of implementation bias during testing, providing improved experimental control
- The strict definition of software interfaces allows improved assessment of software testability and appropriate test boundary
- Understanding of the system is improved for developers, systems engineers, and test practitioners

The application of sequence-based specification on industrial projects has resulted in the development of new sequence-based practices and the extension of existing practices to meet the needs of specific software domains, including real-time software, distributed systems, and dynamic network environments. These additions to the theory will have implications with respect to software testing which should be explored further. For example, one common extension of the basic enumeration practice is to allow stimuli to interrupt the production of a response. This has implications with respect to testability which must be considered in a software test plan.

Abstractions are used in the development of Markov chain usage models to control the growth of the state space. The relationship of these abstractions to the abstractions defined during analysis of software behavior should be explored. At present, it seems that abstractions developed during specification are often equally useful during development of a usage model.

The use of the enumeration Mealy machine as a test oracle requires further development. In particular, the problem of how to deal with nondeterministic behavior during testing must be addressed. If the communication which is obscured is with a component which receives no stimuli from other external systems, and if a sequence-based specification is available for the component, then the component may be modeled by a second Mealy machine, and the nondeterminism eliminated. This would increase the accuracy of the test oracle, and thus yield a more effective software test.

REFERENCES

- [1] S.J. Prowell, "Sequence-Based Software Specification," (Dissertation), University of Tennessee, Knoxville, Tennessee, 1996.
- [2] S.J. Prowell and J.H. Poore, "Sequence-Based Software Specification of Deterministic Systems," *Software—Practice and Experience*, v. 28, n. 3, March 1998, pp. 329-344.

- [3] S.J. Prowell, J.C. Martin, C.J. Trammell, J.H. Poore, *Cleanroom Software Engineering: A Practitioners Guide* (version 2.0), Knoxville, Tennessee: Q-Labs, Inc., 1998, Chapter 3.
- [4] S.J. Prowell, C.J. Trammell, R.C. Linger, J.H. Poore, *Cleanroom Software Engineering: Technology and Process*, Reading, Massachusetts: Addison-Wesley-Longman, to appear fall 1998.
- [5] J.A. Whittaker and M.G. Thomason, "A Markov Chain Model for Statistical Software Testing," *IEEE Transactions on Software Engineering*, v. 20, n. 10, October 1994, pp. 812-824.
- [6] G.H. Walton, J.H. Poore, and C.J. Trammell, "Statistical Testing of Software Based on a Usage Model," *Software—Practice and Experience*, v. 25, n. 1, January 1995, pp. 97-108.
- [7] W.J. Gutjahr, "Importance Sampling of Test Cases in Markovian Software Usage Models," *Probability in the Engineering and Information Sciences*, v. 11, 1997, pp. 19-36.
- [8] C.J. Trammell and J.H. Poore, "Experimental Control in Software Reliability Certification," *Proceedings of the 7th Annual NASA/Goddard Software Engineering Workshop*, November 1994, reprinted in *Cleanroom Software Engineering: A Reader*, C. J. Trammell and J. H. Poore, eds., Cambridge, Massachusetts: Blackwell, 1996.
- [9] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite State Models," *IEEE Transactions on Software Engineering*, v. 17, n. 6, June 1991, pp. 591-603.

Quality Week Europe
9 - 13 November 1998, Brussels, Belgium

Evolutionary Testing of Temporal Correctness

Matthias Grochtmann, Joachim Wegener

Daimler-Benz AG
Research and Technology
Alt-Moabit 96 a
D-10559 Berlin
Germany

Tel.: +49 30 39982-{229,232}

Fax: +49 30 39982-107

Matthias.Grochtmann@dbag.bln.daimlerbenz.com

Joachim.Wegener@dabg.bln.daimlerbenz.com

Evolutionary Testing of Temporal Correctness

Overview

- **Introduction**
- **How to Use Genetic Algorithms for Testing Temporal Behavior**
- **Comparing Random and Evolutionary Testing**
- **Comparing Systematic and Evolutionary Testing**
- **Test Strategy for Real-Time Systems**
- **Summary**

Objectives of Testing Real-Time Systems

- Finding errors
 - in functional behavior and
 - in **temporal** behavior of the system under test
- building up confidence in the correct functioning of the test object by executing the test object with selected inputs.



- ➔ Additional requirements for timeliness, simultaneity, predictability, and reliability of computations, and
- ➔ technical system characteristics such as the close interaction with the application environment, the development in host-target environments, as well as parallel and distributed system architectures
- ➔ increase expenditure on and complexity of the test.

Testing Functional System Behavior

- Functional test methods such as the classification-tree method are well-suited.
- Structure-oriented test procedures which require an instrumentation of the test object cause probe effects.

Testing Temporal System Behavior

- There is a lack of appropriate test procedures.
 - ➔ The tester goes back to conventional test methods.
 - ➔ Arising methodological shortcomings shall be compensated for by more or less heuristic tests in worst-case scenarios.

What does the demand for methodological support to testing temporal behavior mean?

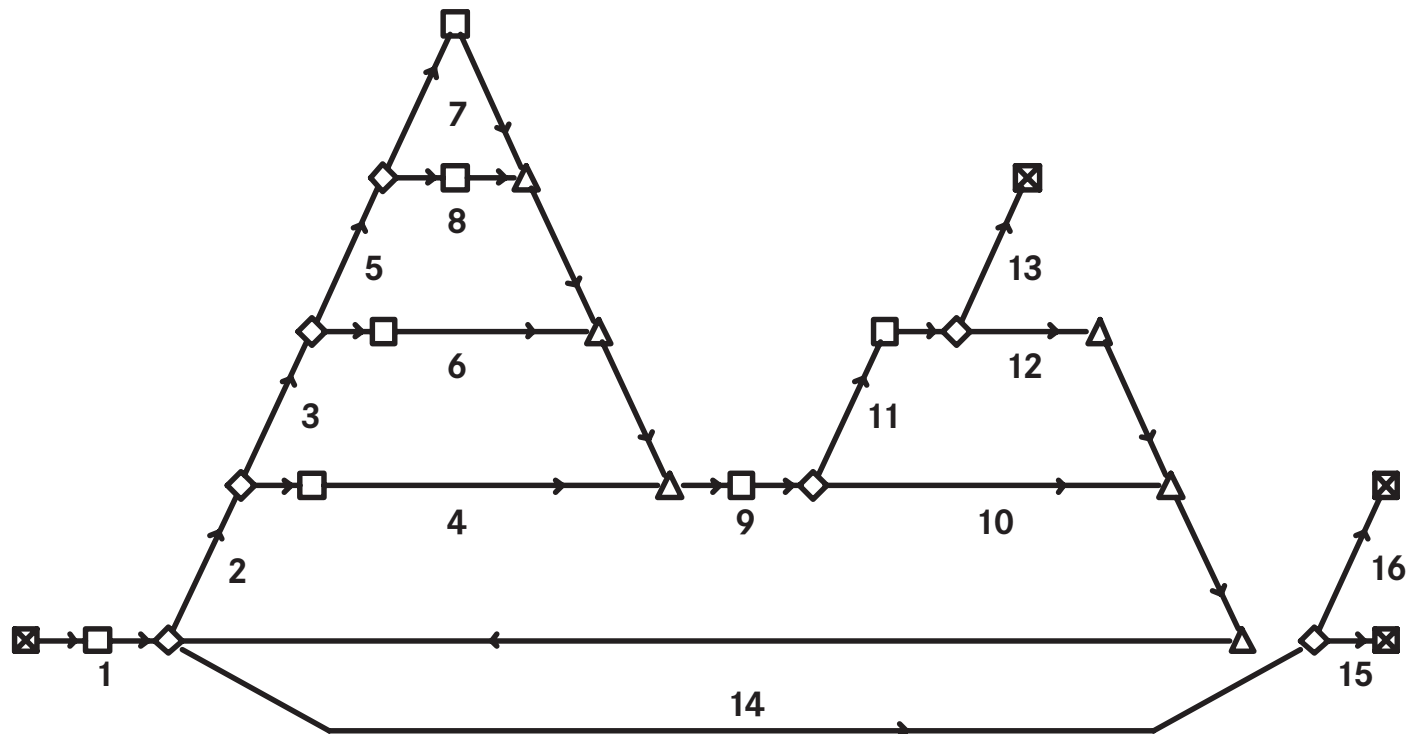


Testing Temporal Correctness Experiment

Sample Test Object: C-Function with

- 16 branches
- 37 statements
- 107 LOC
- 22 noncyclic paths
- cyclomatic number of 8

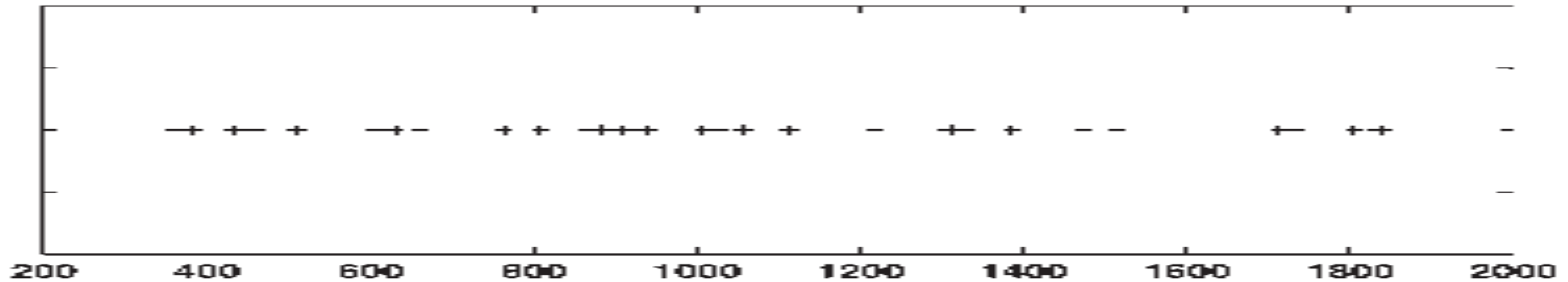
Control-Flow Graph



Testing Temporal Correctness Experiment

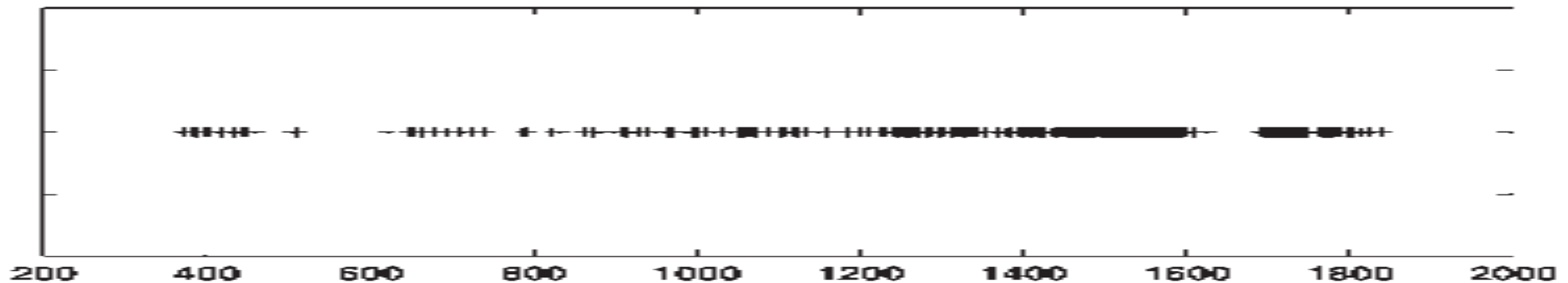
Systematic test

- ➔ 49 test cases
- ➔ 100 % branch coverage (C1)
- ➔ 43 different execution times from 5.27 to 26.27 μsec (359 to 1839 cycles)



Statistical test

- ➔ 4603 randomly generated sets of test data
- ➔ 94 % C1
- ➔ 298 different execution times from 5.27 to 26.27 μsec (359 to 1839 cycles)



Testing Temporal Behavior

- ➔ Very complex task which requires specialized procedures, methods, and tools

Objective of Testing: Finding Errors

- The temporal behavior of real-time systems is defective when input situations exist in such a manner that their computation violates the specified timing constraints.
- ➔ Find the input situations with the longest or shortest execution times to check whether they produce a temporal error.

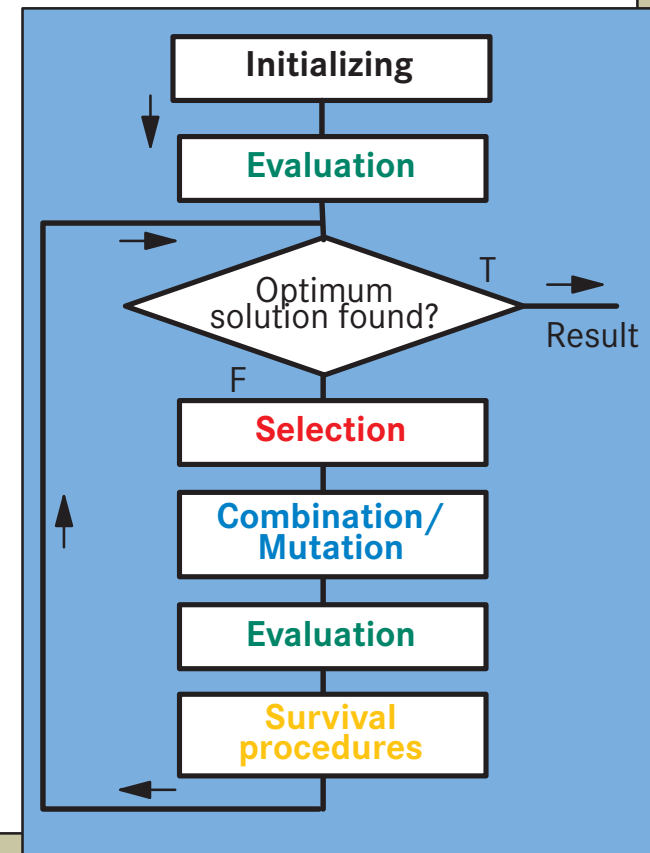
Our Approach

Use genetic optimization to determine the longest and shortest execution times automatically.



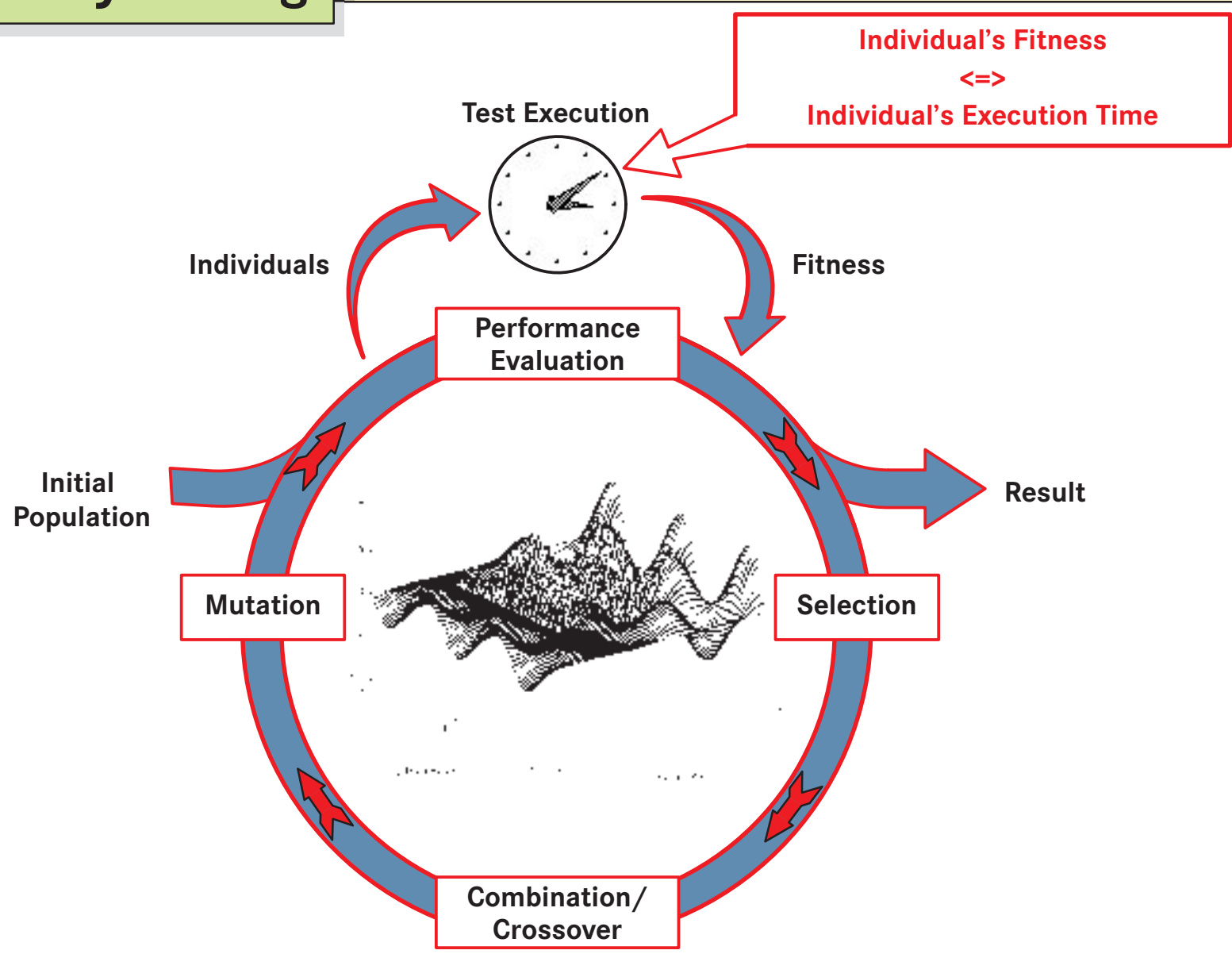
Genetic Optimization

- Iterative optimization procedure which imitates natural genetics and biological evolution
- For each iteration a new population of individuals (potential solutions of problems) is generated and evaluated.
- A new population is created from an existing one
by means of
 - **selection**
 - **recombination**
 - **performance evaluation &**
 - **survival**until
 - an optimum solution is found or
 - a stopping condition defined beforehand is reached.



Testing Temporal Correctness

Evolutionary Testing



Evolutionary Testing of Temporal Correctness

Searching Sample Test Object

the longest execution time



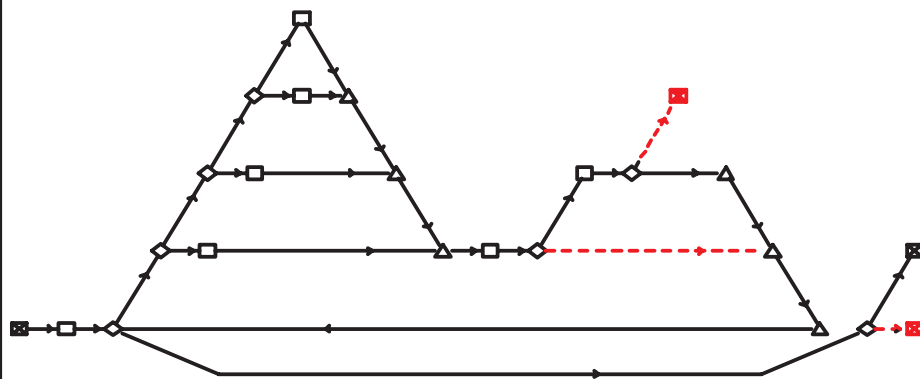
for



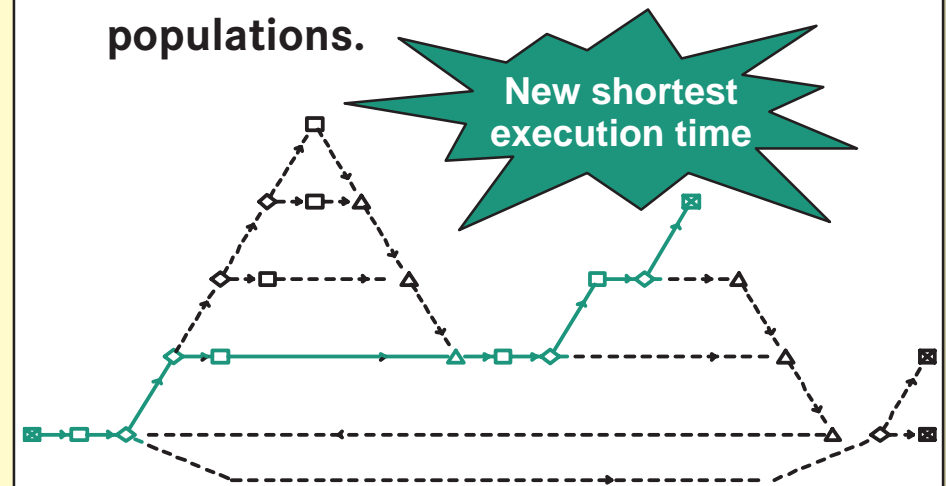
the shortest execution time

varying selection procedures, genetic operators, and survival procedures

- in all experiments the longest execution time so far (1839 cycles) was found very fast, always within 1 to 20 generations.



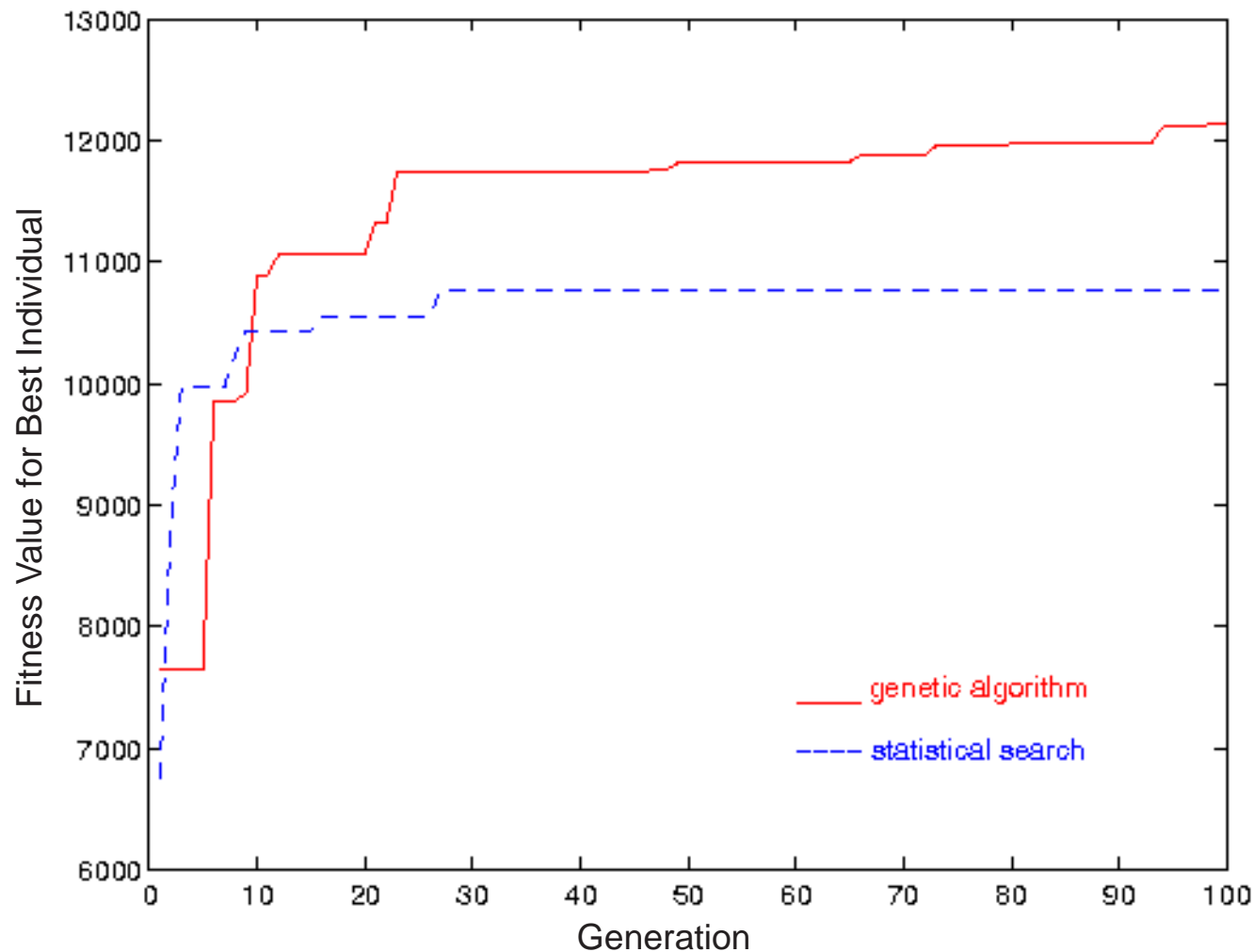
- in all experiments the shortest execution time so far (359 cycles) was found very fast.
- In several experiments a new shortest execution time of 355 cycles (5.07 μ sec) was found, e.g. by using sub-populations.



Evolutionary Testing of Temporal Correctness

Example: Automotive Electronics

- Comparison of **evolutionary testing** and **random testing** (in search of the longest execution time)

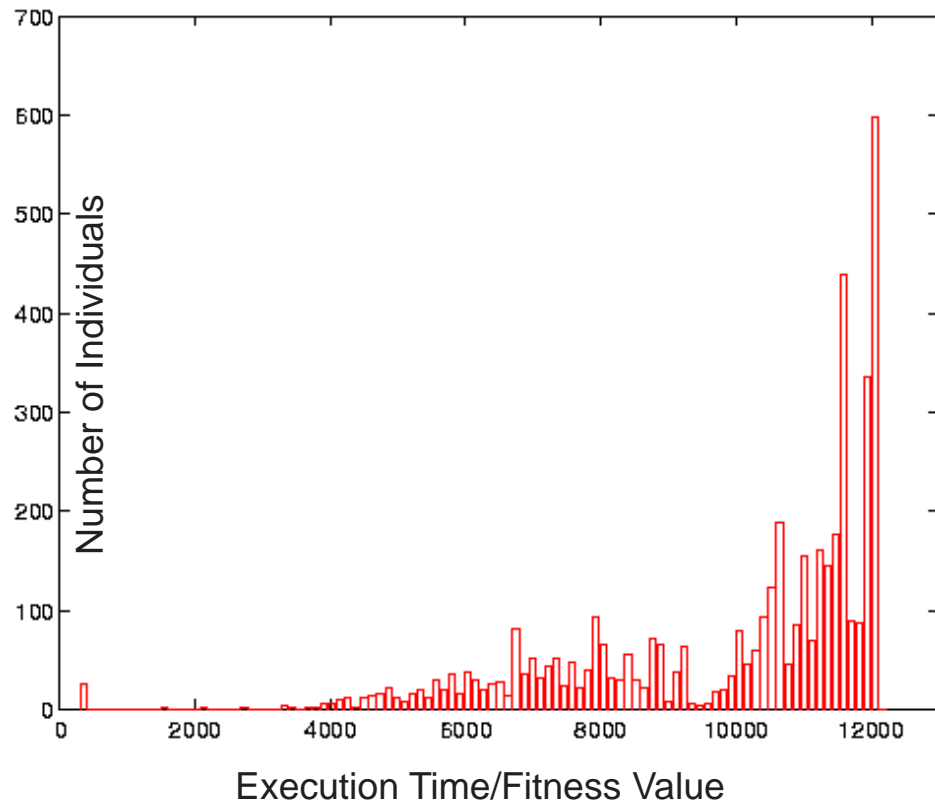


Evolutionary Testing of Temporal Correctness

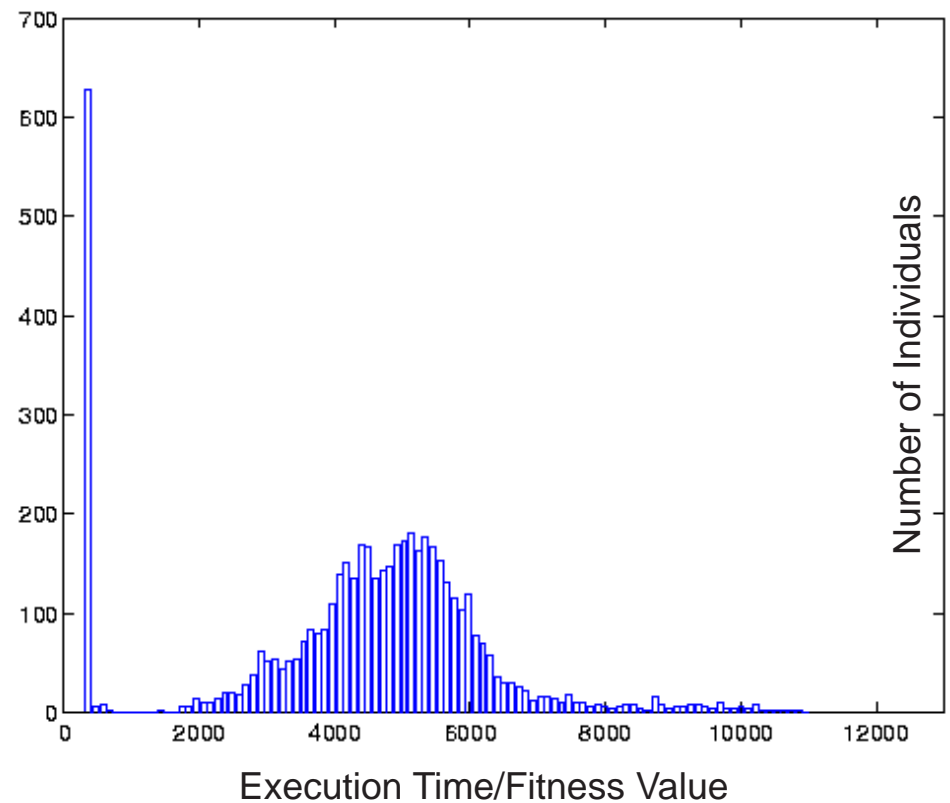
Example: Automotive Electronics

- Frequency of different fitness values over 100 generations for **evolutionary testing** and **random testing** (in search of the longest execution time)

Distribution of Fitness Values for Evolutionary Testing



Distribution of Fitness Values for Random Testing

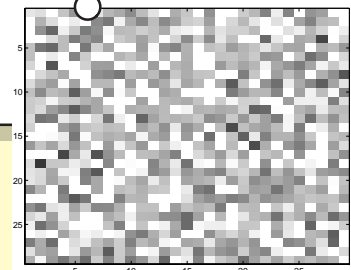
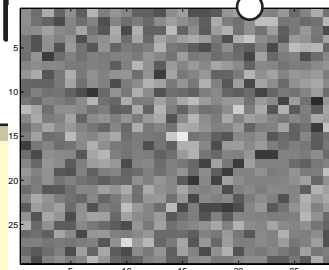


Evolutionary Testing of Temporal Correctness

Experiments

Applications	Run Times Random Testing			Run Times Evolutionary Testing		
	Test Runs	Shortest*	Longest*	Test Runs	Shortest*	Longest*
Computer Graphics (61 LOC)	9200	99	384	1200	99	392
Computer Graphics (107 LOC)	4600	359	1839	800	355	1839
Automotive Electronics (432 LOC)	2700	66	104	1350	45	104
Automotive Electronics (1511 LOC)	5000	366	10774	4850	366	12185
Railroad Control Technology (389 LOC)	10000	1050	11529	9500	399	14175
Defense Electronics (879 LOC)	56000	27258	110842	30000	26814	114393

* processor cycles



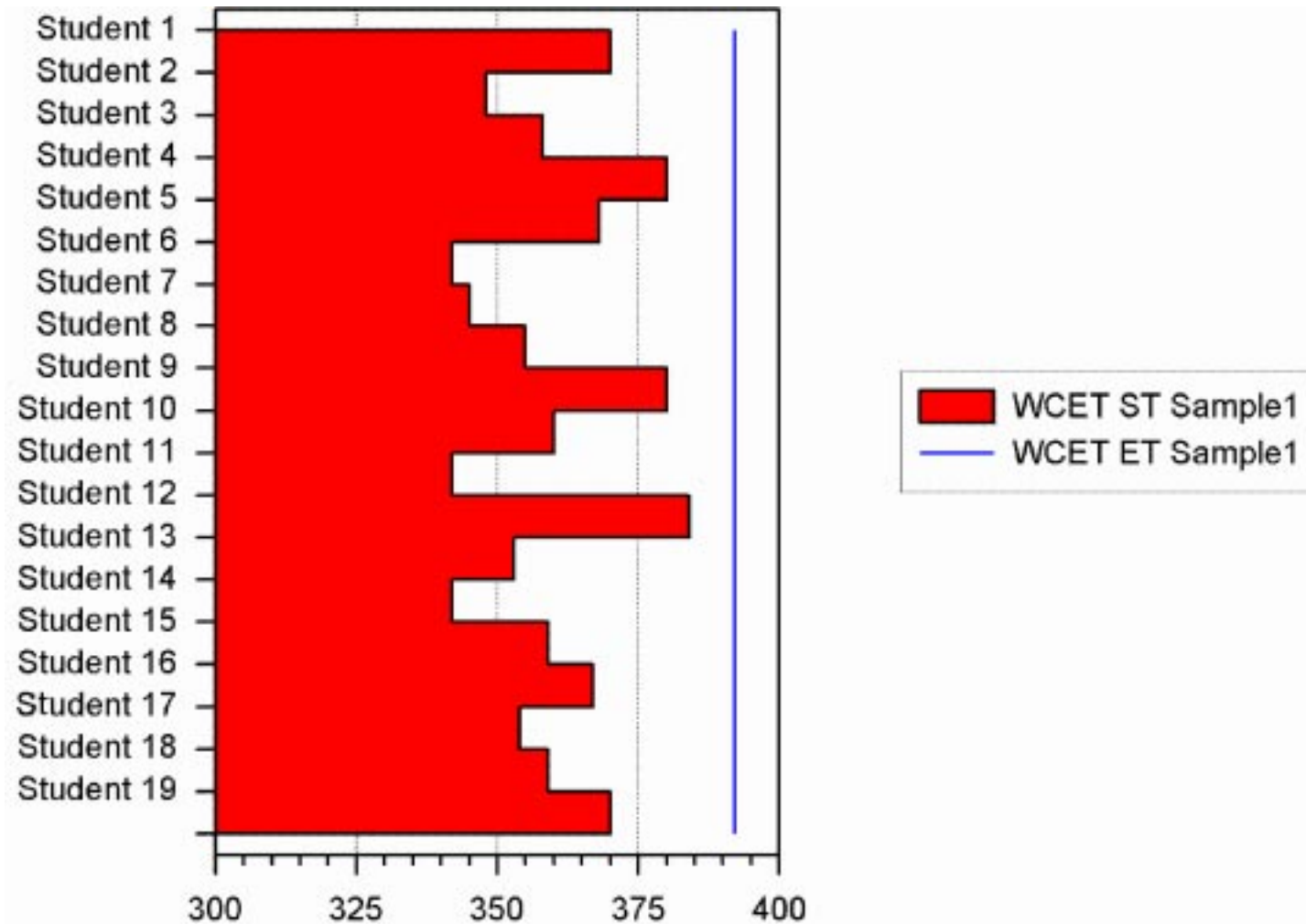
Comparison of Systematic and Evolutionary Testing: Experiment

- 19 students testing two different sample test objects from the field of computer graphics for functional as well as temporal correctness
- students were participating in a software testing course at the Technical University of Berlin and were trained in software testing methods
- the tests were marked and made up more than 20 % of the aggregate mark of each student
- the classification-tree method was used for the test case design
- test execution times were measured and compared to results achieved by evolutionary testing

Evolutionary Testing of Temporal Correctness

Comparison of Systematic and Evolutionary Testing

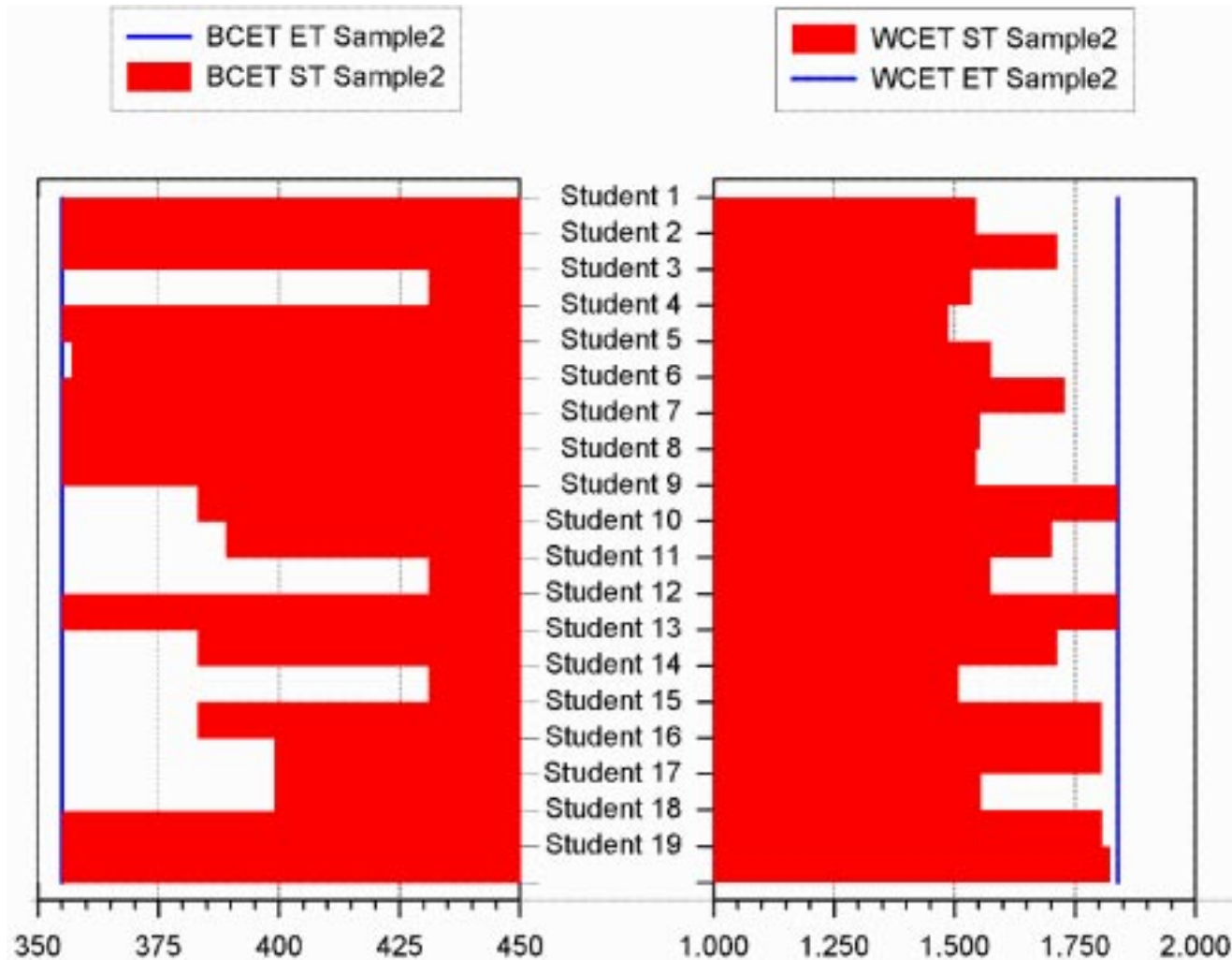
- Longest execution times found by **systematic testing** compared to evolutionary testing



Evolutionary Testing of Temporal Correctness

Comparison of Systematic and Evolutionary Testing

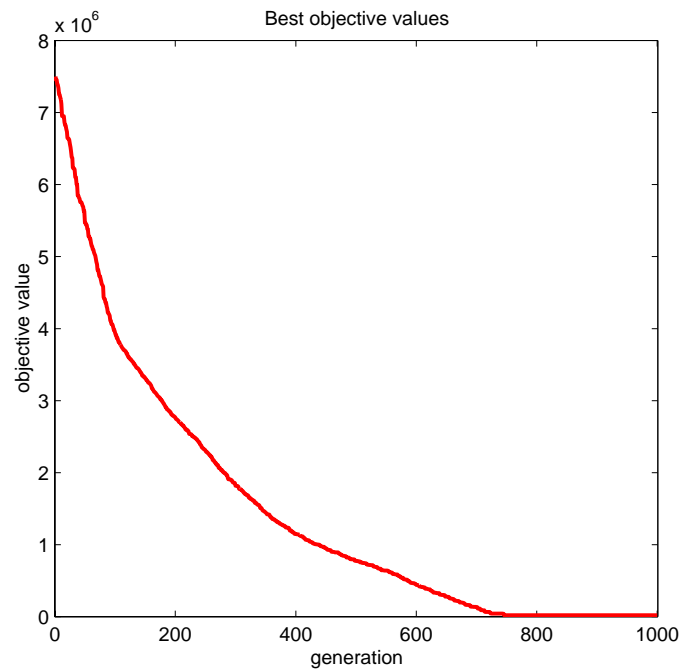
- Execution times found by **systematic testing** compared to evolutionary testing



Evolutionary Testing of Temporal Correctness

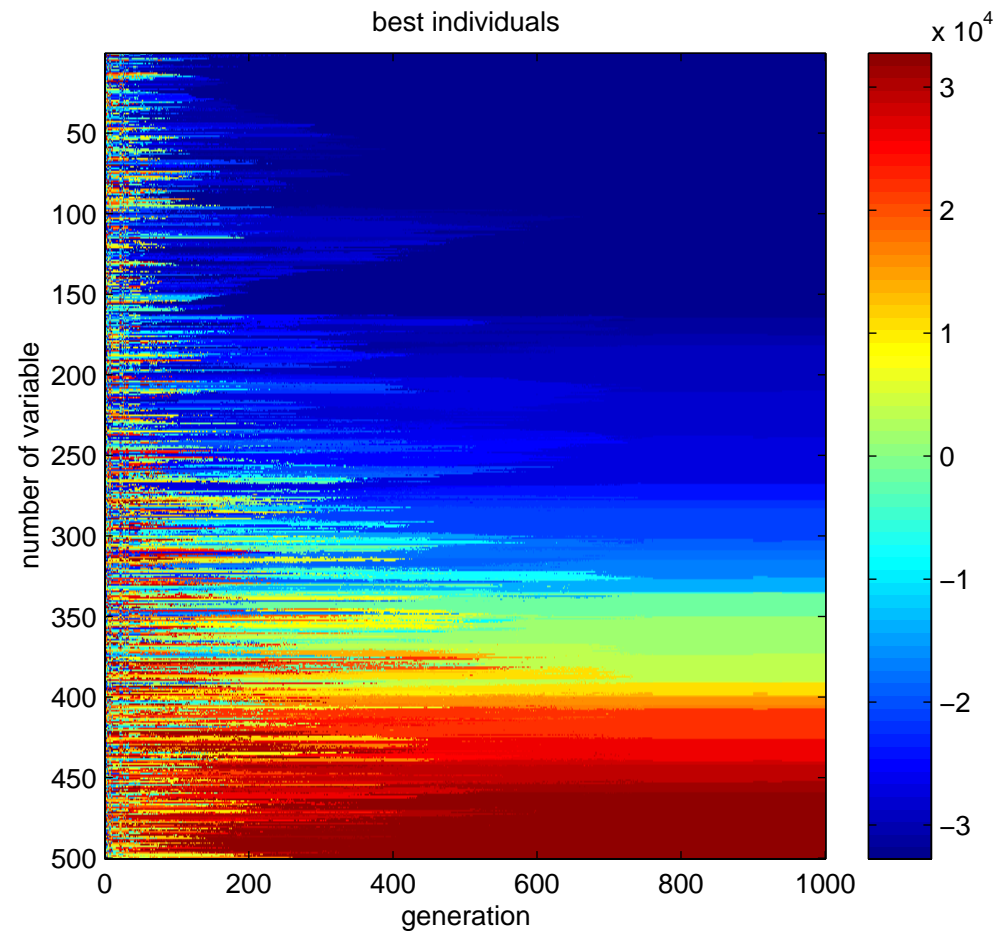
Bubblesort

Improvement of execution times over 1000 generations (searching for the shortest execution time)



➔ **Actual shortest execution time (20998 cycles) found after 746 generations!**

Visualisation of the test data with the shortest execution time in each generation



Evolutionary Testing of Temporal Correctness

Bubblesort

variable values for all
300 individuals after

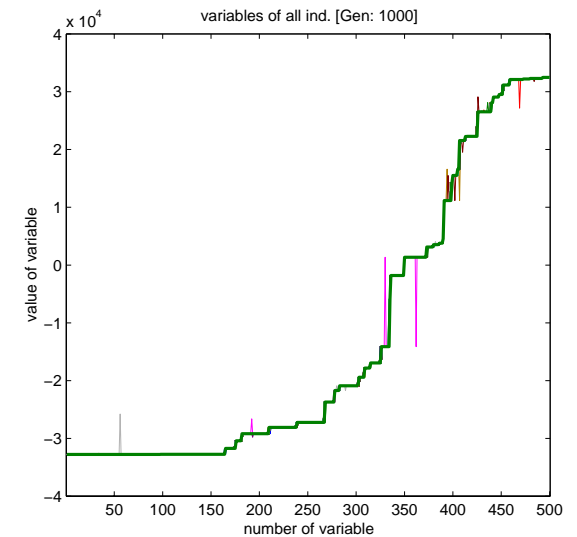
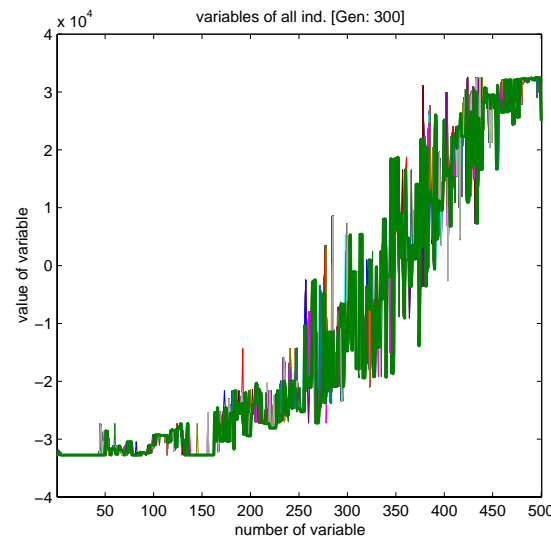
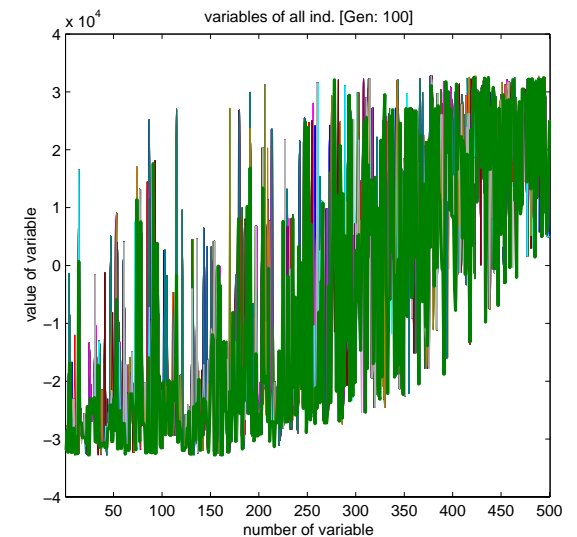
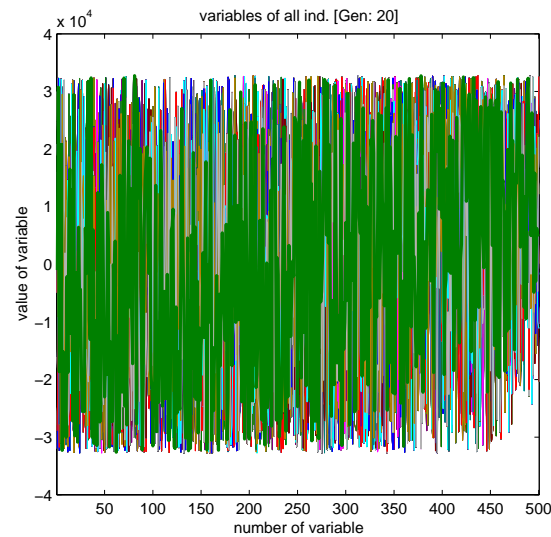
➔ 20

➔ 100

➔ 300, and

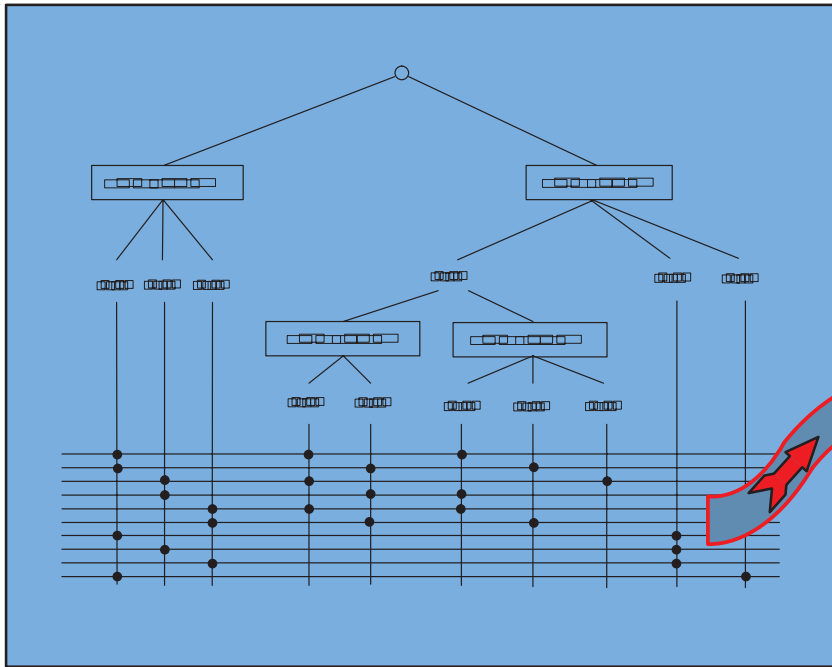
➔ 1000

generations
(when searching for the
shortest execution time)



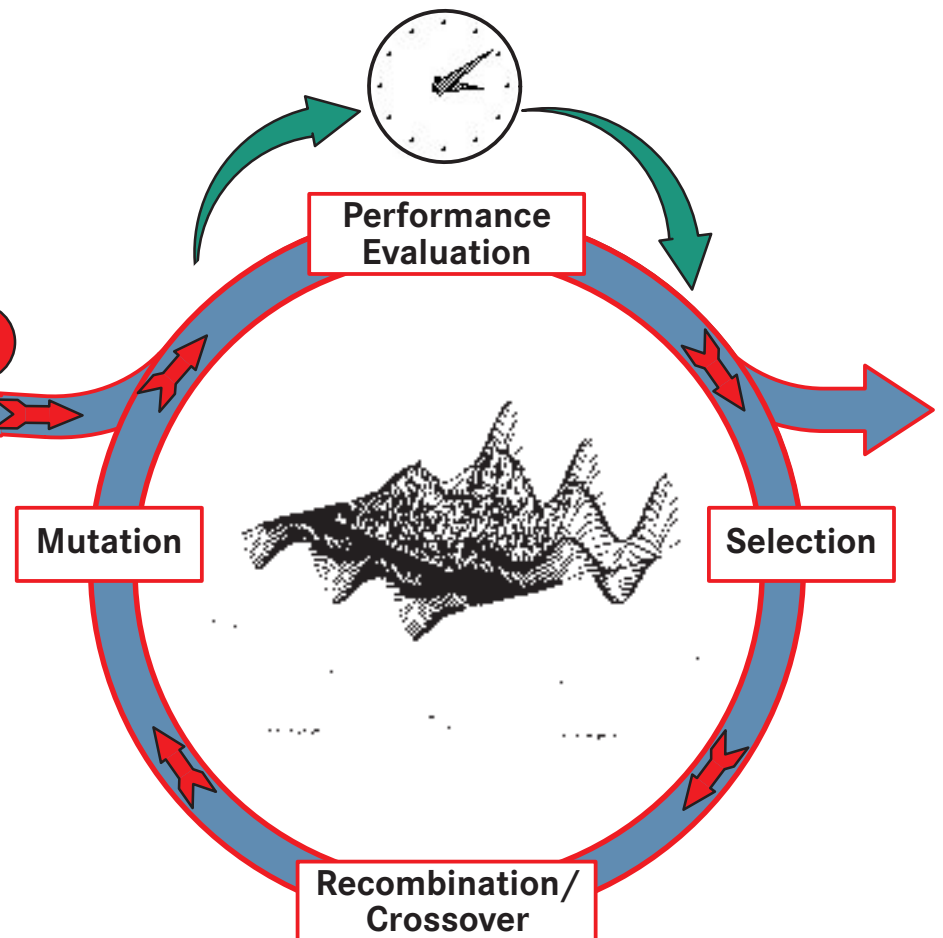
Test Strategy

- 1 Perform a systematic test to examine functional correctness.



- 2 Use systematically produced test data as initial population for the automatic search for extreme execution times.

- 3 Apply evolutionary testing for the examination of temporal correctness.



Evolutionary Testing of Temporal Correctness

Summary

- In practice testing is the most important analytical quality assurance method for real-time systems.
- Functional as well as temporal system behavior need to be examined:
 - ⊕ proven test methods suited for testing functional system behavior,
 - ⊖ no procedures specialized in testing temporal behavior.
- ➔ New approach for testing temporal behavior: **evolutionary testing**
 - First experiments for testing temporal behavior with genetic algorithms have been successfully completed.
 - In all experiments evolutionary testing achieved better results than random testing.
- Combination of systematic testing and evolutionary testing opens up further potential for improvement.

Evolutionary Testing of Temporal Correctness

Matthias Grochtmann, Joachim Wegener

Daimler-Benz AG
Research and Technology
Alt-Moabit 96 a
D-10559 Berlin
Germany

Tel.: +49 30 39982-{229, 232}, Fax: +49 30 39982-107

Matthias.Grochtmann@dbag.bln.daimlerbenz.com,

Joachim.Wegener@dbag.bln.daimlerbenz.com

Abstract

The development of real-time systems is an essential industrial activity. Dynamic testing is the most important analytical method to assure the quality of real-time systems. It is the only method that examines the run-time behavior, based on an execution in the application environment.

An investigation of existing software test methods shows that they mostly concentrate on testing for functional correctness. They are not specialized in the examination of temporal correctness that is essential to real-time systems. Therefore, existing test procedures must be supplemented by new methods that concentrate on determining whether the system violates its specified timing constraints. In general, a violation means that outputs are produced too early or their computation takes too long. The task of the tester is to find the inputs with the longest or shortest execution times to check whether they produce a temporal error. If the search for such inputs is interpreted as a problem of optimization, genetic algorithms can be used to find the inputs with the longest or shortest execution times automatically. The fitness function is the measured execution time. The use of genetic algorithms for testing is called evolutionary testing.

Experiments using evolutionary testing on a number of programs with up to 1511 LOC and 843 input parameters have successfully identified new longer and shorter execution times than those that had been found using random testing and systematic testing. Evolutionary testing therefore seems to be well-suited for checking the temporal correctness of real-time software. A combination of evolutionary testing with systematic testing offers further opportunities to improve the test quality and is suggested in this paper as an effective test strategy for real-time systems.

0 Introduction

Many industrial products use embedded computer systems. Usually, embedded computer systems have to fulfil real-time requirements, and correct system functionality depends on their logical correctness as well as on their temporal correctness.

In practice, dynamic testing is the most important analytical method for assuring the quality of embedded computer systems. Testing is aimed at finding errors in the systems and giving confidence in their correct behavior by executing the test object with selected inputs. Often more than 50 % of the overall development budget is spent on testing [Davis, 1979].

For testing real-time systems the examination of the functional system behavior alone is not sufficient. Additionally, the temporal behavior of the systems needs to be thoroughly examined. An investigation of existing

software test methods shows that they mostly concentrate on testing for functional correctness. They are not suited for an examination of temporal correctness which is also essential to real-time systems. This work tries to fill this gap by giving support to testing temporal behavior. It investigates the effectiveness of genetic algorithms to validate the temporal correctness of embedded systems by establishing the maximum and minimum execution times. Promising results have been achieved. However, the sole use of evolutionary testing is not sufficient for a thorough and comprehensive examination of real-time systems. A combination with existing test procedures is necessary to develop an effective test strategy for embedded systems. A combination of systematic testing and evolutionary testing is promising.

The first section contains an overview of the current state in the field of testing real-time systems. The second section gives a brief introduction to genetic algorithms and describes how they are applied to solve different testing problems. This is followed by a depiction of the way genetic algorithms are used for evolutionary testing of real-time systems' temporal behavior. Several experiments were performed comparing evolutionary testing with random testing as well as systematic testing. Their results will be described in detail. Section 4 discusses the combination of systematic testing and evolutionary testing and derives from it a test strategy for real-time systems. After some concluding remarks the paper closes with a short outlook on current and future work.

1 Testing Real-Time Systems

Testing is one of the most complex and time-consuming activities within the development of real-time systems [Heath, 1991]. It typically consumes 50 % of the overall development effort and budget since embedded systems are much more difficult to test than conventional software systems. The examination of additional requirements like timeliness, simultaneity, and predictability make the test costly. In addition, testing is complicated by technical characteristics like the development in host-target environments, the strong connection with the system environment, the frequent use of parallelism, distribution and fault-tolerance mechanisms as well as the utilization of simulators.

Nevertheless, systematic testing is an inevitable part of the verification and validation process for software-based systems. Testing is the only method that allows a thorough examination of the test object's run-time behavior in the actual application environment. Dynamic aspects like the duration of computations, the memory actually needed during program execution, or the synchronization of parallel processes are especially important for the correct functioning of real-time systems.

Real-time systems must be tested for compliance with their functional specification and their timing constraints. An investigation of existing software test methods shows that a number of proven functional and structural test methods is available for examining logical correctness, e.g. the classification-tree method [Grochtmann and Grimm, 1993]. When using structure-oriented test methods the tester must take into account that an instrumentation of the test object causes probe effects, i.e. deviations from the real system behavior are possible. There are no specialized procedures for testing temporal behavior. This is why in practice testers often go back to conventional test procedures. They try to compensate for the existing methodological shortcomings by using more or less heuristic tests in worst-case scenarios. However, even a small experiment makes clear that testing temporal behavior is a very complex task which requires special methods and tools.

1.1 Experiment

For the experiment, a simple computer graphics function written in C was thoroughly tested. It contained a total of 37 statements and had 16 program branches; its control flow graph is shown in Figure 1. A systematic test with the classification-tree method led to 49 test cases which covered all the branches and resulted in 43 different execution times. The timings varied between 359 processor cycles (equivalent to 5.27 μ s) and 1839 processor cycles (equivalent to 26.27 μ s).

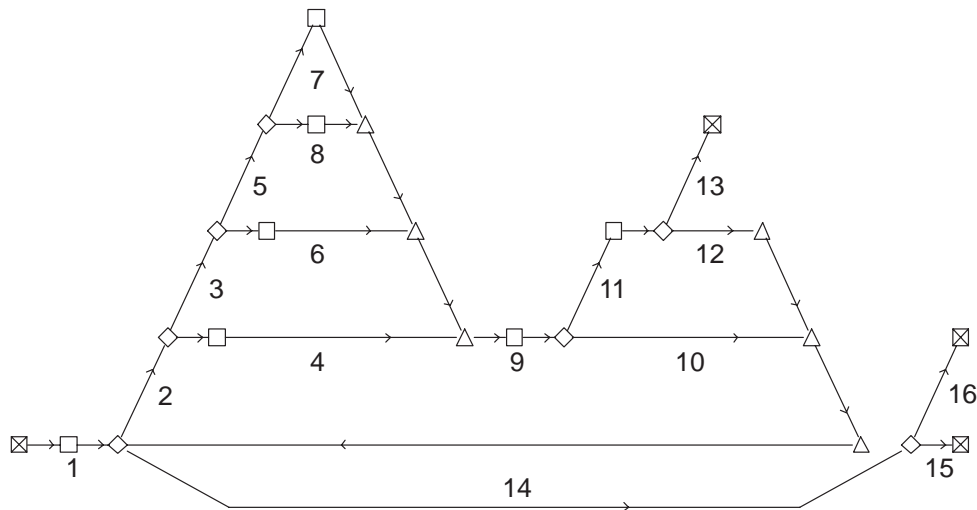


Figure 1: Control Graph

This was followed by applying a total of 4603 randomly generated tests, which resulted in 298 different execution times even though full branch coverage was not achieved; one branch remained untested. The timings varied again between 359 cycles and 1839 cycles.

These initial results demonstrate the importance of investigating a specialized approach to testing temporal correctness: already very small systems show a wide range of different execution times. Only a fraction of the possible execution times is covered by the systematic test. Random testing, however, does not detect certain value combinations that might be significant for the temporal behavior. Consequently, for testing temporal behavior the application of a new approach was examined, namely genetic algorithms.

2 Genetic Algorithms

Genetic algorithms are a well-established method of optimization in many areas. An overview of different successful applications is, for example, provided by Davis [1996]. Genetic algorithms represent a class of adaptive search techniques and procedures based on the processes of natural genetics and Darwin's theory of evolution. Genetic algorithms model natural processes, such as selection, reproduction, mutation, migration, locality, and neighborhood [Pohlheim, 1996]. The fundamental concept of genetic algorithms is to evolve successive generations of increasingly better combinations of those parameters which significantly effect the overall performance of a design. The genetic algorithm achieves the optimum solution by the random exchange of information between increasingly fit samples (combination/crossover) and the introduction of a probability of independent random change (mutation). Traditionally, parameters involved in genetic optimization have been represented as strings of binary bits where crossover is achieved by choosing a point along two bit strings at random and swapping the tails, and mutation by picking a bit at random and flipping its value. The adaptation of the genetic algorithm is achieved by the selection and survival procedures since these are based on fitness. The fitness-value is a numerical value that expresses the performance of an individual with regard to the current optimum so that different designs may be compared. The notion of fitness is fundamental to the application of genetic algorithms; the degree of success in using them may depend critically on the definition of a fitness that changes neither too rapidly nor too slowly with the design parameters.

Figure 2 gives an overview of a typical procedure for genetic algorithms. A population of guesses to the solution of a problem is initialized, usually at random. Each individual in the population is evaluated by calculating its fitness. This will result in a spread of solutions ranging in fitness from very poor to good – the chances of hitting on the optimum solution initially are, of course, infinitesimally small for most problems. The remainder of the algorithm is iterated until the optimum is achieved. Pairs of individuals are selected from the popula-

tion using a pre-defined strategy, and are combined in some way to produce a new guess in an analogous way to biological reproduction. Combination algorithms are many and varied. Additionally, mutation is applied. The new individuals are evaluated for their fitness, and survivors into the next generation are chosen from the parents and offspring, often according to fitness though it is important to maintain a diversity in the population to prevent premature convergence to a sub-optimal solution.

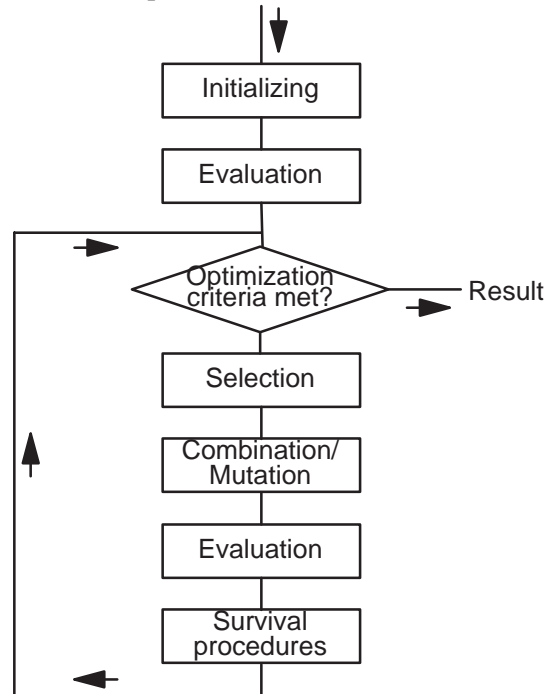


Figure 2: Block Diagram of Genetic Algorithms

2.1 Genetic Algorithms Applied to Testing

Genetic algorithms have been applied successfully to various testing problems. Several papers deal with structural testing; others concentrate on test case generation based on formal specifications, the testing of APIs, and testing for robustness.

Jones et al. [1996] have used genetic algorithms to generate test data automatically to execute every branch in a variety of programs written in Ada83. In most cases full branch coverage was obtained. The branch predicate formed the basis of the fitness function so that boundary value data were generated.

Roper [1996] has obtained encouraging results by applying genetic algorithms to achieve branch coverage for programs written in C or C++. The test object is instrumented with probes to provide feedback on the coverage achieved. For each individual, the program path executed determines its level of fitness.

Xanthakis et al. [1992] describe a method in which a constraint propagation graph is formed; the nodes of the graph may represent either predicates or variables connected by elementary path functions. The fitness function depends on the predicates which are satisfied by modifying the adjacent variable nodes by a small amount. This work was continued by Watkins [1995] who used a fitness function based on the reciprocal of the number of times a path was exercised.

Jones et al. [1995] have derived test sets from Z specifications by a method that used a variety of algorithms, including genetic algorithms and simulated annealing. A language was developed to enter the Z schemas into the machine and a series of test cases was formed for both valid and invalid inputs.

Boden and Martino [1996] have developed a testing facility in which genetic algorithms are used to generate API tests. The fitness function is a weighted sum of various factors of a test response, e.g. depending on the generation of exceptions, well-defined errors or return codes by the API. Furthermore, sequences of API calls are determined as useful, based on expected or recommended usages for the API.

Schultz et al. [1993] have achieved promising results using genetic algorithms for testing the robustness of autonomous vehicle controllers. The aim of the test was to find test scenarios in which minimal fault activity causes a mission failure or vehicle loss and in which maximal fault activity still permits a high degree of mission success. These scenarios provided some insight into parts of the controller and allowed the designer to improve the controller's robustness. The fitness function was based on the current fault activity and the quality of mission fulfilment.

In this work, we investigate the feasibility of genetic algorithms for testing the temporal correctness of real-time systems.

3 Evolutionary Testing of Temporal System Behavior

The major objective of testing is to find errors. Real-time systems are tested for logical correctness by standard testing techniques such as the classification-tree method. A common definition of a real-time system is that it must deliver the result within a specified time interval and this adds an extra dimension to the validation of such systems, namely that their temporal correctness must be checked.

The temporal behavior of real-time systems is defective when input situations exist in such a manner that their computation violates the specified timing constraints. In general, this means that outputs are produced too early or their computation takes too long. The task of the tester therefore is to find the input situations with the shortest or longest execution times to check whether they produce a temporal error. This search for the shortest and longest execution times can be regarded as an optimization problem to which genetic algorithms seem an appropriate solution. The use of genetic algorithms for testing is called evolutionary testing [Wegener et al., 1997a; Wegener and Grochtman, 1998].

Genetic algorithms enable a totally automated search for the longest and shortest execution times. They are particularly suited to problems involving large numbers of variables and complex input domains. Even for non-linear and poorly understood search spaces genetic algorithms have been used successfully. Since genetic algorithms search from a population of points rather than from a single point, the probability of getting stuck at local optima is significantly reduced compared with more traditional optimization techniques, like hill climbing. The use of mutation and subpopulations can further reduce the chance of getting stuck. When genetic algorithms are used to solve optimization problems, good results are obtained surprisingly quickly [Sthamer, 1996].

Figure 3 illustrates the use of evolutionary testing for determining the shortest and longest execution times. The initial population is generated at random. Each individual of the population represents a test datum with which the test object is executed. For every test datum the execution time is measured. The execution time determines the fitness of the respective individual or test datum. If one searches for the longest execution time, test data with long execution times obtain high fitness values. If one searches for the shortest execution times, individuals with short execution times obtain high fitness values. Afterwards, members of the population are selected with regard to their fitnesses and subjected to combination and mutation to generate a new population. First, it is checked whether the generated test data are in the input domain of the test object. Then the individuals of the new generation are also evaluated and united with the previous generation to form a new population according to the survival procedures laid down. Afterwards, this process repeats itself, starting with selection, until a given stopping condition is reached or a temporal error is detected. Thus an execution time is found which is outside the specified timing constraints. If all the times found meet the timing constraints specified for the system under test, confidence in the temporal correctness of the system is substantiated.

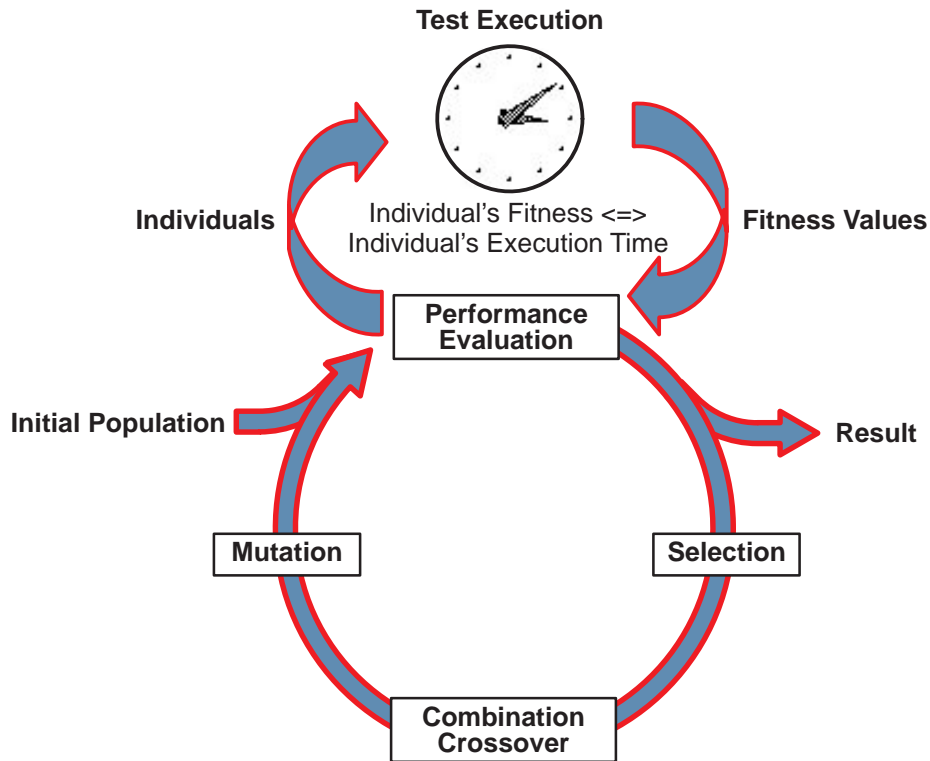


Figure 3: Evolutionary Testing of Temporal System Behavior

3.1 Experiments

We used evolutionary testing in several experiments to determine the shortest and longest execution times of different systems. Of course the results are dependent on the hardware/software platform and generally are not directly transferable from one to another since the processor speed and the compiler used directly affect the temporal behavior. All the experiments to be described were carried out on a SPARCstation 10 running under Solaris 2.5. The duration of executions was measured in processor cycles to rule out overheads by the operating system such as service interrupts. Thus the execution times reported were the same for repeated runs with identical parameters.

The fitness was set equal to the execution time when searching for the worst-case execution time and was set equal to the negative value of the execution time when searching for the best-case execution time. For each experiment evolutionary testing was applied twice, first, to find the longest execution time, and then the shortest. The library of genetic algorithms we used was a Matlab-based toolbox developed at our Daimler-Benz Laboratories by Hartmut Pohlheim.

3.1.1 Comparison of Random Testing and Evolutionary Testing

For the experiments six test objects from real-world applications were tested by means of evolutionary testing. The results of the experiments are summarized and compared to random testing in table 1.

The first computer graphics example (Computer Graphics I) shows that even when the number of test runs for random testing was multiplied, compared with the number of test runs for evolutionary testing, the longest execution time determined by evolutionary testing could not be found by random testing. When evolutionary testing was applied to testing the simple C function already mentioned in section 1.1 (Computer Graphics II), the longest execution time of 1839 cycles was found in less than 20 generations and a new shortest time of 355

cycles (5.07 μ s) was discovered in a smaller number of tests than executed for random testing. For the first example from the field of automotive electronics shorter execution times were found by evolutionary testing than by random testing.

Applications	Random Testing			Evolutionary Testing		
	No. of Test Runs	Shortest Exec. Time	Longest Exec. Time	No. of Test Runs	Shortest Exec. Time	Longest Exec. Time
Computer Graphics I (61 LOC)	9200	99	384	1200	99	392
Computer Graphics II (107 LOC)	4600	359	1839	800	355	1839
Auto Electronics I (432 LOC)	2700	66	104	1350	45	104
Auto Electronics II (1511 LOC)	5000	366	10774	4850	366	12185
Railroad Technology (389 LOC)	10000	1050	11529	9500	399	14175
Defense Electronics (879 LOC)	56000	27258	110842	30000	27154	114160

Table 1: Shortest and Longest Execution Times in Processor Cycles Measured for a Variety of Programs by Random Testing and Evolutionary Testing; the Overall Optimum is Shown in Bold

The second automotive electronics system implements the entire functionality of an airbag controller and therefore is safety critical. It contains 1511 LOC. For this example about 80 parameters were varied by evolutionary testing, among other things the interval between the occurrence of different events. There is no means of deciding when an optimum has been found and the genetic algorithms were allowed to continue for 100 generations before they were stopped.

When searching for the longest execution time of the airbag controller software, a maximum of 12185 processor cycles was found in generation number 97. In the same way, the random test was terminated after 5000 tests, and at this time had found a maximum of 10774 cycles. Evolutionary testing had found an execution time 13% longer than was found for random testing. Figure 4 shows that random testing reaches its maximum execution time after only about 1500 test runs, whereas for evolutionary testing a continuous improvement up to the 100th generation can be observed. The curve trace suggests that the genetic algorithms would find even longer execution times if the number of generations were increased. After only eleven generations – that corresponds to 550 test runs – the execution times found by evolutionary testing are above those of random testing.

When searching for the shortest execution time, both evolutionary testing and random testing found the same time of 366 cycles. The genetic algorithms discovered this in the first generation at which point it is still effectively a random search since the recombinations and mutations could have had no effect. This path was clearly one whose tests occupied a large input subdomain with a high probability of being found at random.

This is also emphasized by Figure 5 which indicates the frequency with which different execution times occurred during the search for the longest execution time. For random testing far more than 10 % of all test runs goes to particularly short run times. For the other execution times almost a Gaussian curve results from random testing. During evolutionary testing, however, not even 1 % goes to the area of particularly short run times. Furthermore, a clearly increasing number can be seen for the long execution times. The genetic algorithms obviously succeed in avoiding the generation of test data with short run times and in concentrating on test data with long execution times.

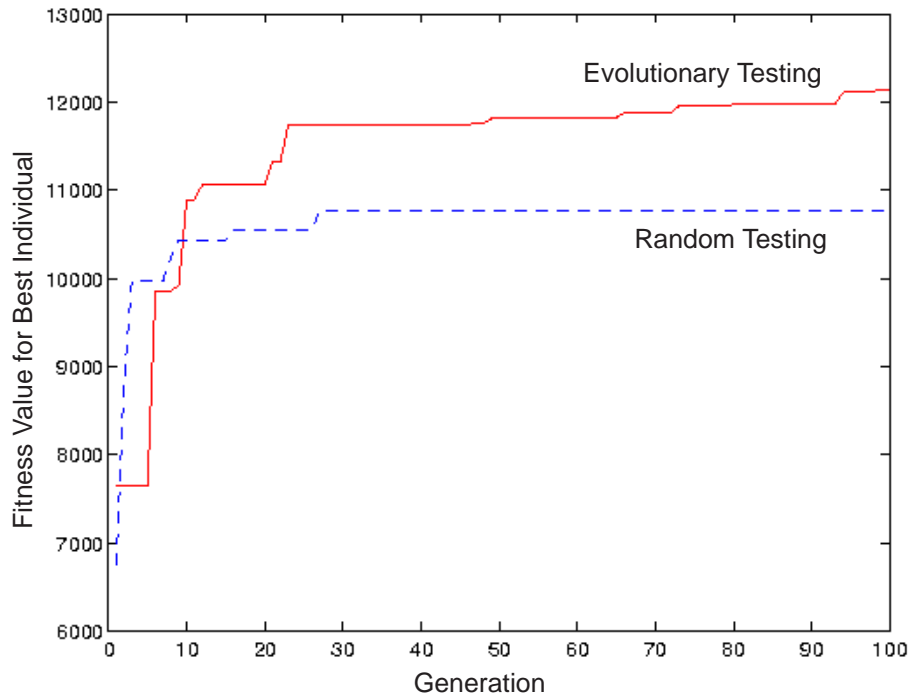


Figure 4: Comparison of Evolutionary Testing and Random Testing Searching for the Longest Execution Time for the Airbag Controller

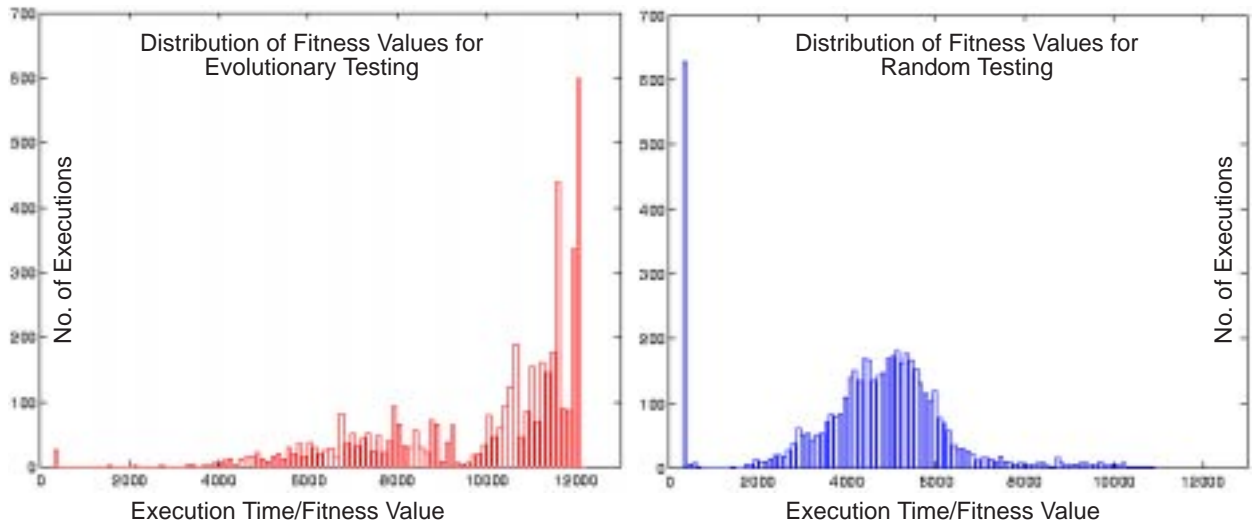


Figure 5: Distribution of Execution Times for Evolutionary Testing and Random Testing when Searching for the Longest Execution Time for the Airbag Controller

The railroad control and instrumentation technology example is also safety critical. The population size in this experiment was increased from 50 to 100 because of the complexity of the test object. The shortest execution time found by evolutionary testing is more than 60 % below the one detected by random testing. 14175 processor cycles were determined as the longest execution time by the genetic algorithms. This is 23 % above the maximum execution time of 11529 cycles that was found by random testing. Figure 6 shows the comparison of random testing and evolutionary testing for the search of the longest execution time. It becomes clear after the

fourth generation that evolutionary testing is superior to random testing. Random testing stagnates after 3500 test runs; the generation of 6500 other test data sets does not produce any longer execution times. Evolutionary testing, however, manages once again to improve the execution times continuously. In generation number 70 even a significant leap of more than 1500 cycles can be noted.

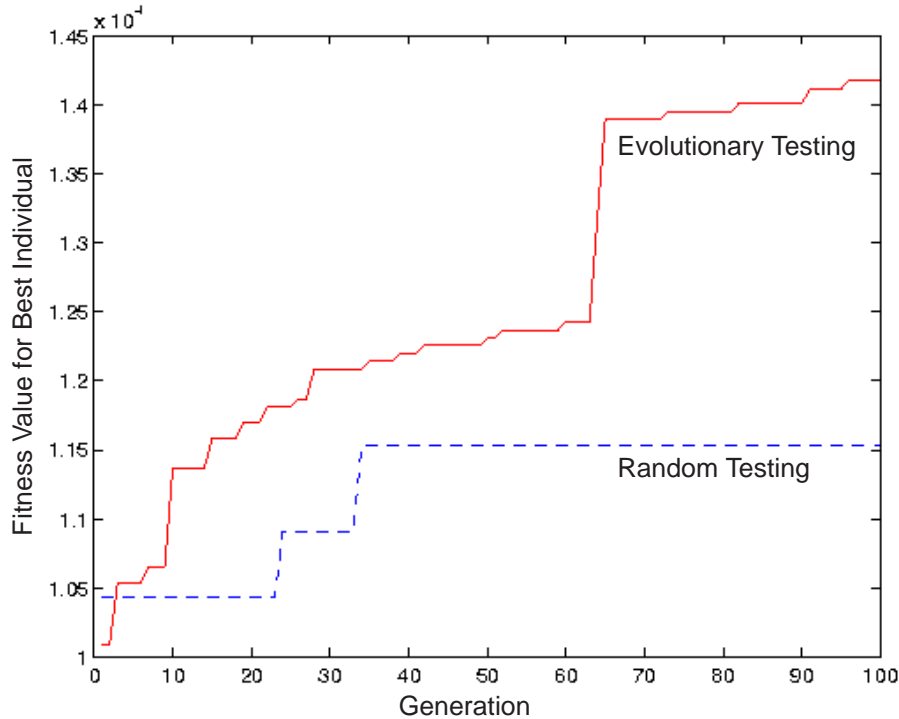


Figure 6: Comparison of Evolutionary Testing and Random Testing Searching for the Longest Execution Time for the Railroad System

The Defense Electronics program with 879 LOC has 843 integer input parameters. The first two input parameters represent the position of a pixel in a window and lie within the range 1..1200 and 1..287 respectively. The remaining 841 parameters define an array of 29 by 29 pixels representing a graphical input located around the specified position; each integer describes the pixel color and lies in the range 0..4095. Genetic algorithms were used in this example to generate pictures surrounding a given position. The longest execution time increased steadily with each new generation and asymptoted towards the current maximum of 114160 cycles when the run was terminated after 300 generations. The population size in this experiment was also set to 100 because of the large range of the variables and the large number of input parameters. The fastest execution time was found to be 27154 after 100 generations. Evolutionary testing found more extreme values for the longest and shortest execution times than those found by random testing that were 110842 and 27258 respectively. Where genetic algorithms allowed to search further a longest execution time of 114393 processor cycles was found in generation 657 and a shortest execution time of 26814 was detected in generation 372.

3.1.2 Comparison of Random Testing and Systematic Testing

Furthermore, experiments were performed to compare evolutionary testing with systematic testing. The results of these experiments are ambiguous: On the one hand, an experiment comparing evolutionary testing with the results of systematic tests carried out by 19 different testers on the two test objects from the field of computer graphics showed evolutionary testing to always be superior or equal to systematic testing. On the other hand, however, in an experiment testing a standard sorting routine evolutionary testing quickly approached the shortest execution time searched for but found the exact time only after several hundred more

generations. In this case, systematic testing easily found the input with the shortest execution time possible: the already sorted list.

During the first experiment which involved 19 testers, both applications from the field of computer graphics were tested for their functional as well as their temporal correctness. The testers were 19 students participating in a software testing course at the Technical University of Berlin. They were trained in software testing methods and tools. Test cases were designed by means of the classification-tree method. The tests were marked and made up more than 20 % of the aggregate mark of each student.

In addition to the test cases, the students determined test data with which the test was executed. The execution times were measured and compared to the results obtained by evolutionary testing. For the first example, all students had found the shortest execution time. However, none of the students found the longest execution time (392 processor cycles). The determined execution times varied between 342 cycles and 384 cycles (Figure 7). For the second example, 9 students ascertained the shortest execution time (355 cycles), but only two students found the longest execution time (1839 cycles). Only one of 19 students found both the shortest and the longest execution time for the second computer graphics example (Figure 8). The total number of test cases amounts to 639 for this example. The number of testcases varied from 11 to 58 for the students.

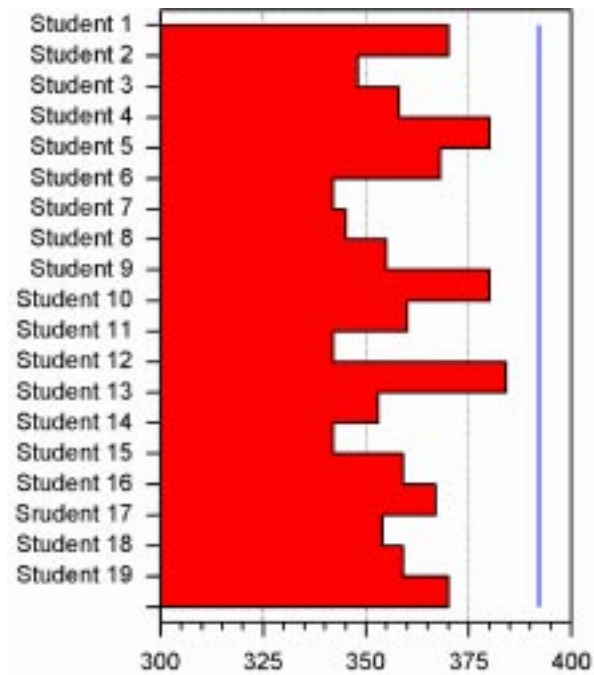


Figure 7: Execution Times for Computer Graphics I

While evolutionary testing was superior to systematic testing in the described experiments, the testing of a bubblesort algorithm showed advantages of the systematic test. The task of the test object was the sorting of a list of 500 elements. In this example the evolutionary test only reached the shortest execution time after ca. 750 generations. In this case, this equals 225000 executions of the bubblesort. Systematic testing easily found the input with the shortest execution time possible: the already sorted list. The longest execution time results from the list sorted in reverse order.

3.2 Discussion

In all our experiments evolutionary testing obtained better results than random testing, regardless of whether the shortest or the longest execution times were searched for. The disadvantage of a random method which is

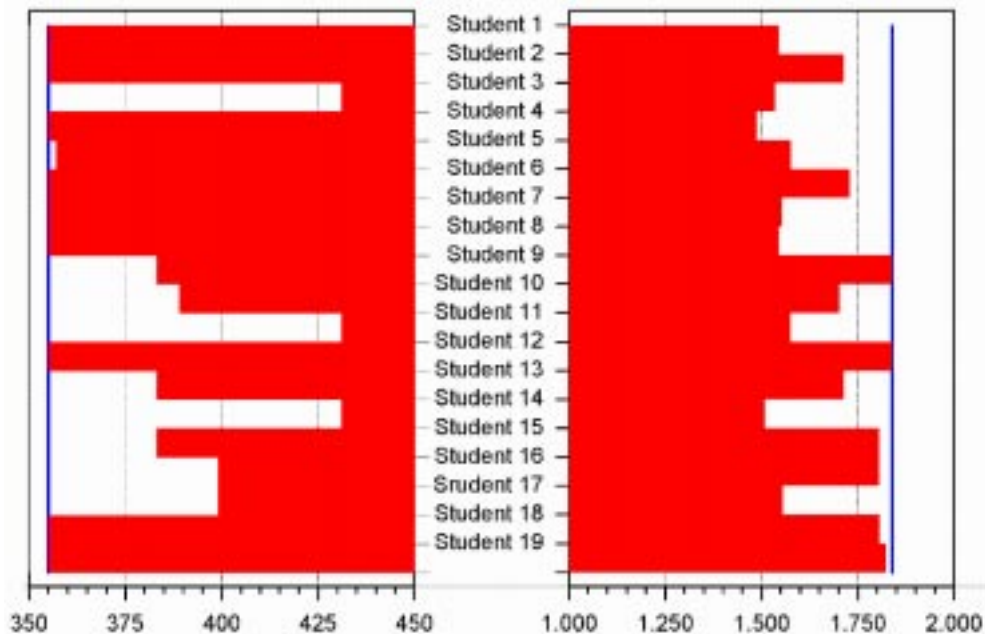


Figure 8: Execution Times for Computer Graphics II

that no step builds upon another is avoided by using genetic algorithms. Genetic algorithms take advantage of the old knowledge held in a parent population to generate new guesses with improved performance. Their iterations are based on the experience which has been gained from previous trials.

For both computer graphics examples systematic tests with the classification-tree method were also performed. The longest execution time of 392 processor cycles found by the genetic algorithms for the first example was detected by none of the systematic tests. Only 384 cycles as maximum and 99 cycles as minimum run time were found by the systematic tests. For the second example systematic testing and evolutionary testing detected the same run times, namely 355 and 1839 processor cycles. In some simpler experiments systematic testing turned out to be superior to evolutionary testing. For a sorting program, for example, the sorted list caused the shortest run time. Evolutionary testing needed more than 200000 executions of the test object to determine the best-case execution time.

Since genetic algorithms try to achieve the optimum solution by the random exchange of information between increasingly fit samples (combination) and the introduction of independent random change (mutation), they share a problem with random and statistical testing: it is not predictable if and when certain input situations will be found, which might be especially important for the run-time behavior of the system under test. Therefore, we cannot prove that the timings found are the longest and shortest possible values. On the other hand, existing approaches to systematic testing are not sufficient to examine the temporal behavior of systems thoroughly. Consequently, an effective test strategy for real-time systems should contain systematic testing as well as evolutionary testing.

4 Test Strategy

As a strategy for testing real-time systems we recommend the combination of systematic testing with evolutionary testing. By means of the systematic test errors in the logical program behavior of the test object shall be detected. Furthermore, special value combinations shall be defined which are relevant to the testing of temporal behavior but which might be difficult to find with the help of genetic algorithms. On the basis of the systematic test genetic algorithms are then used to detect input constellations with particularly long and short run times which the tester did not find by means of the systematic test.

The classification-tree method should be applied for the systematic test because it is a functional test method which has already proved very worthwhile in practice (cf. [Grochtmann and Wegener, 1995]). The function-oriented test is indispensable to the thorough examination of systems for only by means of test cases derived from the system specification can it be found out if specified requirements or functions were omitted (e.g. simply forgotten) during the software development process [Grimm, 1996].

The test strategy comprises two steps. At first, the tester uses the classification-tree method for the systematic design of black-box test cases. The tester also adds aspects assessed as relevant to the temporal system behavior, for example the simultaneous occurrence of several events or time-consuming system states. However, test cases determined with the classification-tree method focus mainly on the examination of logical correctness. Afterwards, the second step of our test strategy concentrates on the examination of temporal correctness. The test data specified for the systematic test is used as initial population for the optimization of execution times by means of evolutionary testing – as described in section 3. Thus the genetic search for the shortest and longest execution times benefits from the tester’s experience and his domain knowledge [Wegener et al., 1996]. Figure 9 illustrates the suggested test strategy for real-time systems.

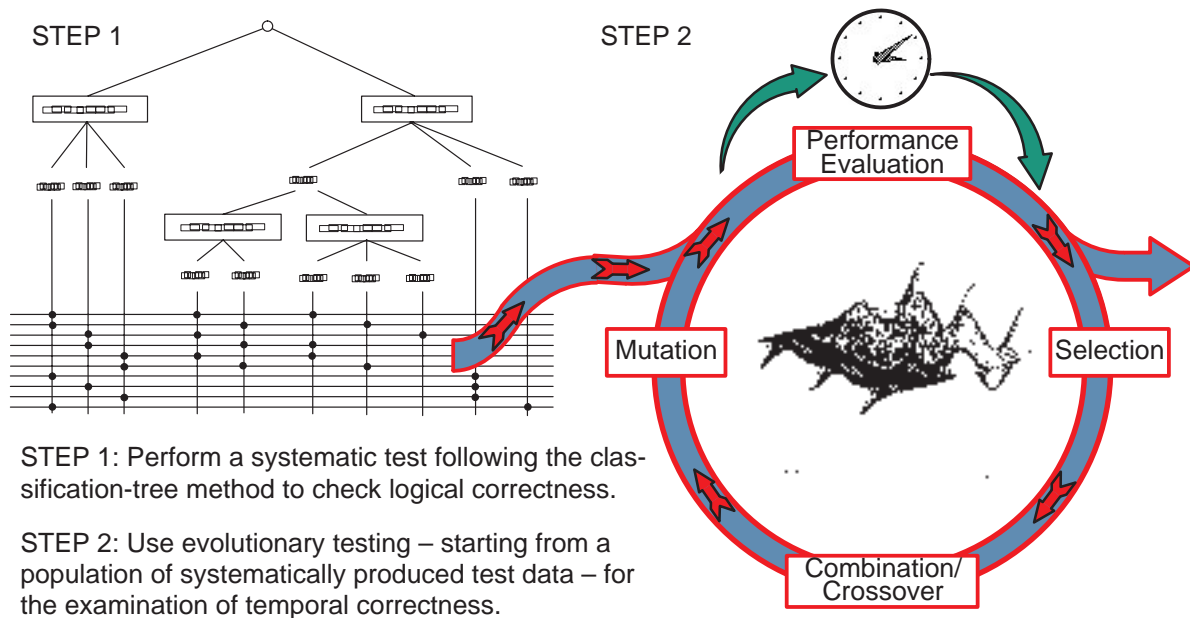


Figure 9: Test Strategy Suggested for Real-Time Systems

In principle structure-oriented test cases are also suited as initial population for the genetic search because there is a close correlation between temporal behavior and program structure. The number of processor cycles measured will generally be directly related to the number of statements in the control flow path, though there will be exceptions because some statements require more cycles than others [Wegener et al., 1997b]. In this case evolutionary testing will benefit from the tester’s knowledge of the internal program structure. Another idea for further improvement is to link evolutionary testing directly with structural testing. The fitness-function could be expanded in such a way that individuals which execute a new program branch or path would get a high fitness-value to ensure their survival in the next generation. Thus the diversity of the population would not only be maintained with respect to the temporal behavior of individuals but also in consideration of the test object’s internal structure. From this follows that the new program structures would be executed several times in the next generations. If no longer execution times resulted from this, the corresponding test sets would become extinct again during subsequent generations.

5 Conclusion and Future Work

The correct functioning of real-time systems depends critically on their temporal correctness. Testing is the most important analytical method for the quality assurance of such systems. An investigation of existing testing approaches showed a lack of support for testing the temporal behavior. Therefore, existing test procedures must be supplemented by new methods and tools. In various experiments evolutionary testing has been successfully applied to search the longest and shortest execution times of real-time programs in order to check whether they violate the specified timing constraints. Compared with random testing, evolutionary testing always obtained better results. Compared with systematic testing, evolutionary testing seems to be better for complex test objects where the temporal behavior is difficult to assess by systematic testing.

Further improvements are possible through the combination with systematic test methods. If the genetic search does not start with a randomly generated population but with a set of test data systematically determined by the tester, the disadvantage of genetic algorithms that they might not find certain test relevant value combinations can be compensated for. Moreover, depending on the test method applied, evolutionary testing benefits from the tester's knowledge of the program function or the program structure.

Evolutionary testing shows considerable promise in testing and validating the temporal correctness of real-time systems and further research work in this area should prove fruitful. More work is needed to find the most appropriate parameters for genetic algorithms and to define suitable criteria for the decision when to stop the search [Sullivan et al., 1998]. Further studies are focusing on the question how stagnations can be reacted to with appropriate changes of the search strategy.

References

- Boden, E.B., and Martino, G.F. (1996). *Testing Software Using Order-Based Genetic Algorithms*. In Koza, J.R. et al. (eds.). Proceedings of the First Annual Conference on Genetic Programming, 28 - 31 July 1996, Stanford University. The MIT Press, Cambridge, USA, pp. 461 - 466.
- Davis, C.G. (1979). *Testing Large, Real-Time Software Systems*. Software Testing, Infotech State of the Art Report, Vol. 2, 1979, pp. 85 - 105.
- Davis, L. (1996). *Handbook of Genetic Algorithms*. International Thomson Computer Press, Boston, USA.
- Grimm, K. (1996). *Systematic Testing of Software-Based Systems*. Proceedings of the 2nd Annual ENCRESS Conference, June 1996, Paris, France.
- Grochtmann, M., and Grimm, K. (1993). *Classification Trees for Partition Testing*. Software Testing, Verification & Reliability, Vol. 3, No. 2, pp. 63 - 82, Wiley.
- Grochtmann, M., and Wegener, J. (1995). *Test Case Design Using Classification Trees and the Classification-Tree Editor CTE*. Proceedings of Quality Week '95, 30 May - 2 June 1995, San Francisco, USA.
- Heath, W.S. (1991). *Real-Time Software Techniques*. Van Nostrand Reinhold, New York, USA.
- Jones, B.F., Sthamer, H.-H., and Eyres, D.E. (1996). *Automatic Structural Testing Using Genetic Algorithms*. Software Engineering Journal, Vol. 11, No. 5, pp. 299 - 306, IEE & BCS, Stevenage, UK.
- Jones, B.F., Sthamer, H.-H., Yang, X., Eyres, D.E. (1995). *The Automatic Generation of Software Test Data Sets using Adaptive Search Techniques*. Proceedings of Software Quality Management '95, Seville, Spain, pp. 435 - 444.
- Pohlheim, H. (1996). *GEATbx: Genetic and Evolutionary Algorithm Toolbox for Use with Matlab – Documentation*. Technical Report, Technical University Ilmenau.
- Roper, M. (1996). *CAST with GAs – Automatic Test Data Generation via Evolutionary Computation*. Computer Aided Software Testing (CAST) Tools, IEE Colloquium C6, Digest No. 96/096.
- Schultz, A.C., Grefenstette, J.J., and De Jong, K.A. (1993). *Test and Evaluation by Genetic Algorithms*. IEEE Expert, Vol. 8, No. 5, pp. 9 - 14, IEEE Computer Society.
- Sthamer, H.-H. (1996). *The Automatic Generation of Software Test Data Using Genetic Algorithms*. PhD Thesis, Department of Electronics and Information Technology, University of Glamorgan, Wales, UK.
- Sullivan, M., Voessner, S., Wegener, J. (1998). *Testing Temporal Correctness of Real-Time Systems – A New Approach Using Genetic Algorithms and Cluster Analysis*. Proceedings of EuroSTAR '98, 30 November - 4 December 1998, Munich, Germany.
- Watkins, A.E.L. (1995). *A Tool for the Automatic Generation of Test Data Using Genetic Algorithms*. Proceedings of Software Quality Conference '95, July 1995, Dundee, Scotland.
- Wegener, J., Grimm, K., Grochtmann, M., Sthamer, H.-H., and Jones, B.F. (1996). *Systematic Testing of Real-Time Systems*. Proceedings of EuroSTAR '96, 2 - 6 December 1996, Amsterdam, Netherlands.
- Wegener, J., Grochtmann, M. (1998). *Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing*. to appear in Real-Time Systems.
- Wegener, J., Grochtmann, M., Jones, B.F. (1997a). *Testing Temporal Correctness of Real-Time Systems by Means of Genetic Algorithms*. Proceedings of Quality Week '97, 27 - 30 May 1997, San Francisco, USA.
- Wegener, J., Sthamer, H.-H., Jones, B.F., and Eyres, D.E. (1997b). *Testing Real-Time Systems Using Genetic Algorithms*. Proceedings of Software Quality Management '97, 24 - 26 March 1997, Bath, UK.
- Xanthakis, S., Ellis, C., Skourlas, C., LeGall, A., and Katsikas, S. (1992). *Application of Genetic Algorithms to Software Testing*. 5th International Conference on Software Engineering, December 1992, Toulouse, France.

A Software Engineering View of Data Quality

Mónica Bobrowski
Universidad de Buenos Aires

Joint Work with
Martina Marré and Daniel Yankelevich

Outline

- Motivation
- The Data Quality Problem
- What is Data Quality?
- Software Engineering and Data Quality
 - Measuring Data Quality
 - Testing Data Quality
 - Data Quality in the Software Development
- Conclusions and Future Work

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Which one is my name?

- Mónica Bobrowski
- Monca Bobrowski
- Mónica Bobrowsky
- Mónica Bobrovsky
- Mónica Brobrovsky
- Mónica Bobrosky
- Mónica Bovrosky
- etc.

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

The client point of view

I could not sign for a new house because I was found in Fidelitas records as not trustable. Of course it wasn't me!! I've put a lawsuit against them. They made me lose a lot of money!



Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

The company point of view

We are having troubles with our people identification system. We are losing our clients confidence and we are losing lots of money too!



Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

The Data Quality Problem

- Even with good software systems (and marketing staff) we can get bad results
Organizations cannot use their systems because of the data
Economic impact of poor quality data
Bad data is a problem that has to be

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

What is Data Quality?

- Quality: a relative concept
- Data attributes:
 - Accuracy
 - Timeliness
 - Usability, etc.
- Quality Data does not necessarily mean

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Software Engineering and Data Quality

- *“The system works perfectly. Of course, if wrong data is being loaded, what can the*
- Poor system design may lead to bad data

Software engineering has to deal with data quality problems!!!

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Software Engineering and Data Quality

- Software Engineering has been dealing for long with quality problems

Product and Process Quality

How can we deal with data quality

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Data Quality in Software Engineering

- Measuring Data Quality
- Testing Data Quality
- Data Quality in the Software Development

We want to use existing techniques!!!

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Measuring Data Quality

- “If you can’t measure it, you can’t manage

Value of the information
How and what to improve

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Measuring Data Quality

- Identify interesting attributes (dimensions, Wang, Strong and Lee, 1997)

Use existing techniques (e.g., GQM,

Measure the quality of the data instance and

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Testing Data Quality

- “Data is not a problem for software

System testing concentrates on system

But systems may use data generated by

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Testing Data Quality

- Verify data quality independently of the systems that may use it
 - Complete validation of all data
 - Statistical indicators
 - Testing techniques to define and execute test

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Using Known Software Testing Techniques

- Define testing criteria based on quality

Notion of coverage

Construct test cases

Need for an Oracle

≈ Structural Testing?

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Data Quality in the Software Development Process

- We have functional and non-functional requirements in traditional software

We define them at the early stages of the development process, and verify them

Data Quality Requirements are Non-functional Requirements

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

16

Data Quality Requirements

- Define them at the early stages of the development process

Using formal notations

Incorporate them to the final product

Verify them using:

- Data Quality Metrics
- Data Testing

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Conclusions and Future Work

- Organizations need quality data in order to

Some data quality dimensions may be incorporated to software systems

Software engineering is able to contribute to

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

Conclusions and Future Work

- Research program:
 - Define a set of metrics (what and how to

Define data testing criteria based on dimensions

Define data quality requirements as non-functional requirements
- *Empirical Validation is Mandatory!!!*

Bobrowski, Marré, Yankelevich: A Software Engineering View of Data Quality

A Software Engineering View of Data Quality

Mónica Bobrowski, Martina Marré, Daniel Yankelevich

Departamento de Computación, FCEyN, Universidad de Buenos Aires, Argentina

{monicab,martina,dany}@dc.uba.ar

Abstract

Thirty years ago, software was not considered a concrete value. Everyone agreed on its importance, but it was not considered as a good or possession. Nowadays, software is part of the balance of an organization. Data is slowly following the same process. The information owned by an organization is an important part of its assets. Information can be used as a competitive advantage. However, data has long been underestimated by the software community. Usually, methods and techniques apply to software (including data schemata), but the data itself has often been considered as an external problem. Validation and verification techniques usually assume that data is provided by an external agent and concentrate only on software.

In this work, we present different issues related to data quality from a software engineering point of view. We propose three main streams that should be analyzed: data quality metrics, data testing, and data quality requirements in the software development process. We point out the main problems and opportunities in each of them.

Keywords: Software Quality, Data Quality, Software Engineering.

1. Introduction

Thirty years ago, the software owned by an organization was not considered a concrete value. Everyone agreed on the importance of software, on its virtual value, but it was not considered as a good, as a possession. In those days, the value of software was defined by its cost.

Nowadays, software is part of the balance of an organization, it contributes to its value, and for almost every software project the ROI is calculated. Data is slowly following the same process. In fact, people is now talking about “the value of information.” Many organizations want to possess information. Managers know that having the right information at the right time may lead them to obtain great benefits. Moreover, organizations have information that may help them to improve their work, make decisions, and increase their profits. This information is usually stored in large databases accessed via software applications. However, it is not enough to have good applications; an organization needs good data in order to achieve its goals.

Now, how could an organization know that it has the right information at the right time? How could an organization evaluate its information? That is a matter of *data quality*. In fact, the quality of information is crucial when determining its usefulness. When quality is not achieved, information is not used, or leads to incorrect decisions, and even loss. As it is known, “decisions are no better than the data on which they are based” [Red98]. But, what does information quality mean?

In recent years, researchers have been studying data quality problems from the perspective of the data generation processes [SLW97, SLW97, WW96]. They have identified problems in data, and tried to associate them with problems in the process that lead to this data. The underlying idea is that improving the process

may lead to an improvement in data.

In [Red96], Redman gives a deep introduction to DQ issues. He points out many aspects of data quality: definition, management, policies, experiences, requirements, measurements, etc. Although his approach differs from ours (it is mainly statistical, and concentrate on the data generation process), his book offers clear examples, motivations, definitions, and useful tips.

Data quality problems are well known to practitioners. In fact, many failures of software are not due to poor quality of the systems, but to inconsistencies or other problems in data. The quality of data has a great impact on the usefulness and overall quality of software systems.

However, the mainstream of Software Engineering ignored data quality issues up to day. Validation and verification techniques exist and have been used for software processes and products, but few has been done related to data. The only concern for computer engineers regarding data quality has been the extraction of data for data warehouses. In the context of Data Warehousing, an European project investigated quality and in particular the requirements on data needed to implement a data warehouse [JV97].

In our view, software engineers must take into account data quality issues in the design, validation, and implementation of software systems. Moreover, standard techniques can and should be applied to these problems. In this paper, we present different issues related to data quality from a software engineering point of view. We point out three main streams that should be analyzed and the main problems and opportunities in each of them. These themes are:

- Data quality metrics

Measuring the quality of the information will help us to know its value, and also its pitfalls. We will be able to know how valuable our information is, and also what we need to improve in order to increase quality. Moreover, measuring quality would clarify the goals of a quality improvement strategy or process. We agree with the well-known proverb: "if you can't measure it, you can't manage it."

- Data quality and testing

Usually, testers and engineers assume that the data (in a production environment) is correct, and test the system considering its behavior. However, as we have said, this is not the case in the real world. When a new system is incorporated to an existing environment, the data it uses must be analyzed to understand its usefulness. Moreover, an old system may be using corrupted data. We believe that the verification of a system must include the verification of the data it works on. Besides, we believe that many testing techniques can be adapted in order to be used to test data.

- Data quality in the software development process

The dimensions in which data quality is analyzed (for instance consistency, accuracy) can be considered data quality requirements for a project, and can be assessed from the beginning of the software development process in the same way that we have functional and non-functional requirements. In fact, they are a particular sort of non-functional requirements. So, we want to deal with them from the beginning of the process, and incorporate them to our specification, our design, and our system implementation. In this case, our system should give us a warning when the data is in close or already in a "bad quality" situation with respect to our requirements. And hence we can prevent our system from entering such a situation.

In Section 2 we discuss what data quality means. In Section 3 we describe typical data quality problems. Section 4 presents data quality metrics: why, what, and how to measure. Section 5 is devoted to data testing. Section 6 points out why data quality requirements should be incorporated to the software development process. In section 7 we present our conclusions and future work.

2. What is Data Quality?

It is difficult to give a universal definition of what quality means. When we talk about quality we do not always refer to the same concept. We will try to exemplify this issue. Suppose that you are planning a trip to a foreign country. You have to choose an airline to fly. Which one do you prefer? Of course, you will prefer the airline that offers the *best* quality. But, what does quality mean? You want to arrive in time, you want comfortable seats, you want a helpful crew, you want a quiet trip, and you want low prices. These attributes

(punctuality, comfort, helpfulness, peace, low prices) constitute your notion of quality in this particular context. Even in the trip situation, someone else may not be concerned about the price, but be very worried about the meals served. So his notion of “airline quality” is different from yours. It may differ not only in the attributes taken into account; the relevance of each item may be different. Moreover, you could have different notions of “airline quality” for different trips.

This example shows that quality is not an absolute concept. The word *quality* by itself has not a unique connotation. We have to make assumptions on which aspects apply on a particular situation. In the case of data quality, we may want to take into account only specific attributes with some specific relevance, depending on the particular context we are analyzing. In our view, the quality of data in the context of software systems is related to the benefits that it might give to an organization.

As we have said, the quality of data depends on several aspects. Therefore, in order to obtain an accurate measure of the quality of data, one have to choose which *attributes* to consider, and how much each one contributes to the quality as a whole. In what follows, we present several attributes that we think may determine the quality of our data. These attributes or *dimensions* have been taken from [WW96, SLW97] following the point of view of the value of the data, i.e., our pragmatic view of data quality.

We present an informal definition for each of the attributes considered. This selection is not exhaustive, but is representative enough for our purposes.

Completeness	Every fact of the real world is represented. It is possible to consider two different aspects of completeness: first, certain values may not be present at the time; second, certain attributes cannot be stored.
Relevance	Every piece of information stored is important in order to get a representation of the real world.
Reliability	The data stored is trustable, i.e., it can be taken as true information.
Amount of data	The number of facts stored.
Consistency	There is no contradiction between the data stored.
Correctness	Every set of data stored represents a real world situation.
Timeliness	Data is updated in time; update frequency is adequate.
Precision	Data is stored with the precision required to characterize it.
Unambiguous	Each piece of data has a unique meaning.
Accuracy	Each piece of data stored is related to a real world datum in a precise way.
Objectivity	Data is objective, i.e., it does not depend on the judgment, interpretation, or evaluation of people.
Conciseness	The real world is represented with the minimum information required for the goal it is used for.
Usefulness	The stored information is applicable for the organization.
Usability	The stored information is usable by the organization.

Notice that dimensions may be related to others. For example, the amount of data may be important only in conjunction with correctness (lot of incorrect data has no sense, and even may damage the organization), usability (inefficient access to data due to the size of the database is worthless), and so on. In some way, these attributes complement each other.

3. The Data Quality Problem

- I can't use this application. Look, I know that wells in this field are at most 3000 feet, and the depth of this well in the system is 4500! This system is useless.

- The system works perfectly. Of course, if wrong data is being loaded, what can the computer do?

- I don't know. I just say that it is not good for me. It makes me loose more time looking for data than before.

- It is not our problem. The system works, we detect wrong data when it is loaded –and in the cases YOU specified-. It is a problem of the users: you should tell them to use it right.

This dialog, at least in spirit, happened in many places many times. Data quality problems are real problems in most information systems. With different degrees of criticality and deepness, these problems are being treated in many organizations. In most cases, in an ad-hoc way.

For instance, let us consider mailing lists. How many times do you usually receive a brochure for a conference? How many combinations of first, second, and last name have you seen your name on envelopes? This simple example shows how expensive data quality errors can be: mailing can be quite expensive, and using a faulty list, a mailing campaign can be many times more expensive. Moreover, the lost caused by wrong advertising goes long beyond the cost of mailing: customers and potential customers do not trust someone that is not even capable of keep his/her data right. The image of the organization suffers offering and using wrong data.

However, the particular case of names (and, mainly, occidental names) has been extensively studied and many heuristics have been proposed for the problem of determining whether two names correspond to the same person (in general, in the presence of more data, like date of birth, addresses, etc.). Commercial products and algorithms are available to attack (not to solve) this problem. Even though, this particular problem is cause of misuse of systems in several different contexts.

For instance, criminal identification systems determine if a person has criminal records. This information is used, for example, by judges (to decide whether the person has to be punished, and how), and by organizations (to decide whether to hire him). These systems are critical because, in some sense, our future may depend on the quality of the data and the procedures used to recover it. Although a high quality application is used to access the data, the identification is based on several attributes, and sophisticated algorithms are used to match them, it turns out that wrong conclusions can be obtained when bad quality data is present in the system. It has been found that 50-80% of computerized criminal records in the U.S. were found to be inaccurate, incomplete, or ambiguous [Tay98]. This poor quality data may imply send people to jail or not to hire them.

Data quality problems are not only related to pattern matching of persons or organizations. Such problems arise in many different contexts, and the consequences can be disastrous. The cultural change imposed by the use of computers in many different environments, only makes the problem worse. In fact, people trust computers and utilize them as the main source of data: digital information is used minute by minute to take important decisions that affect people lives.

Recently, data warehouse and data mining projects exposed many data quality problems in big enterprises. When the information collected was analyzed or was checked for integrity, some “hidden” problems were detected. For instance, data from different sources was detected to be inconsistent in data warehousing.

The usual (implicit or explicit) position of software professionals facing data quality issues is that “this is not our problem.” Somehow, information professionals are not responsible of dealing with information.

On the other hand, we have two ideas that contradict that belief. First, some data quality issues are caused by poor design of software systems. In particular, the effect of poor interface design on data quality is direct. For instance, in many cases users of a complex interface with mandatory values have the tendency to choose a random value. If there is a list of values available, users choose the first of the list or the default value.

For example, by studying last year information the managers of a hospital discovered that most of the patients suffered from hemorrhoids. The resources of the next year were assigned on this basis. The number of beds,

nurses, and other resources needed were determined using this information. However, it came out that “hemorrhoids” was the default choice at the check-in application, and clerks selected it because it was difficult to look for the correct choice. This bad data -due to a poor interface design- had terrible consequences on the hospital finances [Tay98].

Another way in which poor design may affect the quality of the data is by failing in a complete analysis of business rules or by not taking into account data quality issues during the requirements analysis phase. In fact, if data quality is a risk, the design of the system must take measures to minimize that risk.

A rule of thumb [Orr98] proposes to improve data quality by increasing the use of the data. Data that is not used cannot be maintained. We agree with this rule. However, several times it has been used to illustrate that problems in the quality of information are not caused by poor design. This is not true. The process of creating and using data must be subsumed in the design of the system. The data flow, the organization of the processes, and the overall design must be created with this data life cycle in mind [Red96]. Not to do so is a modeling and design fault. During the analysis phase, the processes that are automated must be analyzed not only for efficiency: data quality is also a driver when designing the processes and the use of the applications.

The second idea is that many data quality problems can be prevented and deal with by using standard software engineering techniques – adequately adapted or revisited. For instance, configuration management techniques could be used to solve problems with out-of-date data or versioning of information. Standard metric definition techniques could be used to define useful data quality metrics. These ideas are addressed in more detail in the following sections of this work.

4. Measuring Data Quality

The first step to improve data quality and to define methods and techniques is to understand what “good quality” and “bad quality” is. Hence, we need to measure data quality to be able to know how valuable the information is, and how to improve it. Measuring the quality of the information will help us to know its value, and also its pitfalls. We will be able to know how valuable our information is, and also what we need to improve in order to increase quality. Moreover, measuring quality would clarify the goals of a quality improvement strategy or process. We agree with the well-known proverb: “if you can’t measure it, you can’t manage it”. In fact, it is not possible to make serious empirical analysis of techniques or strategies if there is no agreement on how the results will be evaluated.

We propose to measure information quality using metrics defined using traditional software engineering techniques. Metrics have been deeply studied in software engineering [FP97], so we want to take advantage of it.

In [BMY98] we present a framework for defining and using data quality metrics. The outcome of this work is a suitable set of metrics that establish a starting point for a systematic analysis of data quality. We identify the attributes we want to measure, and obtain a set of metrics and techniques to calculate them. This is a starting point for a systematic analysis of data quality, that may lead to improve the quality of the data in an organization.

We base our work on the GQM methodology [BR88]. GQM is a framework for the definition of metrics. GQM is based on the assumption that in order to measure in a useful way, an organization must:

- specify goals,
- characterize them by means of questions pointing their relevant attributes,
- give measurements that may answer these questions.

We have chosen this framework because it is a top down approach that provides guidelines to define metrics, without a priori knowledge of the specific measures. There are other approaches for metric definition, e.g., [BBL76, MRW77]. We have chosen GQM because of its simplicity, its adequacy to our problem, and because it is well known and proven in software engineering applications [Van98].

Following GQM, we first are able to state which dimensions characterize our notion of data quality. Then, we can ask questions characterizing each dimension, without giving a precise (formal) definition -that is

sometimes impossible-, only focusing on their relevant characteristics from our point of view. Finally, we give metrics (some objective, some others based on people appreciation) to answer these questions, giving us a more precise valuation of the quality of our data.

We cannot measure data and ignore how it is organized. Certain quality characteristics are related to the organization of data, i.e., to the *data model*, and not to data itself. The data model might affect some data quality attributes, since it defines the way data is accessed and maintained. We want to identify and measure those attributes too, and complement measures of data with information on how it is organized. As a consequence, we defined two kinds of metrics: set of data metrics, and data model metrics.

Once we have defined our data quality metrics (i.e., what and how to measure) we want to use them. We can simply take our relational database, identify the dimensions we are interested in, choose the appropriate metrics and techniques depending on specific considerations, apply them, and analyze the results. This is a useful approach, specially when the system is already in production, the database is implemented, there is a lot of data loaded, and we want to have a picture of the current situation in order to decide what to improve. We may even add information about the quality of the data to the meta model, as part of its definition. This way it may be easier to check and evaluate the quality of the data at a certain point. In [JV97], this approach is followed in the data warehouse case.

Once we have measured the quality of our data with respect to the chosen dimensions, we can decide whether or not our current data satisfies our quality expectations. Moreover, we will know in which dimension it fails (although we do not know why), with respect to which specific aspect, and we have a measure of the “badness.” So we can concentrate our efforts in solving that particular problem, and we can decide if it is convenient to do so - may be data is not so bad, and the solving effort is worthless.

This procedure only deals with measuring the quality of data at certain points, and can help in deciding which corrective or preventive actions to implement. In order to reach and maintain high levels of data quality, it has to be part of a broader plan, that takes into account all the aspects of data quality in the organization (see [Red96]).

Another approach is to see the dimensions we are interested in as data quality requirements (see Section 6). These requirements can be assessed from the beginning of the software development process, in the same way that we have functional and non-functional requirements. So, we want to deal with them from the beginning, and incorporate them to our specification, our design, and our system implementation. In this case, our system should give us a warning when the data is in close or already in a “bad quality” situation with respect to our requirements. And hence we can prevent our system from entering such a situation. Metrics may be used here to establish the requirements and check them at different stages of the software development process.

5. Testing and Data Quality

Software systems were often analyzed as if they start from scratch. Only recently the idea of using COTS is being incorporated in formal description of the development process. This is even stronger in the case of the data used by these systems. The idea of testing a system concentrates on testing its functionalities: never the data that it is supposed to work with – even if it makes assumptions on what is the state of the data. The phrase “garbage in/garbage out” only expresses the idea of “data is not a problem of software systems.”

If a system is started from scratch, some of these assumptions can be accepted. However, in the daily practice of our profession, most systems are incorporated on top of existing systems or collaborating with existing systems. Many projects use data generated by other systems, in many cases by systems that are not operative anymore.

In our view, it is important to check whether the data satisfies the requirements of the system or, in other words, that the quality of data reaches the minimum level required for the system to work properly. This activity can be done before the system is developed (in order to take corrective measures or include extra components during the development), before installing the system (in order to check how it will work and to prevent problems during its use) or, independently of any system, just to measure the quality of the data.

This verification can be done in three different ways:

- Complete validation of all data.

- Statistical indicators of mean, variance, intervals, etc.; or random selection with an associated confidence.
- Use testing techniques to define and execute test cases.

The first way is clear: validate the whole data, using automatic and manual verification. This is not equivalent to clean the database or files, because the cost of repairing a data error can be many times greater than the cost of detecting it. However, in most cases complete validation is unfeasible. In this extent, it is not different of complete validation of programs [How76]: in many cases the domain of programs are finite and could, in theory, be validated for all inputs. Even though, complete verification is not done, because it is too expensive, too complex, or unnecessary. Only the thought of checking a 2,000,000 registries database to see if any customer has changed his address is scaring.

It is important to discuss the difference between the last two options. There are testing techniques based on statistics, and the activity of testing is strongly related to statistical analysis. However, there is a subtle difference between taking values that describe distribution of data on one hand, and choosing particular cases that satisfy particular criteria on the other. When we propose testing as a technique to validate data quality, we think that it is possible to define *testing criteria* for data quality based on the quality dimensions of interest. In some cases, it might be even possible to define the notion of *coverage*, and to construct *test cases* to satisfy a particular coverage criteria [My79].

For example, suppose that we are interested in measuring how accurate our data is with respect to time (timeliness). Let us assume that we know which attributes are time dependent. If we have a way to know if specific values are outdated, we may define a test over the data to estimate the number of records that are outdated. Hence, in order to implement these testing activities we need to use a selection criterion (to reduce the number of test cases to be evaluated) and an *oracle* (to know if specific values are outdated). We believe that in this case, the selection criterion should use the specific information about timeliness of data, improving the results obtained by using sampling. Testing for other qualities should use different information to select data.

Data testing has the flavor of structural testing –because the structure of data will probably play a basic role in defining the criteria- and aspects of functional testing. The type of coverage used to check data quality will be fundamental to create new testing techniques –or adapt existing techniques for new goals. A lot of work must be done to define adequate notions, and those notions must be validated by empirical data (and, possibly, by high quality data!) before proposing concrete techniques. However, it is clear that testing, as presented in this section, has many advantages over statistical analysis. One of the advantages is that we do not need to define the rules that guarantee that a particular piece of data is of high quality explicitly: for each case we can determine whether the output passes or fails the test. The only difficulty is to choose the right tests. But we know how to do that to test programs: the same ideas should apply here.

6. Data Quality in the Software Development Process

The requirements of a software system are usually divided in two groups: functional requirements and non-functional requirements. Functional requirements include the services the system is expected to provide, while non-functional requirements place constraints on the way those services must be provided [Som94]. Examples of non-functional requirements are programming languages (“the system must be implemented using C++”), performance (“the expected response time is 2 seconds”), standards (“the development process must be ISO compliant”), interoperability (“the system must communicate with the accounting system”). Moreover, non-functional requirements may be classified according to the kind of constraints they impose. So, we have process requirements (constrain the development process), product requirements (constrain the final product), and external requirements [Som94].

Besides, when describing non-functional requirements at early stages of the software development process, we assume that they should be verifiable, that is, we want to be able to decide whether the system architecture, the design, the implementation, the process model, etc., satisfy them.

As mentioned in previous sections, we claim that data quality issues are non-functional requirements that may be incorporated into the software development process. In fact, we may add a fourth kind of requirements: *data requirements*. These requirements must be placed at requirements and specification time, and they will constrain the following steps of the development process. In this way, the system constructed will satisfy

the expected levels of data quality, and we may be able to verify these data quality requirements in the system.

Very often, system developers claim that their job do not consist in understanding what the systems they develop are used for, neither the context in which the systems will be used. They just build systems that meet the requirements of the users; the users have to ensure the quality of the data in the databases [Orr98]. According to our view, if data quality requirements are formulated together with other non-functional requirements, the developer has to guarantee that those requirements are met, and consequently, that the expected levels of data quality are achieved and maintained. Of course, it is not always possible to have an a-priori knowledge of all the aspects regarding data quality, but at least a subset should be available. And the analyst is responsible for asking and obtaining this information.

Users have expected levels of data quality in mind. In fact, they obviously want data in the systems to be used, and this alone constitutes a requirement. Sometimes they have more specific demands, concerning accuracy, timeliness, security, accessibility, etc., of data. These requirements are functional by no means, since they are not related to the services the system provides. However, they are related to how the services will be implemented. For example, if a requirement is placed on the security of the data so that certain data is not accessible to every one, the system must comply with this security requirement in order to satisfy the user expectations. Hence, data quality requirements are non-functional requirements. To include data quality requirements from the beginning of the development process may help to improve the quality of the data during system usage, because the system will be designed to take care of the quality of the data according to the user needs.

Moreover, in information systems the expectations on data quality can be even stronger than the expectations on a particular functionality or operation that the system may perform.

Existent approaches consider data to be independent from the applications that use it [Red96]. This is essentially true. Organizations have information that may be used by many systems, although the data has entity by itself. However, systems are build to use this data in agreement with the rules and needs of the organization. Thus, they have to preserve the consistency of the data, make it accessible, extract useful information, maintain its quality, etc. It follows that applications must take care of data quality. And, as we already know from software development models, it is better to have them in mind from the beginning. It is always more expensive to modify existent systems in order to deal with the quality of the data, to preserve it, or improve it. It is cheaper to include them at the starting point of the development process and verify them at each stage, including the final implementation.

In order to have verifiable requirements, we would rather use formal notations that may allow us to use automatic tools to perform verification of requirements. As opposed to other requirement languages, a language for data quality specification should be decidable and quite simple.

Data quality requirements may be formulated in terms of data quality dimensions [SLW97]. Different requirements may be placed over the same set of data and data model. Hence, we want to be able to decide whether a set of requirements is sound. Moreover, we may place different requirements over different subsets of data.

There are other approaches that deal with requirements on the data at early stages of the software development process [Red96]. However, they do not follow a software engineering approach. In fact, they do not incorporate them as part of standard software development processes; they do not look for a formal, simple, and verifiable notation to describe data quality requirements; they do not apply traditional software engineering techniques to data quality problems.

Strong, Lee, and Wang [SLW97] describe common existing problems with data. They identify their source, the dimensions affected, and the impact on the organization. They propose general solutions to these problems, for instance, as guidelines to the process development and management. They do not formalize the expected data quality levels, and cannot verify if they are achieved. This analysis can be done when the problems are detected, and the experience could be used in future developments.

Redman [Red96] proposes to understand the customer needs prior to the software development. He translates user requirements into technical requirements written in natural language. Some of these requirements are formulated in terms of specific conditions over certain data (for example, “the new address must be in the

system within two weeks”). He determines which dimensions are affected by each requirement. To do this, he uses an impact matrix, where the impact of each requirement is rated as “high”, “medium”, or “low”. It is hard to verify if the technical requirements really correspond to the user requirements, since they are both informal. Also, it is difficult to verify if the technical requirements are satisfied. Moreover, with such a limited scale, it is hard to determine the desired quality levels precisely, and consequently, to verify their achievement.

7. Conclusions and Future Work

Problems in the quality of information are real problems in almost all organizations that use large databases. In this work we have discussed the characteristics of the data quality problem, in particular related to other quality topics usually considered in the software engineering field. Moreover, we have proposed three specific lines in which particular techniques could have a direct impact on how organizations deal with these problems.

This work presents more problems than solutions, it is just a particular point of view to attack data quality problems. In order to obtain concrete results, more work must be done in each of the three themes proposed. In particular, empirical validation is mandatory to check the adequacy of the methods and techniques proposed. Actually, this work can be used as a research agenda, and the lines presented are the basis of our research program on data quality.

The start point of this research program is the definition of metrics for data quality. Without a clear knowledge of what and how to measure, it is difficult to attack the underlying problems or to define objective experiments to check improvement after the use of new techniques [BMY98]. A particular issue related to this point is the value of data. At some point in our program, we would like to have a notion of value of information (in the sense of dollar value or market value), probably related to its use.

Data testing and the incorporation of data quality in the software development process are both issues that must be investigated before defining practical techniques.

One of the main conclusions of this work is that software engineers cannot ignore data quality in the day to day practice, and that many among the best practices of the field can be adapted to work with data quality.

Acknowledgements

This research was partially supported by the ANPCyT under ARTE Project grant PIC 11-00000-01856, and the ANPCyT under grant PIC 11-00000-0594.

References

- [BR88] Basili, V.R., Rombach, H.D.: The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988.
- [BMY98] Bobrowski, M., Marré, M., Yankelevich, D.: *Measuring Data Quality*, submitted for publication.
- [BBL76] Boehm, W., Brown, J.R., Lipow, M.: Quantitative Evaluation of Software Quality, *Proceedings of the Second International Conference on Software Engineering*, 1976.
- [FP97] Fenton, N.E., Pfleeger, S.L.: *Software Metrics - A Rigorous & Practical Approach*, 2nd edition ITP Press, 1997.
- [How76] Howden W. E.: Reliability of the Path Analysis Testing Strategy, *IEEE Transactions on Software Engineering*, vol. 2, 1976.
- [JV97] Jarke M., Vassiliou Y.: Data Warehouse Quality: A Review of the DWQ Project, *Proceedings of the Conference on Information Quality*, MIT, Boston, October 1997.
- [MRW77] McCall, J.A., Richards, P.K., Walters, G.F.: *Factors in Software Quality*, Rome Air Development Center, RADC TR-77-369, 1977.

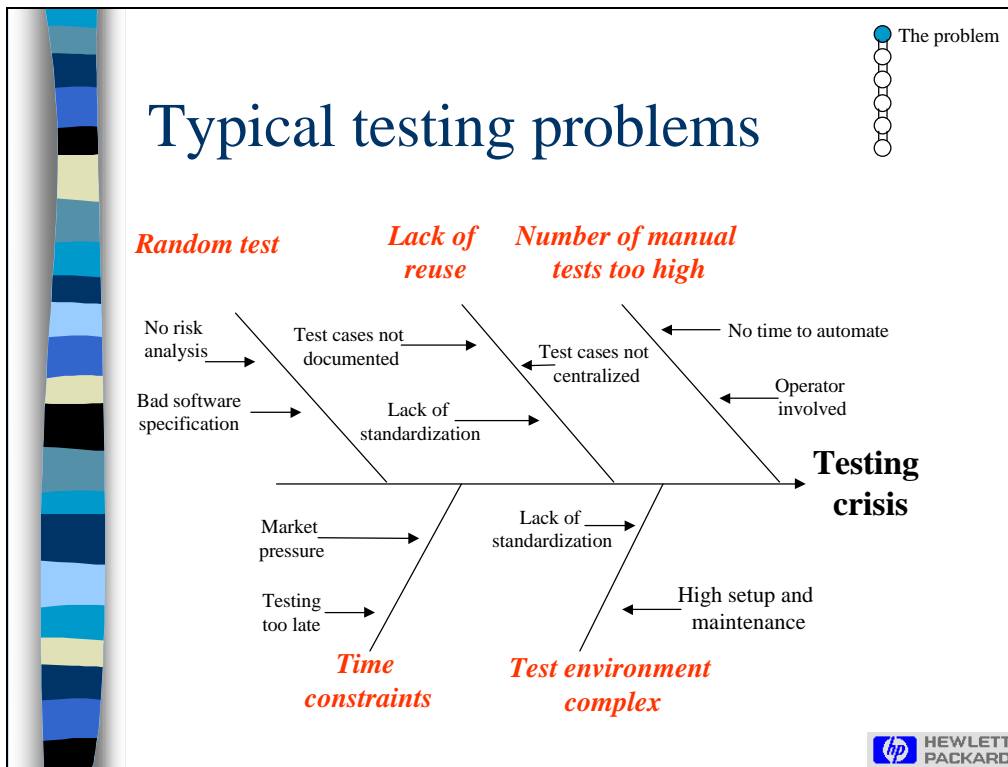
- [My79] Myers G. J., *The Art of Software Testing*, Wiley, New York, 1979.
- [Orr98] Orr, K.: Data Quality and Systems Theory, *Communications of the ACM*, Vol. 41, No. 2, pp. 66-71, Feb. 1998.
- [Red96] Redman, T.: *Data Quality for the Information Age*, Artech House, 1996.
- [Red98] Redman, T.: The Impact of Poor Data Quality on the Typical Enterprise, *Communications of the ACM*, Vol. 41, No. 2, pp. 79-82, Feb. 1998.
- [SLW97] Strong, D., Lee, Y., Wang, R.: Data Quality in Context, *Communications of the ACM*, Vol. 40, No. 5, May, 1997.
- [SLW97] Strong, D., Lee, Y., Wang, R.: 10 Potholes in the Road of Information Quality, *IEEE Computer*, August 1997.
- [Som94] Sommerville, I.: *Software Engineering*, Addison-Wesley, 1994.
- [Tay98] Tayi, G.K.: Research Seminar on Data Quality Management, Universidad de Buenos Aires, July 22th, 1998.
- [Van98] Van Latum F., Van Solingen R., Oivo M., Hoijs B., Rombach D., and Ruhe G.: Adopting GQM-Based Measurement in an Industrial Environment, *IEEE Software*, pp. 78-86, January-February 1998.
- [WW96] Wand, Y., Wang, R.: Anchoring Data Quality Dimensions in Ontological Foundations, *Communications of the ACM*, Vol. 39, No. 11, November, 1996.

Slide 1



System Test Server through the Web

Manuel Gonzalez
Software Quality Engineer
Hewlett Packard Barcelona Division
mgonzal@bpo.hp.com

Slide 2




Slide 3





The solution

Guidelines for a solution

- Zero setup and maintenance at the client side. Facilitating testing deploy, setup and maintenance.
- “*Centralization of distributed testing resources*”. Facilitating reuse.
- High control about test cases (test cases repository). Facilitating test cases documentation and standarization
- A strategy to verify testing results in an automatic way. Reducing manual testing and the need of an operator.




Slide 4





The solution

What to use to implement a solution?




- Internet technologies can bring some relief to this problem



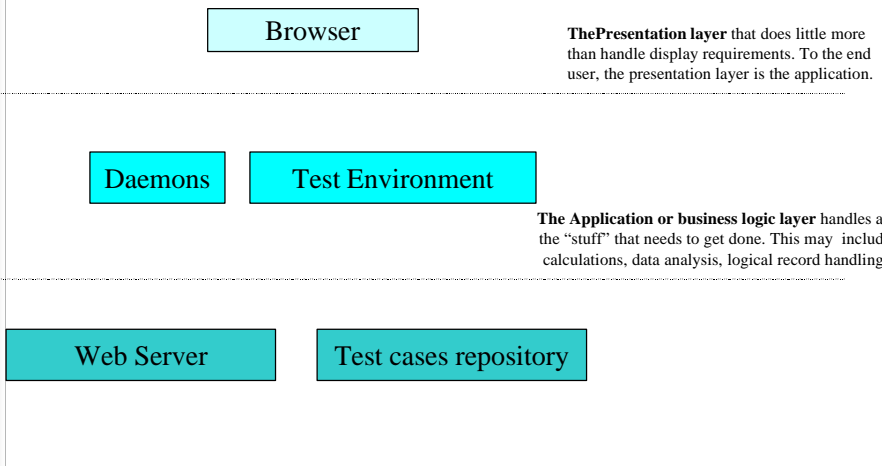


Pieces of the solution

- Test cases repository.
- Web tools and technology (Web server, browser, protocols, ...).
- Testing resources accessed via client-server paradigm.
- Simulation software (when a hardware device is also involved).



System Test Server high level Architecture



Browser


The Presentation layer that does little more than handle display requirements. To the end user, the presentation layer is the application.

Daemons **Test Environment**

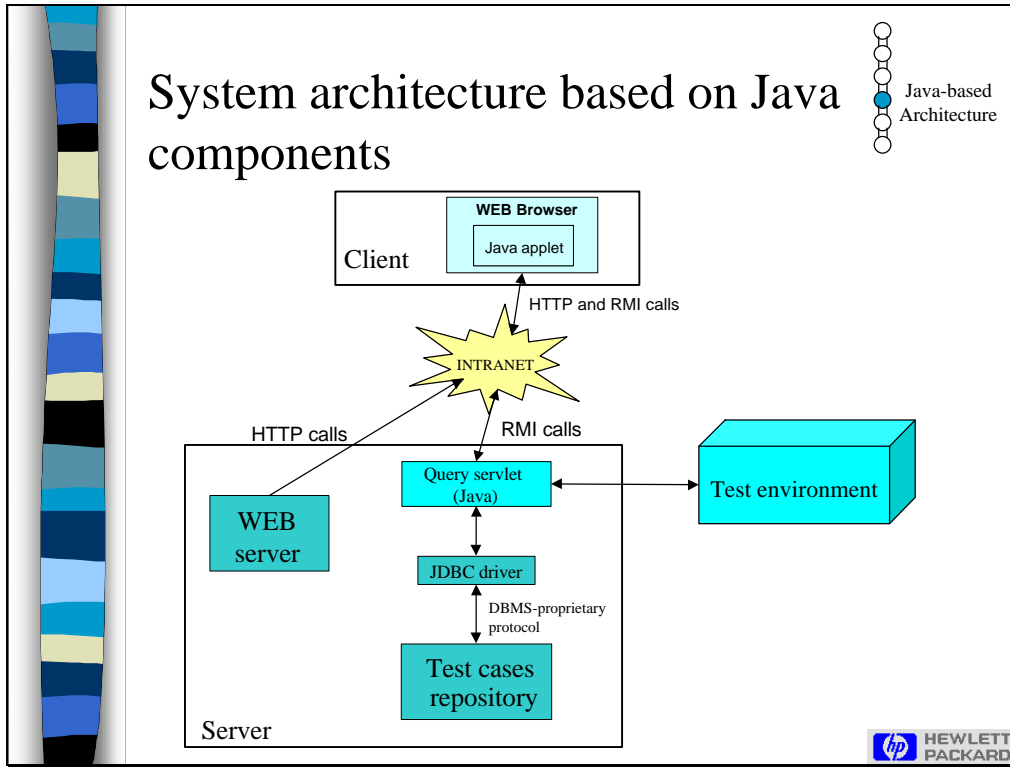
The Application or business logic layer handles all the "stuff" that needs to get done. This may include calculations, data analysis, logical record handling.

Web Server **Test cases repository**

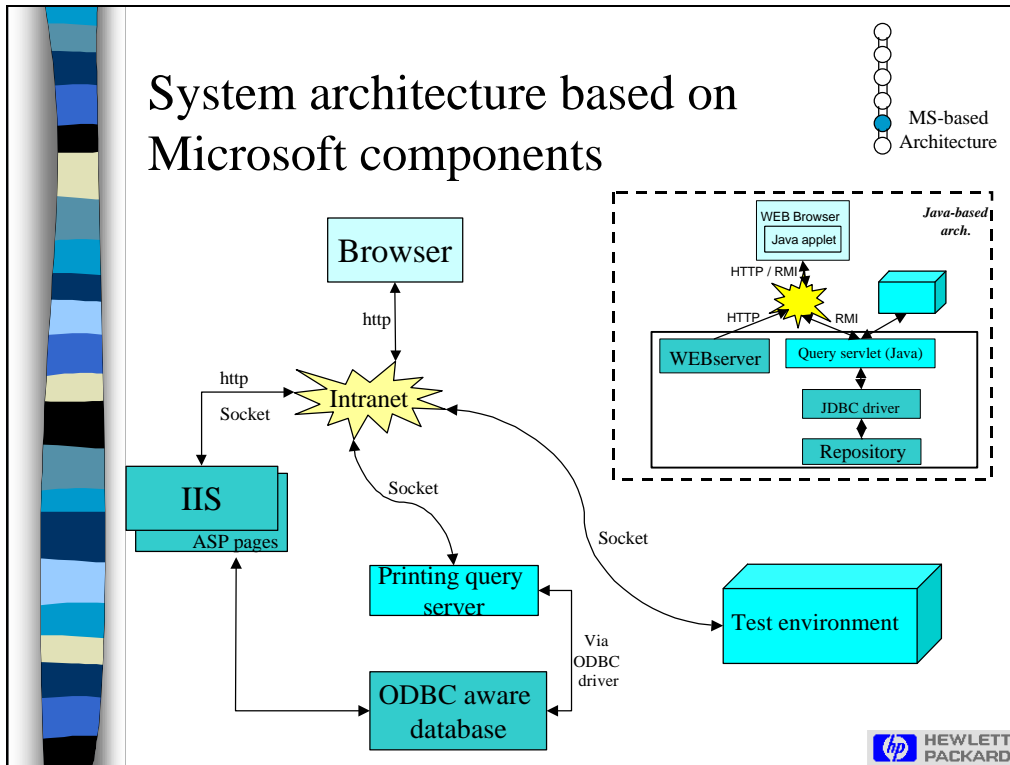
The data layer handles all retrieval and storage of information.



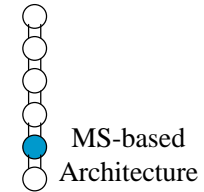
Slide 7



Slide 8



ASP Sample



In the example we will use a simple one-table database, with information about test cases. The test case-id is the primary key and it contains the test case information like name, operating system, what proves, and so on. We are going to create a SQL query that returns the test cases on Windows 95:

```
' We create command and record set objects
Set SQLCommand = Server.CreateObject("ADODB.Command")
Set TestCasesSet = Server.CreateObject("ADODB.RecordSet")

' Set the ActiveConnection property of command object to the ODBC source we will connect with
SQLCommand.ActiveConnection = "ODBC_source"

' Build the SQL query
SQLquery = "SELECT * FROM TestCasesTable WHERE OS='Windows 95'"

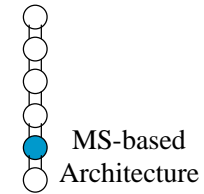
' We assign the SQL query to the CommandText property of Command object
SQLCommand.CommandText = SQLquery
SQLCommand.CommandType = 1

' Execute the command (query), and set the record set object to the result
Set TestCasesSet = SQLCommand.Execute

' Release the resources for command object
Set TestCasesSet.ActiveConnection = Nothing

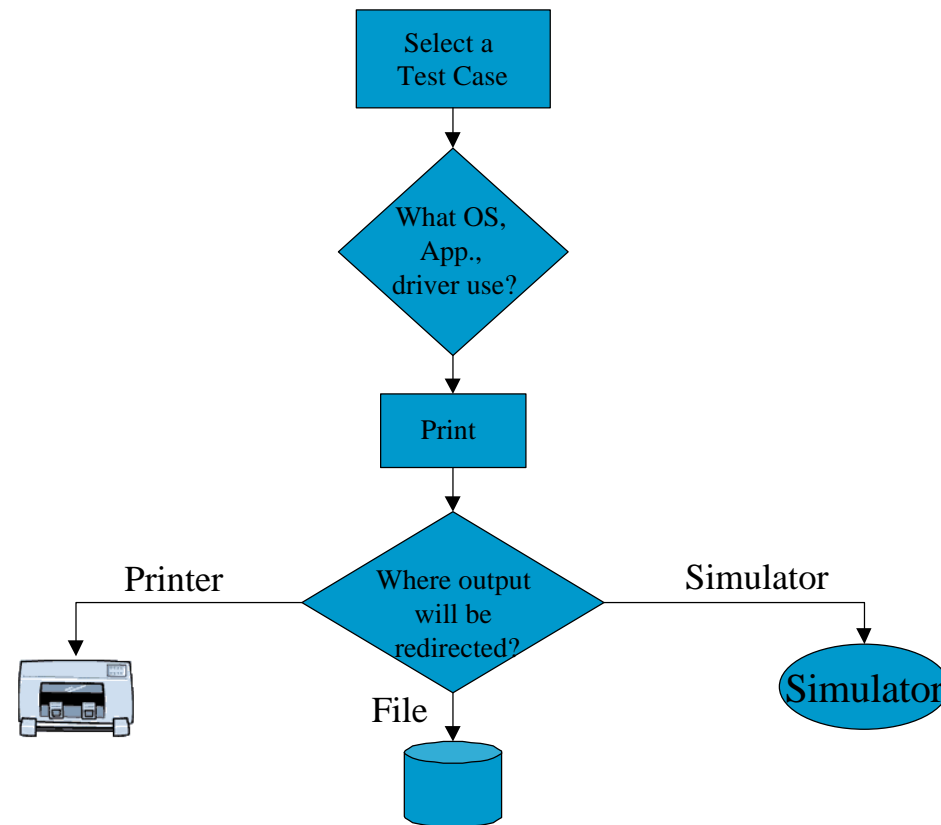
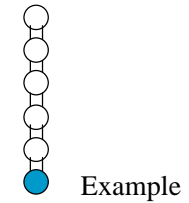
' Use query result to populate your HTML output stream
Do While NOT TestCasesSet.EOF
....
....
TestCasesSet.MoveNext
Loop
```

Differences between the two alternatives

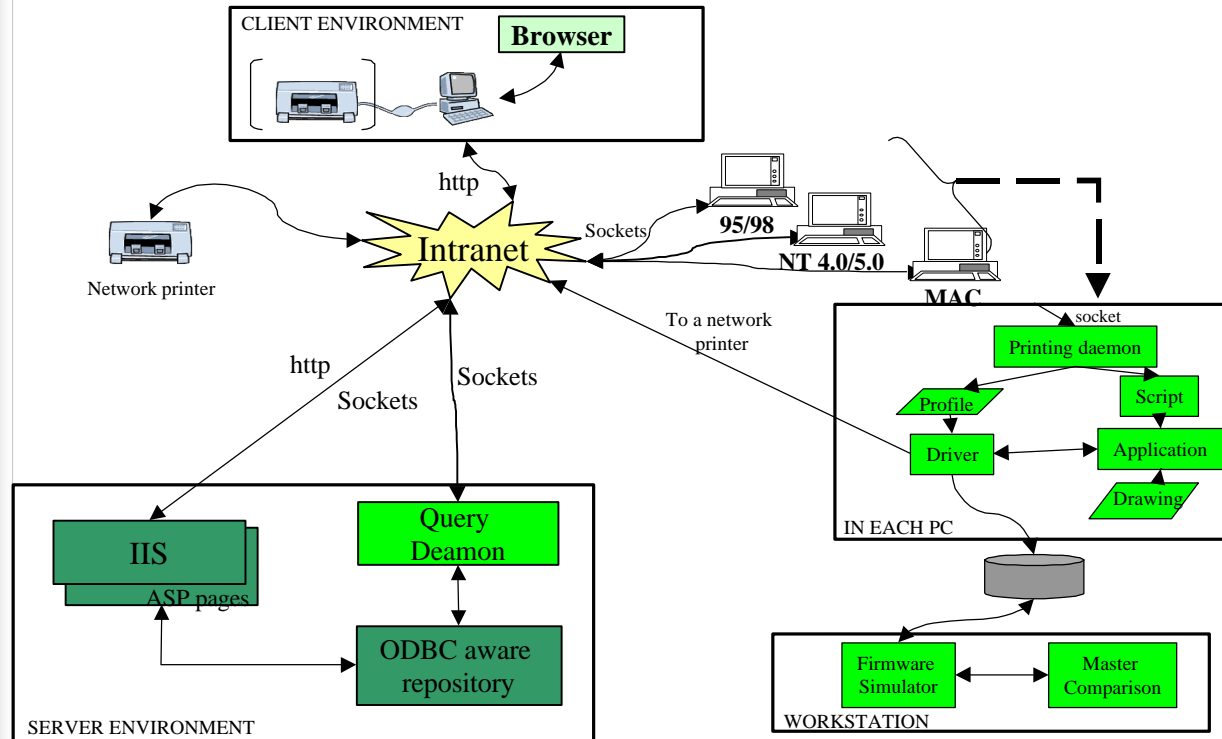
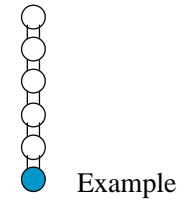


	Advantages	Disadvantages
ASP-based model	<ul style="list-style-type: none"> Visual Basic is easier to learn than Java/RMI. Browser must not have special capabilities 	<ul style="list-style-type: none"> The user interface is constrained to the HTML features. The Microsoft-SQL is not standard. The server must be the Microsoft's Internet Information Server. Therefore it's a proprietary solution.
Java-based model	<ul style="list-style-type: none"> High flexibility in the user interface layout . More flexibility in the server side. 	<ul style="list-style-type: none"> It's necessary to deal with the complexity of Java and RMI. RMI only works in the most recent browsers (browsers that support Java Platform 1.1). Browser must have enabled the Java setting (security issues)

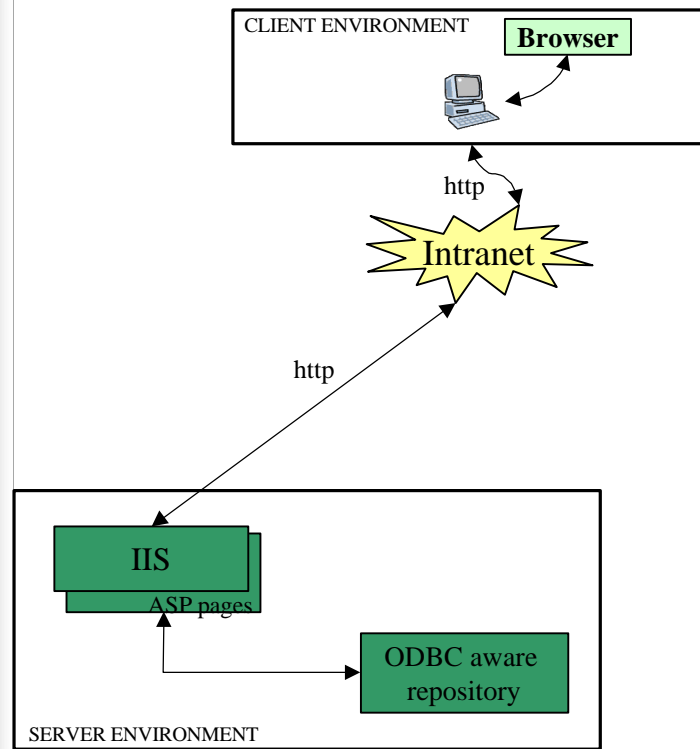
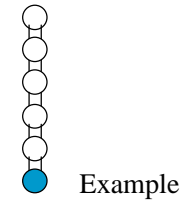
Our testing flowchart



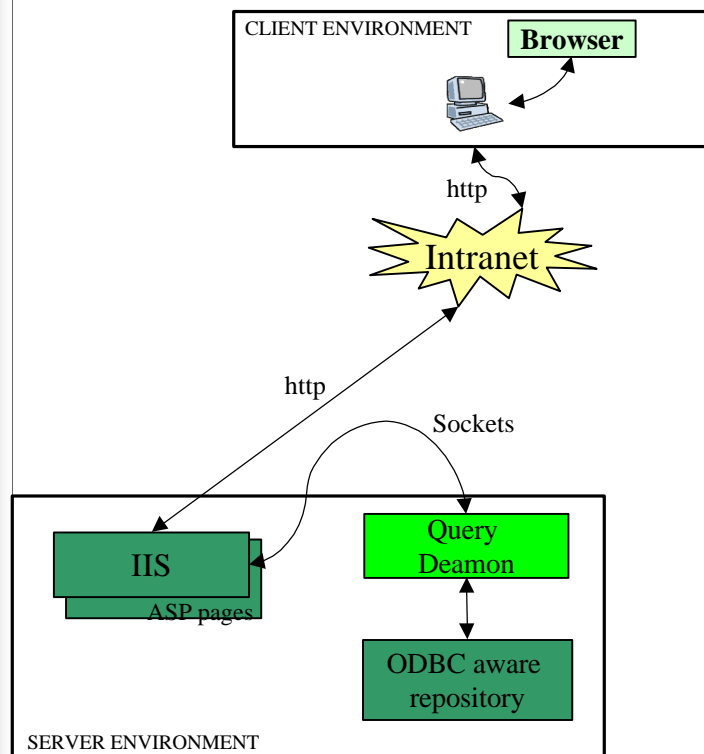
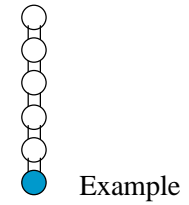
Our solution



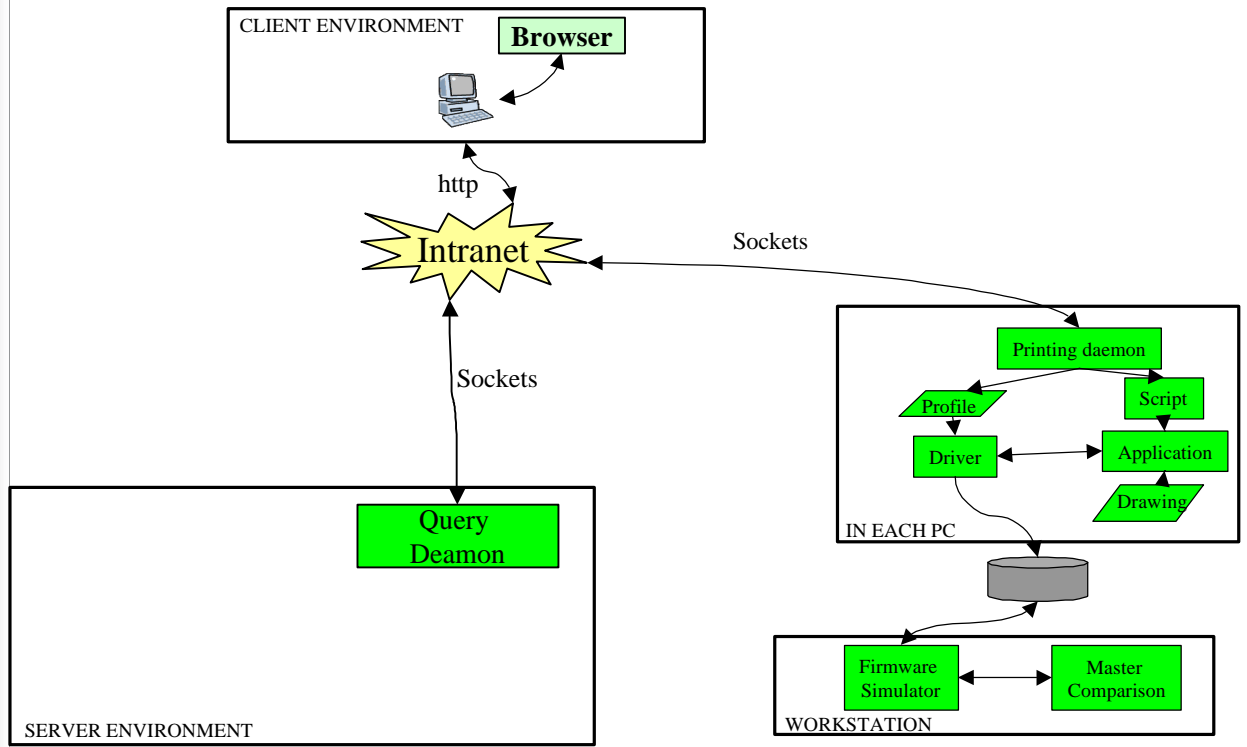
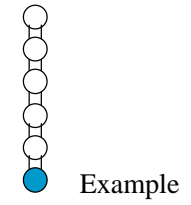
Step I



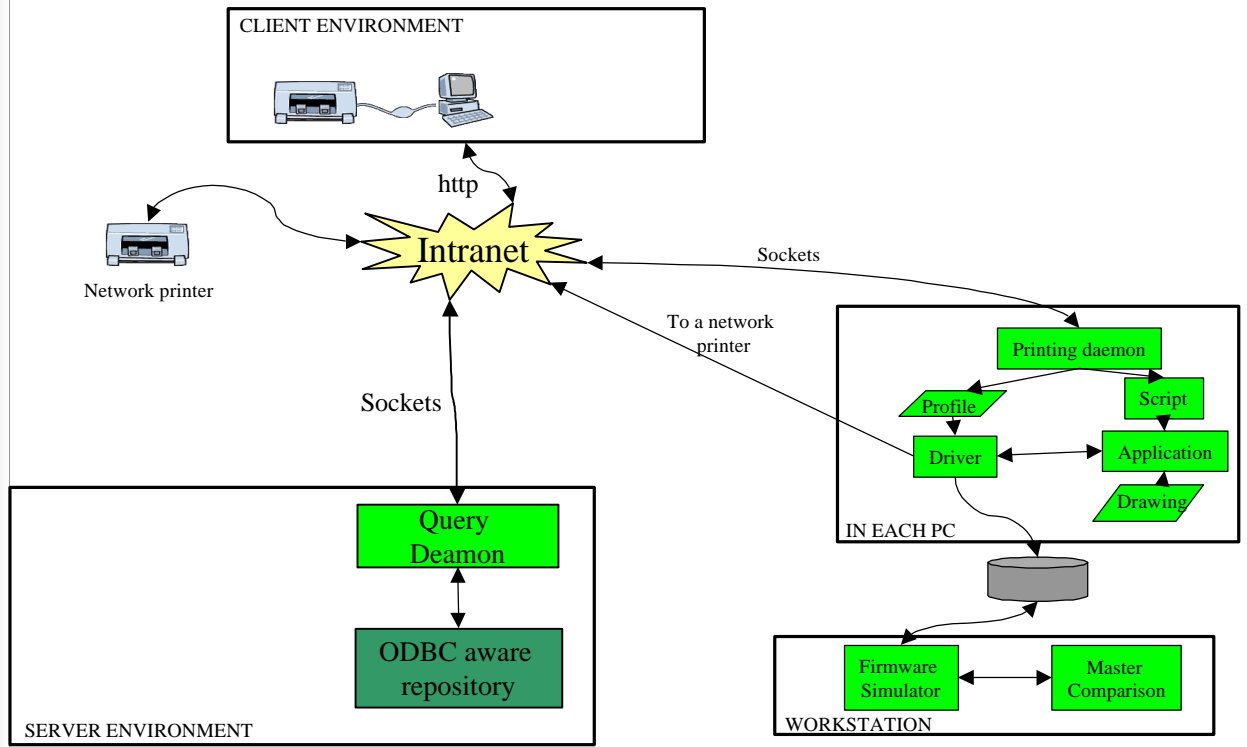
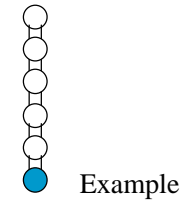
Step II



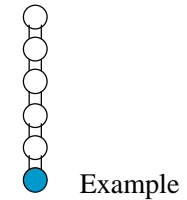
Step III



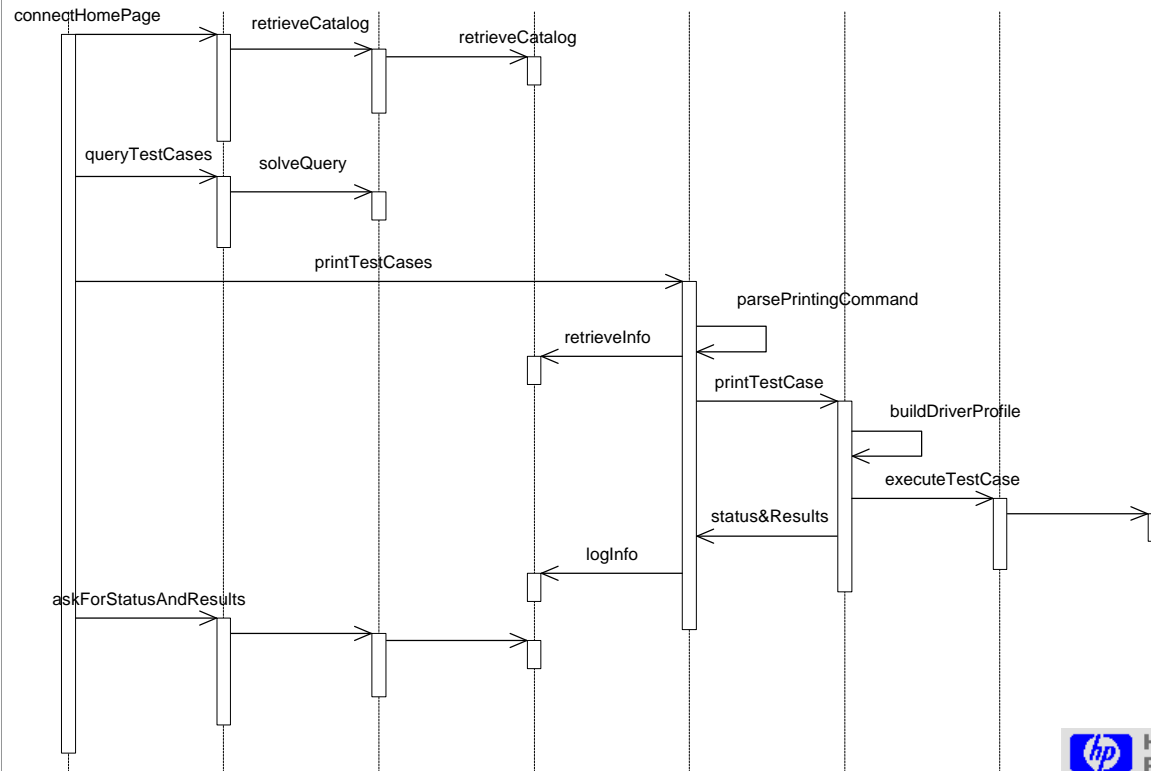
Step IV




Use Model. Sequence Diagram




Example







Use model




Example

1. A User accesses to the home page (an ASP page) of the testing environment using his favorite Web Browser.
2. The User queries the Web Server using special forms.
3. The Web Server executes ASP code in order to solve the query.
4. The ASP code accesses to the Repository and select the Test Cases that meet the query requirements. Then, it sends the query results back to the Web Browser (as HTML).
5. The User selects the appropriate test cases and then sends a printing command to the Query Deamon.




Use model



Example

7. The Query Deamon receives the printing command and parses its content. It retrieves the Test Cases from the Repository and passes them to an appropriate Printing Deamon.
8. The *Printing Deamon*:
 - Builds a driver configuration file (profile) that later will be loaded by the driver.
 - It calls a Test Script in order to execute an application, load a drawing in the application and print the drawing.
9. The output can be:
 - sent to a networked printer
 - stored on hard disk
 - simulated



System Test Server through the Web

Manuel Gonzalez
Software Quality Engineer
Hewlett Packard's Barcelona Division
Avda. Graells, 501
08190 Sant Cugat del Valles
Barcelona -Spain
mgonzal@bpo.hp.com

1. Introduction

For the Software Quality department of a mid-large company testing represents an expensive phase involving a lot of people, and unfortunately sometimes, it's done at the last stages of a project, causing strong time constraints and high cost. [Jones91]

Once the code has reached functionality complete, integration, system and regression testing [Humphrey90] [Myers76] are done in order to stabilize the product. Once the code is frozen, further regression testing is done to check that the product is ready to be shipped. In these stages of the project, schedule constrains are frequent and any time saving is welcome.

Generally, when the project is large, reuse helps to save effort, time and money. In a large project, it's typical to have several groups doing similar tasks in a similar way (for example, a printer manufacturer might have two groups developing different drivers (Postscript and HPGL/2) for the same printer, using the same test strategies but different testing environment implementations).

From our point of view, a valid solution for this problem could consist of:

- a) Facilitating the testing tasks to all project teams (not only software quality engineers, even if they are one of the main addressees). The idea is to make test execution easy enough so that it is not an expensive additional effort for people whose main task is doing something other than testing. So, test deployment through all the project teams will be easier.
- b) Having a centralized testing infrastructure reusing testing knowledge among all teams in a project. Moreover, this infrastructure must allow very fast reaction to software last minute changes at the last moment, and to run the highest amount of tests in a short period of time. In order to achieve this, a total control of test cases and an adequate mechanism for testing automation is required.

We think that Internet technologies can bring some relief to this problem. If we would combine the Web with a well-structured test case repository (and Web-aware) it would be possible to make testing accessible to everyone in the company and to share testing knowledge and technology with very low setup and maintenance.

2. How do we think that the problem may be attacked?

Test technology in a Quality department typically passes through different stages of maturity. In its beginning random test, then formal and repeatable tests are done in a manual way, finally some test automation environment gathers test knowledge trying to reduce time and cost. But when the product is complex (large software projects or hardware/firmware/software mixed projects), normally the test environment is also complex. So, testing environment setup and maintenance starts to be a problem that prevents the sharing and deployment of testing technology. In order to reduce cost and improve reuse rate we propose a testing automation environment integrating several techniques:

1. To create a test cases repository as the core of the testing automation environment which groups the test cases and enough information to execute them (preferably in an automatic way).
2. To use Web technology in a manner which reduces client setup to a minimum and to deploy the maximum testing through all the company.
3. To share testing resources to allow to clients access to expensive resources (due to development time or hardware/device availability).
4. In testing environments where a hardware device is also involved, to use simulation software in order to avoid lack of prototypes and human intervention (for example, to load paper in the printer), and to save money.

We think that the main design criteria of this automation environment, in order to overcome the previous disadvantages, must be:

- a) Zero setup and maintenance at the client side. The software in the client side must be multiplatform. This allows to spread the automation environment without cost increment. Moreover, clients can access testing resources without workload increment and so they have almost no barrier to begin doing testing.
- b) High control about test cases. Reuse testing knowledge is fundamental to avoid unnecessary expenses and to improve testing reuse. Therefore, a repository where test cases are grouped in accordance to different variables (functionality tested, target market...) must be the core of this environment. A query interface must facilitate client requests. Moreover, management of test case results is a central piece in order to improve testing efficiency and to improve development processes.
- c) A strategy to verify testing results in an automatic way. This strategy will be specific for each different type of project. In this paper we will explain the strategy adopted for the HP DesignJet printer's drivers¹.

¹ HP DesignJet printers, commonly called plotters, are large format printers able to print drawings up 54 inches wide. They support several graphic languages (HPGL/2, RTL, and Postscript) and for each of these languages a driver is developed. These printers are used by graphics artists, CAD users, GIS users, ...

In the remaining of this paper, firstly we will explain the general architecture proposing two alternatives, and then we will develop an example about Hewlett-Packard DesignJet printers and the current testing environment for its drivers.

3. System test server's architecture

The model we propose consists of three main components. Firstly, a repository where test cases are stored containing enough information in order to automatically execute them and to know what the test case proves and how the test case result can be verified. Secondly, an Internet-oriented infrastructure that allows to the users with a minimum of client-side setup to access and share of all testing resources with the consequent savings of cost and time. And lastly, the test environment to be deployed.

We propose two similar architectures: one based on a pure Java model, and another one based mainly in Microsoft Internet technologies. Both meet the requirements previously listed with the only difference of the Internet infrastructure implementation. Figures 1 and 2 summarize both approaches:

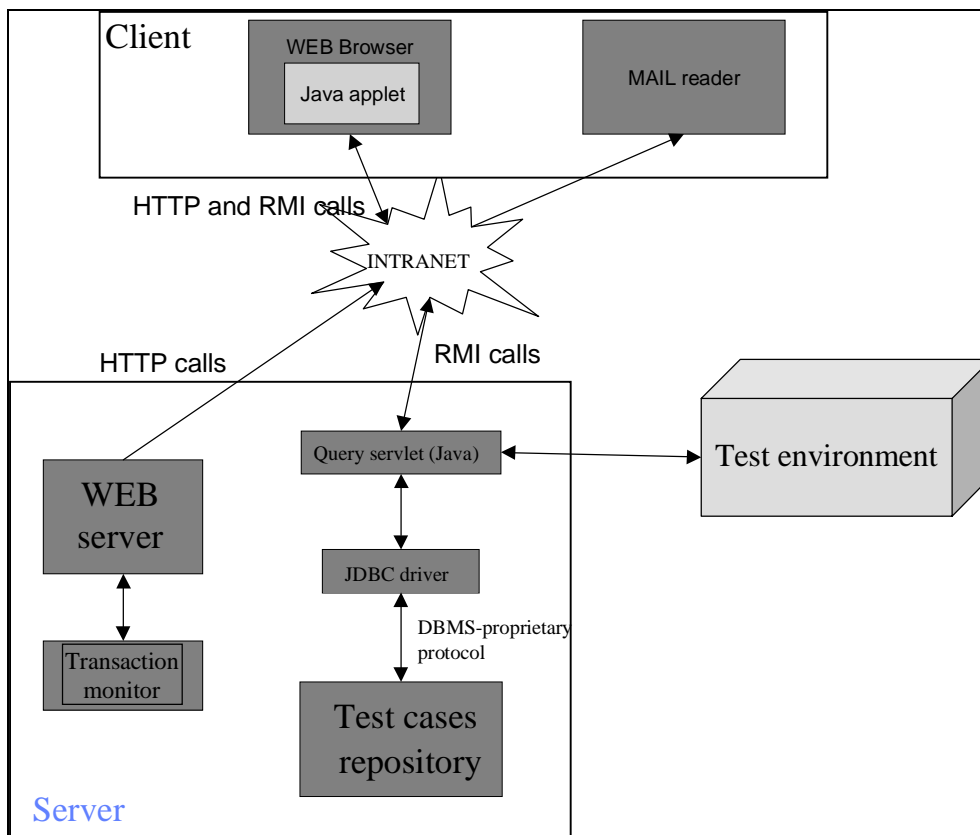


Figure 1: System architecture based on Java components

In these architecture we replicate the known three-tier model. So, the top tier (user interface) is the entire client side, the middle tier (business logic) is represented by

the test environment and the Java servlet, and the bottom tier (data storage) is composed by the database.

In the Java [Java98] alternative (figure 1) the client side just consists of a Java applet running in a Web browser. This applet will provide enough functionality to browse and query the test cases database, and to issue an *execution starting* command based on the user selection. The client side simplicity makes it easy to deploy the test technology beyond quality department. The client and server will communicate using HTTP and Java-to-Java Remote Method Invocation (RMI) [RMI98] protocols.

In the middle tier of the server there are several basic components:

- **The Web server** in order to serve HTTP requests, for example the first connection to the Java applet's page. Any Web server is a valid candidate.
- **The Transaction Processing (TP) monitor**: It manages all the tasks involved in a query and serves the query as an atomic transaction (all or nothing). Moreover it does load balancing, thread management, security, connection pooling. This is an optional component. But, performance and reliability will improve if a TP monitor is used.
- **The servlet**: written in Java, it manages multiple execution of test scripts with the (optional) support of a TP monitor. Client uses the RMI protocol to communicate with the servlet. The main servlet tasks are:
 - a) Once the user have built the final query, the servlet has to retrieve additional data from the repository in order to make the automatic execution of the test cases selected feasible.
 - b) For each test case, the servlet has to pass this information to the client on the machine where execution will be done (W95 PC, or WNT PC, and so on). This information may be passed using any remote procedure call (RMI, RPC,) or a simple interchange of files, depending on the method implemented in the different clients.
 - c) During the entire process, the servlet will log the query's execution status in the database and will inform the users about failures.
- **The test environment** (test resources) to be deployed via the Web. In each machine of the test environment executing test cases, there will be a piece of software in charge of communicating with the servlet on the server. Main tasks of this software will be:
 - a) Launching and configuring the components needed to execute each test case.
 - b) Recovering when an execution error occurs and communicating it to the server.
 - c) Logging useful data about the execution of the test case and communicating

them to the server. For example, performance data.

- **Java Database Connectivity (JDBC)** [JDBC98] plays an important role. JDBC, a Java-based interface to SQL-based database engines, provides a consistent interface for communicating with a database and for accessing database metadata. Individual vendors provide specific drivers to their particular database management system. JDBC is important to allow database access from a Java middle tier abstracting database implementation specifics.

In the top tier we have:

- **Client Java applet**: downloaded from the server embedded in a HTML page. This applet shows an user interface, allowing an user to do queries and issues commands to the system test server. Users should have at least two possibilities: to make a query using keywords or to select test cases directly.
- **E-mail reader**: used to receive notifications about the end of tasks or the existence of errors.

And in the bottom tier:

- **Test cases repository**: this repository contains at least the following information:
 - a) For each test case, a set of keywords defining what the test case proves. Users can build queries using the complete set of keywords.
 - b) For each test case, information about how to verify the result.
 - c) For each test case, information about where to find components involved in its execution.
 - d) Data about features of the *software to be tested* in order to allow flexibility in test cases.
 - e) Data about server capabilities (which are the resources of the test environments, where they are...).
 - f) For each test case executed, data about its execution. For example, number of errors found, time spent in its execution, number of times it has been executed, ...

The second alternative is based on Microsoft-world components. Both alternatives have the same philosophy but different implementations. They use different components for the Internet infrastructure.

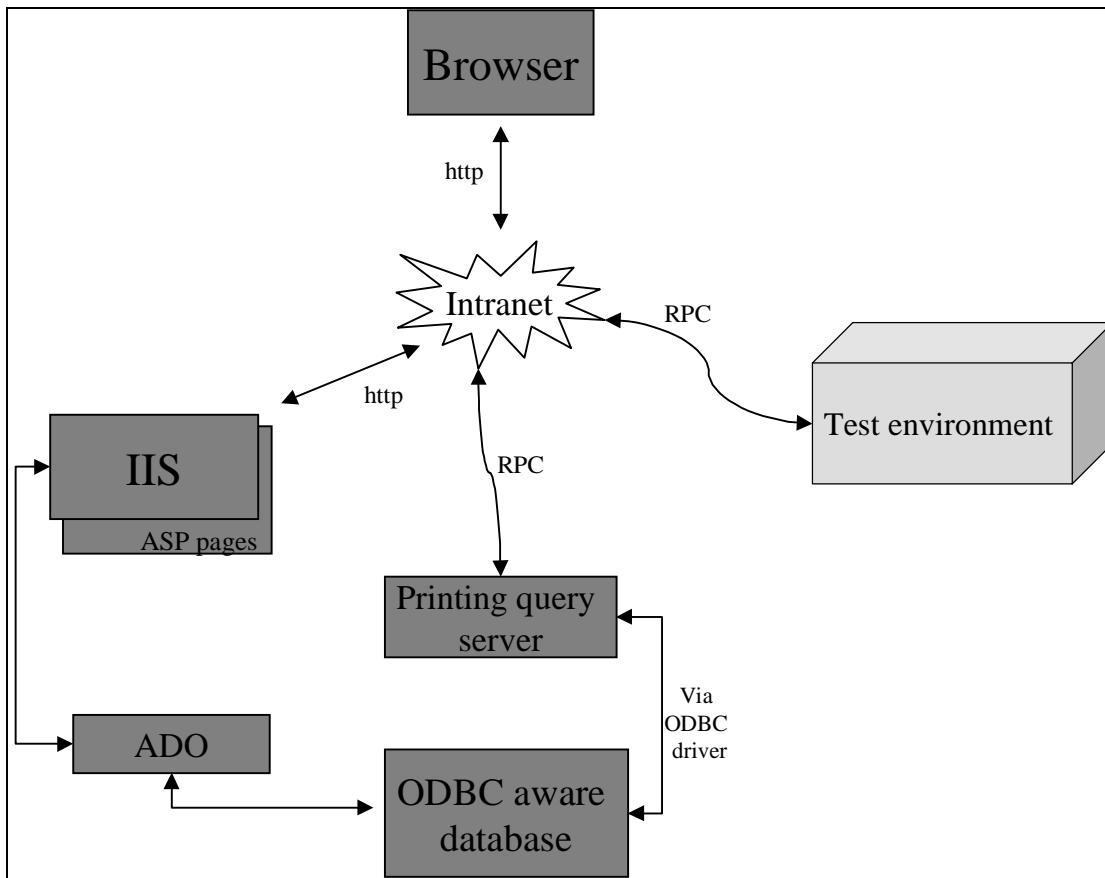


Figure 2: System architecture based on Microsoft components

The differences between the Microsoft-based model (figure 2) and the Java-based model are the following:

- Microsoft-based model uses Microsoft's ASP pages (Active Server Pages) [Francis98]. Active Server Pages allow you an easy means of querying and updating a database from a Web page. ASP pages allow you to combine scripting and HTML on Microsoft's Internet Information Server. ASP embedded scripting code may be written both in JavaScript and VBScript. ASP pages are independent of the type of browser that will be used to access these pages because they are executed on the server; the client receives their output as HTML. Additionally, server components can be easily added to extend the Internet/intranet application.

In figure 3 there is an example of ASP scripting. The code is inserted along with the HTML tags. The code is closed between '<%>' and '<%' symbols. The page must have the '.asp' extension.

In the example we will use a simple one-table database, with information about test cases. The test case-id is the primary key and it contains the test case information like name, operating system, what proves, and so on. We are going to create a SQL query that returns the test cases on Windows 95:

```
' We create command and record set objects
Set SQLCommand = Server.CreateObject("ADODB.Command")
Set TestCasesSet = Server.CreateObject("ADODB.RecordSet")

' Set the ActiveConnection property of command object to the ODBC source we will connect with
SQLCommand.ActiveConnection = "ODBC_source"

' Build the SQL query
SQLquery = "SELECT * FROM TestCasesTable WHERE OS='Windows 95'"

' We assign the SQL query to the CommandText property of Command object
SQLCommand.CommandText = SQLquery
SQLCommand.CommandType = 1

' Execute the command (query), and set the record set object to the result
Set TestCasesSet = SQLCommand.Execute

' Release the resources for command object
Set TestCasesSet.ActiveConnection = Nothing

' Use query result to populate your HTML output stream
Do While NOT TestCasesSet.EOF
....
....
TestCasesSet.MoveNext
Loop
```

Figure 3: ASP example

- Microsoft-based model uses RPC instead of RMI. Moreover, RPC is just used to communicate between the server and the test environment. There is no RPC use in the communication between the browser and the server, this communication is simplified and done via HTML.
- It uses an HTML-based User Interface.

Although they are similar solutions, to use Java instead of ASP or vice versa, has a lot of implications, above all in terms of flexibility and simplicity. Advantages and disadvantages of both approaches are summarized in the following table:

	Advantages	Disadvantages
ASP-based model	<ul style="list-style-type: none"> ▪ Visual Basic is easier to learn than Java/RMI. ▪ Browser must no have special capabilities 	<ul style="list-style-type: none"> ▪ The user interface is constrained to the HTML features. ▪ The Microsoft-SQL is not standard. ▪ The server must be the Microsoft's Internet Information Server. Therefore it's a proprietary solution.
Java-based model	<ul style="list-style-type: none"> ▪ High flexibility in the user interface layout . ▪ More flexibility in the server side. 	<ul style="list-style-type: none"> ▪ It's necessary to deal with the complexity of Java and RMI. ▪ RMI only works in the most recent browsers (browsers that support Java Platform 1.1). ▪ Browser must have enabled the Java setting (security issues)

4. An implementation of this approach: HP Large Format Printers testing

Our lab is dedicated to investigating and implementing new Large Format Printing solutions. So we mainly develop printers and software related to it (drivers, status monitors, printing tools,...). The Software Quality team is in charge of providing quality services to developers, including testing services. System testing is one of our most valuable test strategies, because it covers all the components we develop (device, drivers, ...), and the applications used by our users. Therefore, System testing ensures quality in the entire solution.

In order to do System Test we have a very complex System Test environment, composed of:

- The device(s), printer(s), under test.
- The drivers that are necessary for plotting to the device(s) from different operating systems and commercial applications.
- Main applications used by our users (divided in several markets).
- Several operating systems (Windows 3.1/95/NT, MS-DOS, UNIX and MAC OS)

Figure 4 shows the typical testing flowchart.

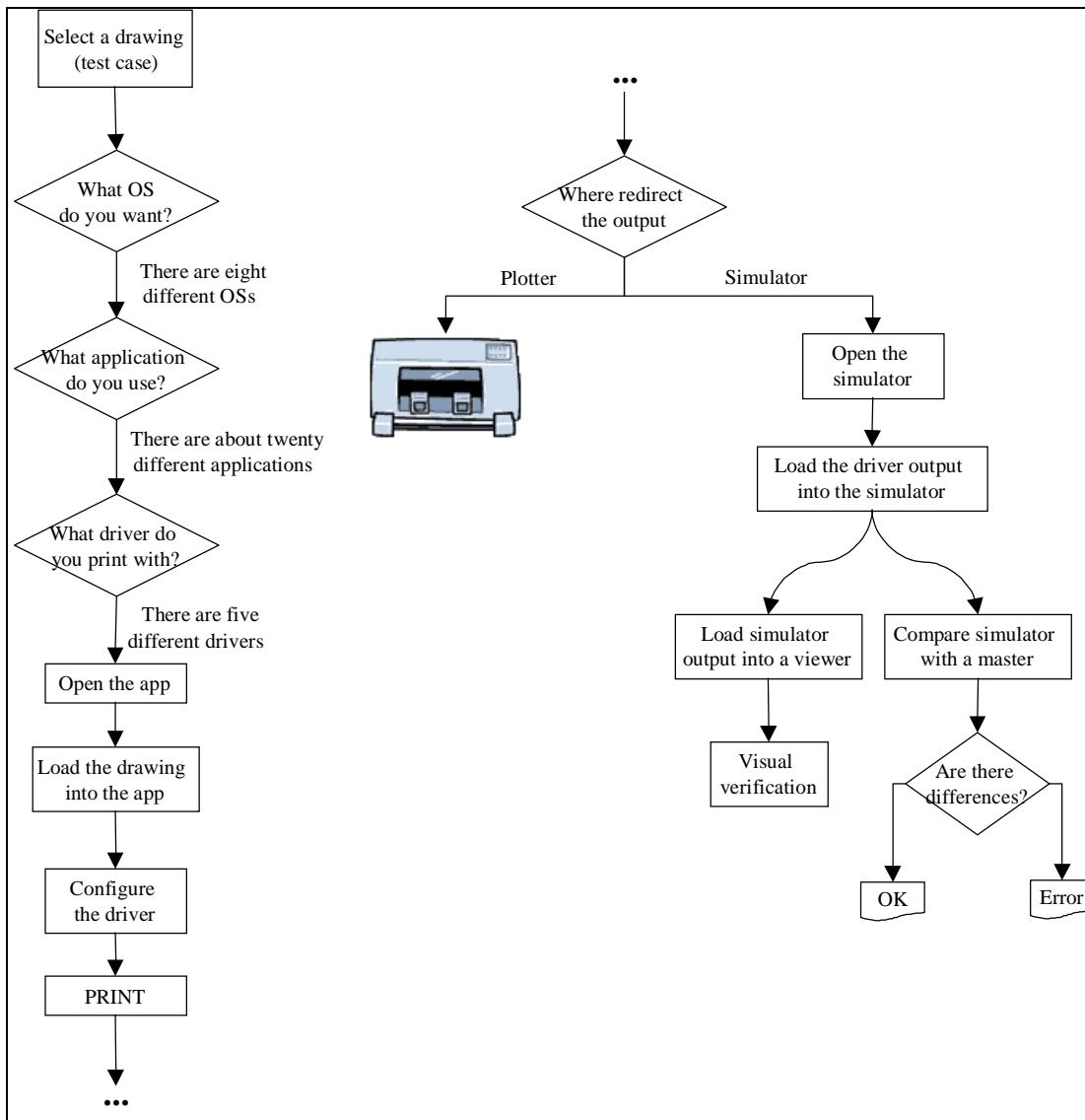


Figure 4: Our testing flowchart

Our system test cases usually consist of a drawing that is printed using one of our drivers. In order to print the drawing the driver must be configured with significant values. A commercial application is used to load the drawing and print it using the driver. We have different drivers (System Printer, Postscript, ...) running on different platforms (Windows 3.1, Windows NT, System 7 and 8, ...).

In order to reduce testing complexity and cost we may redirect the driver output to a printer simulator. The printer simulator consists of the true printer firmware compiled on VxSim/HP-UX instead of on the printer motherboard. This has several advantages: we can simulate a printer when printer prototypes are not available and we reduce testing cost significantly due to the fact that the simulator is cheaper and accessible to everyone in the lab. The output of the simulator is previewed using a tool called Vpaper that generates a bitmap as printed in paper from the simulator's output format.

The main drawback of such complexity is that our quality engineers use various testing automation environments. These environments are faced with the issues of complexity, cost (hardware prototype, several market applications involved) and difficult client setup. Moreover, the existence of various testing environments made that the quality engineer's workload didn't allow to maintain the high amount of test cases under control. These drawbacks caused development engineers to be reluctant to do system testing and so this kind of testing is done mainly by quality engineers and at the last stages of development (with the known cost and inefficiency).

The solution has been implementing the approach presented in this paper. The software quality team focuses on system test deployment, sharing of valuable testing resources, low maintenance and setup and understanding what our test cases prove. The pieces of our anterior test environment have been integrated in the new one in the following way:

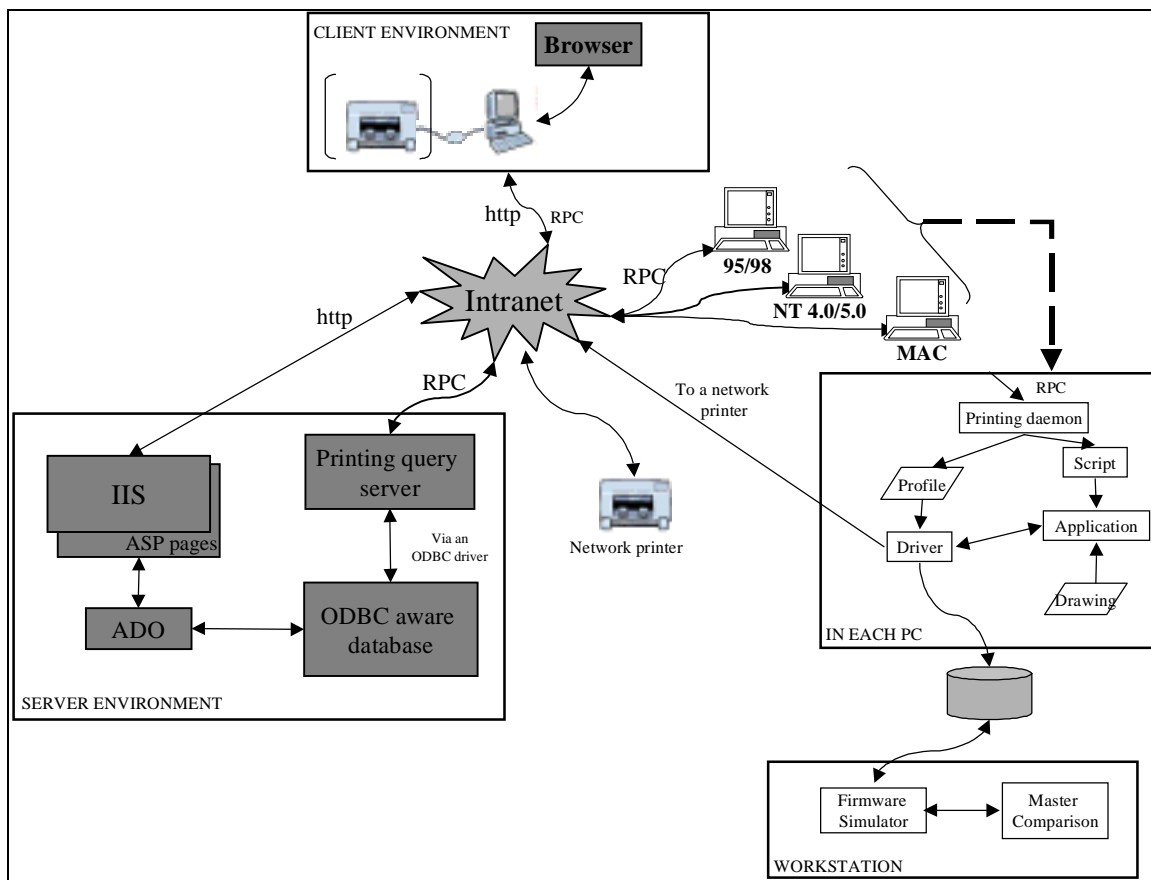


Figure 5: Our test environment deployed via WEB

Our test cases are drawings to be plotted in the printer using a specific driver under any operating system supported, using a specific application. Each drawing tests one or several features of the entire printing path. For each test case stored in the database we gather information about:

- What the drawing tests? For example, correct margins management, line width/line length accuracy, color matching, fill pattern, text, ...
- Test case characteristics. For example, if the test case includes color, how many dots per inch the drawing has, complexity (high | mid | low), applications that can be used to print it (PhotoShop, Corel, AutoCAD, ...), ...
- Target market of the type of drawing (for example, Print Service Providers, Retail, GIS, Sign Shops, Architectural, ...)

Each test case has an associated script. This script executes the test case using both informations in the database as client applet settings (for example, print quality, print area, paper type...)

5. Use model

In this section we detail the use model of the System Test server via the Web, using the Java-based model.

Steps involved in the execution of a query are the following:

1. A user accesses to the home page of the testing environment using his WEB browser. The client applet is executed in the client. The applet shows a query interface built from the database information: drawing catalog, available drivers and applications, and so on.
2. The user queries the server: for example “I want to test color adjustment using the Postscript driver in Windows 95”, or “I want to run a regression for GIS market”, or “I want to test such a drawing using the AutoCAD driver in AutoCAD R14 for Windows NT and in the DesignJet 2500CP”.
3. The servlet gets the query and creates a process in order to manage it. In this step Transaction Processing (TP) monitor services are required, in this way we overcome issues of robustness, reliability, and quality of service.
4. The servlet accesses the database and select test cases that meet query requirements. Then, it creates a data stream with query results and sends it back to the client.
5. The client’s applet shows the query results in a friendly way. Then, the user has the choice of executing all the test cases, selecting a subset or refining the query. When the user has selected the appropriate test cases he sends a printing command to the server.
6. The servlet gets the printing command and parses its content. It retrieves the test cases from the database and all the information needed to execute them. Then, it passes each test case to the correct testing PC. For example, if a test case must be

executed on Windows 95, the servlet passes information to a testing PC running a *test execution* daemon in this OS. (In this step, TP monitor services are also required).

7. The *test execution* daemon (replicated in each testing machine) does two tasks:
 - First, it builds a driver configuration file (profile) that later will be loaded by the driver, and
 - Second, it calls a script in order to execute the application, load a drawing in the application and issue a *printing command*. The printing command will launch the driver, and this will load the profile.

The driver output is sent to a networked printer, a printer simulator or stored on hard disk, depending of the user's choice. The printing daemon will notify the servlet of any error situation during the test case execution.

8. If the user decides to send the binary output to a networked printer, the servlet redirects the output to the corresponding printing queue and sends a notification to the user via e-mail.
9. If the user decides to store the binary output on hard disk, the servlet stores it in any of the hard disks managed by the environment. The user can later access the file generated.
10. If the user decided to simulate the binary output in any of the simulators available, the servlet communicates this proposal to a UNIX daemon that will be the owner of the simulation. The driver output and some additional information are also sent to the UNIX daemon. The latter calls the printer simulator and notifies the user about the location of the final output via e-mail.

6. Conclusions

We have presented two approaches (one based in Microsoft components and another one based in Java) in order to deploy a testing environment using Web technologies. Both approaches are similar, but we finally have implemented the Microsoft one because of its simplicity. The Java model has a high flexibility but at the expense of some additional complexity. In both approaches, a test case repository plays a main role. In this repository test cases are classified in accordance with testing criteria. So, we improve reuse.

We think that Web technologies can help improve testing performance as well as reduce testing cost. But to be successful is important to achieve zero setup and maintenance at the client side, to have a high control about test cases, and to have a strategy to verify testing results in an automatic way.

7. Bibliography

[Francis98] Brian Francis, Alex Fedorow, Richard Harrison, Dave Sussman, Rob Smith, Alex Homer, Shawn Murphy. *Professional Active Server Pages 2.0*. Wrox Press Inc. March 1998.

[Humphrey90] Humphrey, Watts S. *Managing the Software Process*. Addison-Wesley, 1990.

[Java98] Java home page at <http://www.javasoft.com>.

[Jones91] Jones, C. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw Hill.

[Myers76] Myers, G. J. *Software Reliability, Principles and Practices*, New York: Wiley, 1976.

[ODBC98] Java home page at <http://www.javasoft.com>.

[RMI98] Java home page at <http://www.javasoft.com>.

Automated Test Data Generation to Solve the Y2K Problem*

István Forgács and Ákos Hajnal

Computer and Automation Institute
Hungarian Academy of Sciences

**Research was supported in part by AKP grant 97-129 2,1/20*

Outline

- ⇒ Introduction
- ⇒ Y2k fault detection method
- ⇒ Y2k criterion
- ⇒ Test data generation
- ⇒ Example
- ⇒ Related work
- ⇒ Concluding remarks

Introduction

- ⇒ Nobody knows what will happen on January 1, 2000
 - ⇒ What is the Year 2000 (Y2k) problem?
 - ⇒ Many computer programs use only two digits to record year values.
 - ⇒ Both 1900 and 2000 are represented as "00" ⇔ defect
- Example: Mr. Smith was born in 1960, then $2000 - 1960 = 40$
However, $0 - 60 = -60$, i.e., this person is just -60 years old according to the computer!
- ⇒ Problem: we have to fix both the program and the database that contain two digits year-valued data
 - ⇒ The present ad-hoc methods are not reliable and they cannot be used to test the fixed program

Introduction

- ⇒ The technical part of the Y2k problem includes
 - ⇒ (1) testing for the Y2k faults,
 - ⇒ (2) the program and database fixing process,
 - ⇒ (3) post-renovation testing.
- ⇒ Our method addresses (1) and (3), so that Y2k bugs can be found automatically
- ⇒ We do not need to know the value of any output, or how to select input data
- ⇒ Key idea: We compare output (or branch) functions for six different years (1997-2002) ⇔ automated validation
- ⇒ We apply a very reliable criterion ⇔ safety
- ⇒ We reduce programs by using slicing ⇔ efficiency (wrt cost and time)

Y2k fault detection method: An example

```
read(workers_name)
seek(workers_file, workers_name)
read(workers_record)
...
age = CurrentDate.year - WorkersRecord.birthday.year
if age > 60 then
    TaxPercentage = 0
else
    TaxPercentage = 20
...
```

Y2k fault detection method (previous methods)

- ⇒ Let us test the program for some (not too old) employees for years 1998 and 2000
 - ⇒ All the people pay tax in every case
 - ⇒ Since this is obvious, any tester without inspecting the code thoroughly would believe that the testing process was adequate
 - ⇒ Though white box testing may reveal this bug, a deep knowledge is required
 - ⇒ Assume automated test data generation
 - ⇒ We have to validate the results manually
 - ⇒ Reliable criterion requires the generation of numerous test cases
- ↓
- ⇒ The method is very time consuming

Y2k fault detection method

- ⇒ Input: employee was born in 1959
- ⇒ test cases: six different years,
i.e., CurrentDate.year = 1997, 1998, 1999, 2000, 2001 and 2002
- ⇒ branch function:
 $f = \text{age} - 60 = \text{CurrentDate.year} - \text{WorkersRecord.birth.year} - 60$
- ⇒ Assuming two-digits year-valued data we obtain the branch functions:
-22, -21, -20, -119, -118, -117
- ⇒ There is a big jump considering 1999 and 2000
- ⇒ For two consecutive years excluding (1999, 2000) the difference of the branch functions is $f(1998) - f(1997) = -21 - (-22) = 1$
- ⇒ However, the difference of the branch functions for 1999 and 2000 is $f(2000) - f(1999) = -119 - (-20) = -99$
- ⇒ Consequence: the Y2k fault has been recognized for any employee even if the output is correct for all dates!

Y2k fault detection method

- 1 Assume that we are given an executable program path P and a program input x for which P is traversed
 - 2 Select an instruction I (can be a predicate or an output) that is influenced by a Y2k-related input variable
 - 3 Generate six different program inputs by setting year to the values 1997, 1998, 1999, 2000, 2001, 2002, keeping all the other input variables unchanged
 - 4 Compute the differences:
 $d_1 = |f(1998) - f(1997)|,$
 $d_2 = |f(1999) - f(1998)|,$ $d_3 = |f(2000) - f(1999)|,$
 $d_4 = |f(2001) - f(2000)|,$ $d_5 = |f(2002) - f(2001)|$
 - 5 Y2K fault occurs along P, if any d_3 is larger than any other d_i
- ⇒ The method works well surely for linear case, however, in any case it is safe

Y2k criterion

- ⇒ Y2k criterion requires the selection of a set of program inputs that reveals the Y2k faults (if any) of a given program with high probability
- ⇒ It is also an adequacy criterion that efficiently tests the program after the fixing of Y2k bugs
- ⇒ Information we need are:
 - ⇒ The names of date-related input variables (*Y2k variables*)
 - ⇒ The content of the date (year, year and month, year, month and day)
- ⇒ Which program paths should be covered?
 - ⇒ If we cover only each edge, the method is not safe
 - ⇒ If we cover all (cycle-free) paths, the method is very expensive
 - ⇒ Solution: SLICING
 - ⇒ A slice contains all the statements that might affect the set of variables used at a program point I

Y2k criterion

- The forward slice consists of all the statements and predicates that might be affected by a set of variables defined at a program instruction I
- Consider each Y2k variable v one-by-one.
- First, we determine the forward slice for v at an instruction I, where v is read from keyboard, from a database, etc.
- Select output instructions and determine the (backward) slice for each output o
- We obtain the statements that may affect o
- We intersect the statements that are in both slices; the new subprogram is called a *double-slice*
- Any Y2k criterion has to select program path related to the double-slice only ⇔ The number of tests is significantly reduced

Y2k criterion

- ⇒ Select a Y2k variable v and an output O
 - ⇒ generate each cycle-free executable program path that reaches the output corresponding to the double-slice
 - ⇒ generate executable program paths so that each loop be iterated once and twice
 - ⇒ apply Y2k fault detection method for each selected path for the output function

↓

the Y2k criterion for (v, O) has been satisfied

- ⇒ Extend the method to apply Y2k fault detection method for the predicates that are along the prefix path that reaches O
- ⇒ Select each output for the selected Y2k variable
- ⇒ Repeat the above method for every Y2k variable ⇔

The Y2k criterion is satisfied

Test data generation

- ⇒ Our algorithm
 - ↳ generates a set of test data to cover each necessary program path in the double-slice
 - ↳ uses an improved function minimization technique to alter the flow of control at a selected conditional statement
 - ↳ checks these paths with respect to Y2k faults
- ⇒ The *branch function expression* of a simple predicate $A \text{ op } B$ (where $op \in \{<, \leq, >, \geq, =, \neq\}$) is $F = A - B$, which is a function of the current program input

Test data generation

- ⇒ Assume that F is positive
- ⇒ To alter the flow of control we have to find a program input for which the branch function value is negative (or zero)
- ⇒ A two-phase method is applied to decrease the branch function value:
 - ↳ *exploratory search*
 - ↳ *minimization procedure*
- ⇒ Exploratory search
 - ◆ We modify (increase/decrease) input variables (keeping all other variables unchanged) by a small amount until the branch function decreases

Test data generation

- ⇒ Minimization procedure
 - ◆ The current program input is modified repeatedly along the same direction that is selected by the exploratory search
 - ◆ The amount of the modification is doubled until the branch function improves or the prefix path changes
 - ◆ The amount of the modification is halved until a local minimum has been found or F becomes negative
 - ◆ If the latter case occurs, then the algorithm successfully modifies the flow of control at the selected predicate
 - ◆ If F remains positive, then we select another variable and the whole method including exploratory search and minimization is repeated

Test data generation

- ⇒ The coverage algorithm
 - ◆ starts with an arbitrarily selected program input
 - ◆ explores all program paths systematically by applying the test data generation method repeatedly for the selected predicates
 - ◆ executes each loop zero times, once, and twice
- ⇒ The paths are validated with respect to the Y2k problem
- ⇒ If the prefix path changes for a selected series of test data, a *path correction method* is applied

Example

```
read(current_year)
read(job_begin_year)
read(salary_class)
a  if current_year < 0 then
1   exit ('error')
b  if job_begin_year < 0 then
2   exit ('error')
3   salary = 80000
c  if current_year - job_begin_year >= 10 then
4   salary = salary * salary_class
   else
5   salary = salary + 1000 *
      (current_year - job_begin_year)
6  write (salary)
```


Example

- ⇒ Let us select an arbitrary initial program input I_1 , such as
`current_year=1997, job_begin_year=1982,`
`salary_class=1.0` ($I_1=[1997,1982,1.0]$)
- ⇒ Generate the double-slice of the code for input variable `current_year` and output variable `salary`
 - ↳ forward-slicing filters out predicate `b`, statement 2 and 3
 - ↳ backward-slicing filters out predicate `a`, and statement 1
- ⇒ The test data generation algorithm automatically finds the program input $I_2=[1997,1989,1.0]$ that covers the other possible path
- ⇒ Generate and execute the series of program inputs
 - ↳ $I_1 \Rightarrow$ [1997,1982,1.0], [1998,1982,1.0], [1999,1982,1.0], [2000,1982,1.0], [2001,1982,1.0], [2002,1982,1.0]
 - ↳ $I_2 \Rightarrow$ [1997,1989,1.0], [1998,1989,1.0], [1999,1989,1.0], [2000,1989,1.0], [2001,1989,1.0], [2002,1989,1.0]

Example

- ⇒ Since the prefix path changes for the I_2 -series, the path correction method is applied resulting $I_2=[1997,1996,1.0]$
- ⇒ Branch function values of predicate `c` for the I_1 -series

<i>correct</i> ⇒	<table style="border-collapse: collapse; margin: auto;"> <thead> <tr><th>date</th><th><i>F_c</i></th></tr> </thead> <tbody> <tr><td>1997</td><td>5</td></tr> <tr><td>1998</td><td>6</td></tr> <tr><td>1999</td><td>7</td></tr> <tr><td>2000</td><td>8</td></tr> <tr><td>2001</td><td>9</td></tr> <tr><td>2002</td><td>10</td></tr> </tbody> </table>	date	<i>F_c</i>	1997	5	1998	6	1999	7	2000	8	2001	9	2002	10	<i>faulty</i> ⇒	<table style="border-collapse: collapse; margin: auto;"> <thead> <tr><th>date</th><th><i>F_c</i></th></tr> </thead> <tbody> <tr><td>1997</td><td>5</td></tr> <tr><td>1998</td><td>6</td></tr> <tr><td>1999</td><td>7</td></tr> <tr><td>2000</td><td>-92</td></tr> <tr><td>2001</td><td>-91</td></tr> <tr><td>2002</td><td>-90</td></tr> </tbody> </table>	date	<i>F_c</i>	1997	5	1998	6	1999	7	2000	-92	2001	-91	2002	-90
date	<i>F_c</i>																														
1997	5																														
1998	6																														
1999	7																														
2000	8																														
2001	9																														
2002	10																														
date	<i>F_c</i>																														
1997	5																														
1998	6																														
1999	7																														
2000	-92																														
2001	-91																														
2002	-90																														
<i>program</i>		<i>program</i>																													

- ⇒ Branch function- and output values for the I_2 -series

<i>correct</i> ⇒	<table style="border-collapse: collapse; margin: auto;"> <thead> <tr><th>date</th><th><i>F_c</i></th><th>salary</th></tr> </thead> <tbody> <tr><td>1997</td><td>-9</td><td>81000</td></tr> <tr><td>1998</td><td>-8</td><td>82000</td></tr> <tr><td>1999</td><td>-7</td><td>83000</td></tr> <tr><td>2000</td><td>-6</td><td>84000</td></tr> <tr><td>2001</td><td>-5</td><td>85000</td></tr> <tr><td>2002</td><td>-4</td><td>86000</td></tr> </tbody> </table>	date	<i>F_c</i>	salary	1997	-9	81000	1998	-8	82000	1999	-7	83000	2000	-6	84000	2001	-5	85000	2002	-4	86000	<i>faulty</i> ⇒	<table style="border-collapse: collapse; margin: auto;"> <thead> <tr><th>date</th><th><i>F_c</i></th><th>salary</th></tr> </thead> <tbody> <tr><td>1997</td><td>-9</td><td>81000</td></tr> <tr><td>1998</td><td>-8</td><td>82000</td></tr> <tr><td>1999</td><td>-7</td><td>83000</td></tr> <tr><td>2000</td><td>-106</td><td>-16000</td></tr> <tr><td>2001</td><td>-105</td><td>-15000</td></tr> <tr><td>2002</td><td>-104</td><td>-14000</td></tr> </tbody> </table>	date	<i>F_c</i>	salary	1997	-9	81000	1998	-8	82000	1999	-7	83000	2000	-106	-16000	2001	-105	-15000	2002	-104	-14000
date	<i>F_c</i>	salary																																											
1997	-9	81000																																											
1998	-8	82000																																											
1999	-7	83000																																											
2000	-6	84000																																											
2001	-5	85000																																											
2002	-4	86000																																											
date	<i>F_c</i>	salary																																											
1997	-9	81000																																											
1998	-8	82000																																											
1999	-7	83000																																											
2000	-106	-16000																																											
2001	-105	-15000																																											
2002	-104	-14000																																											
<i>program</i>		<i>program</i>																																											

Related work

- ⇒ Commercial products try to find the location where dates are employed
 - ↳ date-manipulation sites can be identified by the places where there is a call to the operating system
 - ↳ automated string-searching tools identify some patterns, for example, `"*date"`, `"*yy*"`
 - ↳ these methods cannot be applied for post-renovation testing, since these methods do not execute programs at all
- ⇒ Path profiling method (Reps et al. 1997)
 - ↳ A path profiler instruments programs so that the different executed program paths can be recognized
 - ↳ After several runs of the code a path spectrum can be obtained and displayed so that the frequency of different executed paths is determined
 - ↳ If the path spectra for pre-2000 and post-2000 values are different, a Y2k fault has been detected
 - ↳ The method is not reliable for some Y2k fault

Conclusion

- ⇒ This Y2k fault detection method
 - ↳ is fully automated, test cases are generated, human validation is not necessary,
 - ↳ requires quite a few initial information,
 - ↳ is reliable, it requires the satisfaction of a very strong criterion,
 - ↳ is fast, since by applying double-slices only the necessary code has to be investigated,
 - ↳ can be used to reveal Y2k bugs,
 - ↳ can be applied as an adequacy method to check whether a program is Y2k bug-free.
- ⇒ Implementation
 - ↳ automated test data generation has been implemented in part,
 - ↳ slicing has been implemented for intraprocedural case

Automated Test Data Generation to Solve the Y2K Problem *

István Forgács and Ákos Hajnal
Computer and Automation Institute
Hungarian Academy of Sciences
1111 Kende u. 13-17.
Budapest, Hungary
{forgacs, ahajnal}@sztaki.hu

September 15, 1998

Abstract

This paper describes a new technique to solve the most critical elements of the "Year 2000 Problem". Y2k problem has both management and technical aspects. The technical part includes (1) testing for the Y2k faults, (2) the program and database fixing process, and (3) the post-renovation testing. Our method addresses (1) and (3), so that Y2k bugs can be found automatically. This means that we do not need to know the value of any output, or how to select input data.

Our method is not only applicable, but fast and reliable as well. To do this our method consists of three main parts. The key idea of our approach is to compare the branch (or output) functions for different input years. This method reveals the Y2k bug even if the output of the program was correct, i.e., the tester would not observe any failure. To satisfy reliability requirements we apply a very strong testing criterion. To speed up testing we apply slicing, by which only a small subset of the program has to be adequately analyzed.

1 INTRODUCTION

Though a tremendous effort has been made to avoid the "bomb of millennium", nobody knows what will happen on January 1, 2000. We cannot be sure that we will have electricity, we can fly to other countries or we can get our money from an ATM.

The Year 2000 (Y2k) problem is the only software engineering problem that is widely known for everybody from young children to pensioners who have never switched on a computer. Surfing the internet there are thousands of sites containing information about the Y2k problem. Lots of consulting companies offer solution to this problem, and numerous tools are available to reveal Y2k bugs. In addition, thousands of programmers are fixing programs suffering

*Research was supported in part by AKP grant 97-129 2,1/20.

from the millennium bug. Why we cannot be sure despite these huge effort that programs will work properly after the first day of year 2000?

The cause is that these solutions are not reliable enough, since they based on ad hoc methods and they require significant human effort that always involves the probability of making errors. To our knowledge there is only one scientific solution to this problem [9], however, as we will see, this method is not always reliable either.

This paper describes a fully automated Y2k-fault detection method that is almost 100% reliable. In addition we hope that this method is applicable and requires manageable time, effort and cost. Before we informally introduce our method, we shortly describe the Y2k problem though it is well-known for everyone.

Many computer programs use only two digits to record year-valued data. In this case there is no difference between 1900 and 2000 since both dates are represented as "00". This false equivalence may cause defect during computation over year-valued data. For example, if we compute the (approximate) age of somebody who was born in 1970, then we should obtain 30 in 2000 by making the operation $2000 - 1970$. However, this operation in a 2-digit year-valued data representation would result in $00 - 70 = -70$, i.e., this person is just -70 years old! The problem is that we have to fix not only the program to correctly compute ages, but we have to recover the database that contains 2-digit year-valued data as well.

However, after changing both the program and the database we have to retest the whole program again. This is sometimes very difficult if the original programmers, system developers and testers are not available. Modified program and database may cause even more serious faults. Therefore, it is very important to change the code and the database if it is really necessary, i.e., if the Y2k bug is present.

We have introduced a new method that requires very few information and yet, its reliability is almost 100%. In addition our method is fully automated and requires as few testing effort as possible. Our method involves three different techniques. We compute output and branch functions for pre-2000 and post-2000 years. The results are evaluated and Y2k bugs are identified with a high probability. We apply a very strong criterion called *Y2k criterion* (see Section 4). The program paths that cover the required criterion are generated automatically by applying a fast test data generation method introduced in [3] implemented for Y2k criterion. Finally, we apply different slicing methods to reduce the program to be analyzed, and thus the number of paths to be covered. We are sure the code that manipulates year-valued variables is a small part of the entire program. In this expected case the slice is relatively small and the reduced all-paths criterion can be easily satisfied.

Our method is applicable both for Y2k-bug detection and testing the fixed program after the Y2k faults have been removed. However, fixing the code and the database is beyond the scope of our paper. Since the method is automated, there is no need for expert testers or programmers. The necessary information can be easily obtained, therefore the checking process is very short, and can be repeatedly applied more times after Y2k-fault reparation.

The remainder of the paper is organized as follows. Section 2 provides the necessary background. Section 3 describes how to address the Y2k problem for a given program point. In Section 4 we introduce Y2k testing criterion. Section 5

describes our test data generation method to satisfy the above criterion. Section 6 discusses related work, while in Section 7 we summarize our results.

2 BACKGROUND

A program structure can be represented by a directed graph called (*control flow graph*) $G = (N, E, s, e)$, where N is a set of nodes, E is a set of edges and s, e are unique entry and exit nodes. Nodes represent program statements and edges represent possible flow of control between nodes. Nodes that correspond to an if-then-else, or while statements are referred to as *predicate nodes*, or *predicate* for short. A path P from n_j to n_k in G is a sequence of nodes $P = \langle n_j, n_{j+1}, \dots, n_k \rangle$, where each adjacent pair (n_i, n_{i+1}) is an edge in E for $j \leq i < k - 1$. A *prefix path* is a path from s , a *postfix path* is a path to e , and a *complete path* is a path from s to e .

A (*program*) *slice* consists of all the statements and predicates that might affect the variables in a set V at a program instruction I [10]. The pair $C = (V, I)$ is called a *slicing criterion*. A slice is a subset of the program code. The term "affect" means that I is (transitively) dependent on a statement included in the slice. There are two types of (direct) dependences: (1) *data dependence* and (2) *control dependence*. There is a data dependence from statement n to statement m if m uses the value of a variable x , that is defined (assigned) in n and there is a path in the control-flow graph of the program from n to m along which x is never defined. There is a control dependence from a predicate p to a statement n if the execution of n directly depends on the evaluation of predicate p .

The *forward slice* consists of all the statements and predicates that might be affected by a set of variables V defined at a program instruction I . The forward slicing criterion is also a pair $C_f = (V, I)$. Note that usually V contains only one element, since only one variable is defined at a statement in most cases. The forward slice also contains a subset of the program. The forward slice determination starts from I and we include transitively all the statements that are either data or control dependent on I .

The *branch function expression*¹ (or *branch function*, for short) F associated to a *simple predicate* expression $(A \text{ op } B)$ (where A and B are arithmetic expressions, and op is a relational operator) is $F = (A - B)$.

A *test data adequacy criterion* is a relation $C \subseteq M_S \times S_S \times T$, where M_S is a set of modules, S_S is a set of specifications related to M_S , and T is a test suite. Such a relation provides a stopping rule for testing, i.e., a test suite T is deemed to be adequate, or sufficient, as soon as it fulfills C . We will also refer to C simply as a *testing criterion* or a *criterion*.

Consider a set of program paths P . P is adequate for the *all-path* criterion if every executable program path is included in P . The *branch testing* criterion is satisfied, if every executable outcome of any predicate has been executed during the test.

¹The term "branch function" is introduced in [7] first. Though its definition used in this paper and the original definition slightly differ, because of the same functionality we kept the notation.

3 Y2K FAULT detection method

In this section we present a method by which Y2k faults can be recognized even if the output is correct, i.e., the fault does not cause any failure. First, consider traditional black box testing approach by applying the small program fragment below.

```
read(workers_name)
seek(workers_file, workers_name)
read(workers_record)
...
age = CurrentDate.year - WorkersRecord.birthdate.year
if age > 60
then
    TaxPercentage = 0
else
    TaxPercentage = 20
...

```

Testing the program for some not too old employees and for inputs `CurrentDate.year = 1999` and `CurrentDate.year = 2000`, the tester observes that all people pay tax in every case. This seems to be quite obvious, and any tester without inspecting the code thoroughly would believe that the testing process was adequate.

Now assume that we apply a white box testing method, such as the branch testing criterion. In our very simple case the tester will find the fault after he/she fails to find input data to cover the *then* branch. After a careful investigation of both the program and the specification, our tester recognizes that if variable *age* were greater than 60, then the *else* branch should be followed. Then he/she tries to select an older employee. Finally, the tester realizes that an erroneous branch has been followed. This usually requires two people, one is expert in programming/testing, the other has a deep knowledge of the content of the database. However, this program is very simple. In practice it can occur that the tester can find inputs covering all the branches without any failure detection, though the Y2k bug is present.

Even the simplest criterion is difficult and time consuming to satisfy manually. Our goal is to automatically detect Y2k failures. We show that traditional methods cannot be applicable for full automation. Though there are methods that automatically select input data, the validation of the result wrt these data necessitates human interaction. For each automatically generated test case we have to know and check the corresponding output values. This is very time consuming, and boring. An efficient criterion may require more thousands tests. An independent tester does not know the related output values, while the user usually has no experience in testing. On the other hand, even if we have an efficient testing criterion, a fault may remain undetected.

Fortunately, there is another method that can reveal Y2k fault entirely automatically. We do not need to know any result, and we do not need to cover even both branches of a given predicate. Our approach is based on the fact that the value of the branch (or the output) function for input date 2000 is significantly different from the corresponding value for 1999.

Assume that we test our example by selecting an employee who was born in 1959 for six different dates: 1997, 1998, 1999, 2000, 2001 and 2002. Assume, in addition, that all dates have a 2-digit representation (59, 97, 98, 99, 00, 01 and 02). Since the branch function is $f = age - 60 = CurrentDate.year - WorkersRecord.birthday.year - 60$ we obtain the following series of results for the branch function: -22, -21, -20, -119, -118, -117. Considering this series we can see that there is a large jump when the system date is modified from 1999 to 2000. This jump is even more recognizable if we determine the difference of the branch functions for two consecutive years, e.g. $f(1998) - f(1997) = -21 - (-22) = 1$. We can see that we always obtain 1 for this difference except in the case of years 1999 and 2000 in which case we get $f(2000) - f(1999) = -119 - (-20) = -99$.

We can see that by applying the approach above **the Y2k fault has been recognized for any employee** even if the output is correct for all dates. Therefore, we do not need to know anything about the database or the specification of the program.

Now we can give the formal description of our *Y2k failure detection method*. Assume that we are given an executable program path P_i and a program input \mathbf{x} for which P_i is traversed. First, we select an instruction I (can be a predicate or an output) that is influenced by a Y2k-related input variable x_k , i.e., the modification of x_k may change the value of the output or branch function f of I . Then, we generate six different program inputs by setting x_k to the values 1997, 1998, 1999, 2000, 2001, 2002, keeping all the other input variables for \mathbf{x} unchanged. We execute the program for these six program inputs, store the values of f , and compute the differences: $d_1 = |f(1998) - f(1997)|$, $d_2 = |f(1999) - f(1998)|$, $d_3 = |f(2000) - f(1999)|$, $d_4 = |f(2001) - f(2000)|$, $d_5 = |f(2002) - f(2001)|$.

We assume that a Y2K fault occurs along P_i , if d_3 is larger than any other d_i . The reason of this assumption is safety. If function f is linear that occurs in most cases, then our method works well. Note that we mean that the correct function is linear but the realized is not just because of the Y2k fault. Even if f is non-linear (that is a nonsense since quadratic years can be used for nothing), the method probably reveals Y2k faults. For the same reason, we can expect that *false positive* occurs only in very few cases (or never). This “false alarm” (when the method erroneously finds an Y2k bug) can be identified easily.

4 Y2k CRITERION

In this section we introduce a criterion called *Y2k criterion* that requires the selection of a set of program inputs that reveals the Y2k faults (if any) of a given program with high probability. The Y2k criterion is also an adequacy criterion that efficiently tests the program after the fixing of Y2k bugs.

Our method necessitates relatively few information as follows. We assume that the names of date-related input variables referred to as *Y2k variables* are known. These are the variables that may contain year-values such as “15/07/98”, “06/1998”, etc. Input means that the value of the variable is read from an input device or from a database. We also assume that we know the content of the date, i.e., which date-value involves years, years and months, years, months and days, etc. However, it is not necessary to know the internal

format of the date in the database or in the variable. This is important since the user usually knows which type of dates he/she enters, while the internal format is usually hidden. If these information are available, we can select the year-valued part from the entire value.

Since we have no any preliminary knowledge about the place of Y2k faults, a thorough examination of the target program is necessary. This requires the generation of a large number of program inputs that may reveal the Y2k faults with very high probability. As we will show in the last section, manual methods are inefficient and unreliable. In the previous section we described a method that reveals Y2k faults for a given output or predicate. However these instructions can be traversed along many program paths. Which ones should be selected? If we select quite a few, then the method will not be reliable. On the contrary, if we select all the paths, then the method will be extremely slow, though year 2000 is coming soon. Therefore, we introduce our *Y2k criterion* so that we address both issues.

There are two basic types of Y2k errors. First, an output may contain an erroneous value (Y2k computation error). Secondly, the program follows a wrong path (Y2k domain error). Both types of faults can be revealed by applying Y2k fault detection method. Now let us try to answer the question "Which program paths shall we select?". It can happen that reaching a given predicate along two different paths only one of them would cause a failure. Such a case occurs in the example described in the next section. Therefore our test selection is based on a stronger criterion. One of the strongest criterion is the all-paths criterion. Unfortunately, the number of different executable paths may be unbounded, therefore we introduce a manageable criterion as follows.

Definition 1 Consider a set of program paths P . P is adequate for the reduced all-path criterion if (1) every executable cycle-free program path p is included in P , (2) for every loop l in the program a path p is included in P for which l is iterated once and p is executable, (3) for every loop l in the program a path p is included in P for which l is iterated at least twice and p is executable.

Note that this criterion is a slightly modified and "applicable" (see [2]) version of boundary interior criterion [5].

To reduce testing effort drastically we apply *slicing*. Ignoring statements that are not related to the Y2K problem we obtain a significantly smaller code than the original one in most cases. The most important consequence of the reduced code is that our criterion has to be satisfied for this subprogram only. This results in a reliable test at a reduced cost. We consider each Y2k variable v one-by-one. First, we determine the forward slice for v at an instruction I_{in} , where v is assigned as an input. This means that the value of v is assigned without using any variable, i.e., its value is read from a database, from keyboard, etc. Then, we select output instructions. Next we determine the static (backward) slice for each output instruction I_{out} . With this we obtain the statements that may affect I_{out} . Then we intersect the statements that are in both slices. The new subprogram is called *double-slice*. The slicing criterion of a double-slice is a quadruple $C_d = (v, I_{in}, I_{out}, O)$, where v is the Y2k variable assigned (defined) in I_{in} , I_{out} is an output instruction for which the Y2k problem is analyzed, finally, O is a set of variable used in I_{out} . The reason of the use of double-slices is that we have to analyze each Y2k variable separately, and we would like to reduce the number of necessary program executions as well.

In this way a double-slice contains all the statements that are affected by an Y2k variable and which may have an influence on the selected output. Consider a loop-free program. In this case any program path in the control flow graph of the double-slice corresponds to a unique computation (representation) of the given output. Therefore, the coverage of all these program paths is as effective as the application of the all-paths criterion for the original program. The coverage of the reduced set of all paths in the slice is extended with the Y2k failure detection process described in Section 4. As result the Y2k criterion for a given output I_{out} is as follows:

Definition 2 *Y2k criterion for an output I_{out} . Consider an output instruction I_{out} in a program P . Assume that the set of Y2k variables is V and $v \in V$ is defined as input in instructions $I_{in}^v \in II_{in}$. Let us derive the double-slices $S_d(v, I_{in}^v)$ for all criteria $C_d = (v, I_{in}^v, I_{out}, O)$, $v \in V$ and $I_{in}^v \in II_{in}$. If all the S_d s are satisfied by applying the reduced all-paths criterion, such that the Y2k failure detection method is successfully applied for each selected input wrt I_{out} , the Y2k criterion for I_{out} has been satisfied.*

We can assume that the total number of Y2k variables is small and similarly, the number of instructions where Y2k variables are defined as input is small as well. In this case we have to determine only quite a few number of different double-slices.

In the Conclusion we show that the satisfaction of Y2k criterion is much more reliable for Y2k bugs than the satisfaction of the original all-paths criterion. The reason is that we apply our Y2k detection method instead of output value checking. Though we cannot cover each different computation of an output in the presence of loops (since the number of these computations may be unbounded), we strongly believe that our method is also reliable for any program. The method involving the automated test data generation, double slicing and the satisfaction of the Y2k criterion is illustrated in the next section. We call the readers attention that we should determine only the predicates in a double-slice. This information is sufficient to apply the Y2k criterion.

Now consider the case of predicates. We can also consider predicates as outputs, and thus the above method can be applied for predicates in a straightforward way. However, if a predicate p is in the double-slice of an output I_{out} , then satisfying the Y2k criterion for I_{out} involves the satisfaction of this criterion for p . Really, the set of program paths to be covered for p is a subset of the ones that satisfy the criterion for I_{out} . Therefore, it is sufficient to apply the Y2k fault detection method for the required paths that cover p and reach I_{out} (see the example in the next section). The definition of Y2k criterion for a program P is as follows:

Definition 3 *Y2k criterion for a program M . Consider the set of all output instructions II_{out} and the set of all predicates $p \in P$ in M . The Y2k criterion is satisfied for M if all the Y2k criteria for $I \in II_{out}$ and $p \in P$ are satisfied.*

5 TEST DATA GENERATION METHOD

In this section we introduce a method that generates test data covering each path in the double-slice and checks these with respect to Y2k problem. Our method

is based on either the general function minimization technique introduced in [7], and on [3] in which an automated method for domain testing has been described. Note that we execute the original program, and we use the slice only for path selection. First we describe a basic test data generation algorithm that modifies the flow of control at a selected branch such that the program follows the same prefix path. This method will be combined with a path management procedure to generate the required program inputs automatically.

Assume that we are given a program input I_0 for which a conditional statement p is executed along a complete path P . Our goal is to find such a program input I that alternates the boolean outcome of p keeping the execution of the prefix path to p unchanged. Remember, that the *branch function expression* F associated to a simple predicate expression $A \text{ op } B$ (where A and B are arithmetic expressions, and op is a relational operator) is $F = (A - B)$, that is a real-valued function of the current program input. Without loss of generality we can assume that F is positive for I_0 , and we should find a program input I , that satisfies the branch function condition shown in Table 1.

	op	I
case 1	$<, >$	$F(I) < 0$
case 2	$>, <$	$F(I) \leq 0$
case 3	$=, \neq$	$F(I) = 0$

Table 1:
Branch function conditions

An improved function minimization method will be applied to find I . The algorithm consists of two major parts:

1. exploratory search,
2. minimization procedure.

In the exploratory search we modify a selected input variable of the current program input keeping all the other input variables unchanged. The amount of the modification is a small value called *unit* that is associated to the data type of the selected input variable. For example in the case of integers unit is 1, and 0.1 in the case of floating point data types. First we increment a selected input variable by unit, re-execute the program and evaluate the branch function for the modified program input. If the branch function value has not decreased, then we decrement the same input variable by unit. If none of the modifications improve the branch function, another input variable is taken. This procedure continues until we find a direction in which the branch function reduces or none of the modifications decrease the branch function. In the latter case the exploratory search fails.

The goal of the minimization procedure is to find a local minimum value of the branch function along the direction determined by the exploratory search. In the first phase of the minimization procedure we repeatedly modify the current program input along the same direction. We say *constraint violation* occurs, if

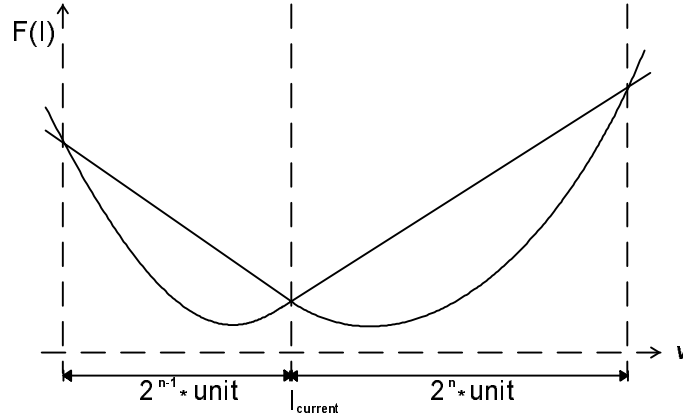


Figure 1: Minimization phase

the prefix path to p is not executed considering the double-slice for the modified program input. The amount of the modification is doubled after each step until the modified program input does not reduce the branch function, or constraint violation occurs. In the second (halving) phase another exploratory search is called for the current program input (restricting to the currently modified input variable) to indicate a possible new direction toward the local minimum, as it is illustrated in Figure 1.

At each step the amount of modification is halved, and the modified program input replaces the current one, only if it improves the branch function.

The above iterations are repeated until the branch function value becomes negative (or zero) changing the flow of control at the selected predicate, or the exploratory search fails for each input variable and the algorithm stops without producing the required program input. This latter case occurs, if the selected path is infeasible actually, i.e. there is no such a program input for which this path is traversed. The method can be used for compound predicates too (see: [3]) extending its application for a wider range of programs.

The complete test data generation algorithm uses the repeated application of the above procedure to cover all the required paths in the control flow graph of the double-slice. Paths for which the method fails are assumed to be infeasible.

The coverage algorithm starts with an arbitrarily selected program input, and explores all program paths systematically. In the beginning we have only one path P_1 that is traversed for the initial program input. At the first iteration we select the first predicate along this path, and apply the test data generation method described above obtaining a new path P_2 . At the second iteration we select the second predicates of all the available paths, i.e., along P_1 and P_2 . The test data generation procedure is now called for the second predicates of these paths one by one resulting four different program paths. At the i th iteration we select the i th predicates along all the generated paths, and apply the method for each path individually. Paths which contain less predicates than the current iteration number are stored in the final path set, and ignored during the further predicate selections. Note that all the generated paths are necessarily different,

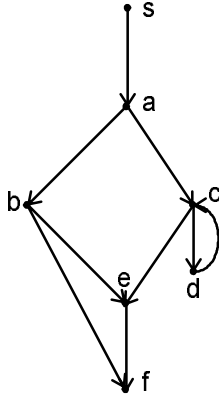


Figure 2: A sample CFG

since at the i th iteration the prefix paths of all the available paths up to the i th predicates are different. The algorithm would cover each loop zero times, once, and twice automatically, however, since we restrict to all-reduced-paths criterion, we have to skip loops which have been covered more than twice. Paths that correspond to zero, and one iteration are stored in the final test data set, but are ignored in further test data generation.

The complete path generation algorithm is demonstrated in the following example. Consider the control flow graph of a double-slice in Figure 2, and assume that a path $P_1 = \langle s, a, b, f \rangle$ has been traversed. P_1 has two predicates: a and b . At the first iteration we select predicate a , and apply the above method to change the flow of control at a obtaining a new path $P_2 = \langle s, a, c, e, f \rangle$. In the second iteration, we have to select the second predicates along the generated paths P_1, P_2 . First, take P_1 , select predicate b , and apply the test data generation method. We get a new path $P_3 = \langle s, a, b, e, f \rangle$. For P_2 , selecting predicate c , we obtain path $P_4 = \langle s, a, c, d, c, e, f \rangle$. Since paths P_1, P_2, P_3 contain only two predicate nodes, in the third iteration we have to deal with path P_4 only. Now we select the third predicate along P_4 which is the second occurrence of predicate c . The test data generation method is applied for predicate c along the prefix path $\langle s, a, c, d, c \rangle$ resulting a new path $P_5 = \langle s, a, c, d, c, d, c, e, f \rangle$. In the fourth iteration we should select the third occurrence of predicate c along P_5 , but as we can see the loop has already been traversed zero times (along P_2), once (along P_4), and twice (along P_5) satisfying our criterion. Since path P_5 does not contain any further predicates, we have finished the test data generation procedure.

It can occur that setting a Y2k variable to the sequence of years 1997, 1998, 1999, 2000, 2001, 2002 the paths differ for some of these program inputs. In this case a *path correction method* is applied, which takes the two program inputs that correspond to year 1997 and 2002, identifies the *problem predicate node* p at which the paths diverge, and calls the automated test data generation procedure for predicate p . Our program path alteration procedure is applied until we obtain the '*corrected program input*' that follows the same path for all

the selected years. Although it can happen, for a mediate test data still causes constraint violation, this case occurs rarely. The entire method is illustrated by the program fragment below.

```

read(current_year)
read(job_begin_year)
read(salary_class)
...
a  if current_year < 0 then
1   exit ('error')
...
b  if job_begin_year < 0 then
2   exit ('error')
3   salary = 80000
...
c  if current_year - job_begin_year >= 10 then
4   salary = salary * salary_class
   else
5   salary = salary + 1000 *
                               (current_year - job_begin_year)
   endif
6  write (salary)

```

The Y2k variables `current_year` and `job_begin_year` are assumed to be identified preliminary. First, we select `current_year` and statement 6, and generate the double-slice of the code. Since predicate *b* and statements 2 and 3 are not influenced by the read statement of `current_year`, therefore these are not in the forward slice. Predicate *a* and statement 1 are not in the backward slice for instruction 6 and variable `salary`, therefore the double-slice contains only one predicate *c*.

Our first subgoal is to find a set of program inputs that covers each individual program paths in the double-slice.

Let us select an initial program input I_1 and set `current_year` to 1997. Let I_1 be `current_year=1997, job_begin_year=1982, salary_class=1.0` ($I_1 = [1997,1982,1.0]$ for short), and execute the code obtaining a program path P_1 . In this case the *then* branch of *c* is executed. Since at the first iteration we have to select the first (and the only) predicate along the traversed path P_1 , we apply the automated test data generation algorithm for predicate *c*. The branch function expression is $F_c = \text{current_year} - \text{job_begin_year} - 10$, that yields 5 for I_1 . To alter the flow of control we should minimize the branch function value, until it becomes negative. The exploratory search selects the variable `job_begin_year` to increment. The steps of the minimization phase are shown in Table 2:

step size	job_begin_year	F_c
1	1983	4
2	1985	2
4	1989	-2

Table 2: Minimization steps

In the third step we find program input $I_2=[1997,1989,1.0]$ for which predicate c becomes false executing an uncovered new path P_2 .

Since P_1 and P_2 cover all the possible paths in the double-slice, our second task is to validate these with respect to the Y2k problem.

First, select path P_1 , generate the series of program inputs: $[1997,1982,1.0]$, $[1998,1982,1.0]$, $[1999,1982,1.0]$, $[2000,1982,1.0]$, $[2001,1982,1.0]$, $[2002,1982,1.0]$, and execute the code for these values. The branch function of predicate c is shown in the case of a correct program in Table 3/a.

date	F_c
1997	5
1998	6
1999	7
2000	8
2001	9
2002	10

Table 3/a: Branch function values of the correct program

date	F_c
1997	5
1998	6
1999	7
2000	-92
2001	-91
2002	-90

Table 3/b: Branch function values of the faulty program

Now assume that the above code contains a Y2k fault, because it considers only the last two digits for years. The branch function for the series of the above program inputs is shown in Table 3/b. The differences of the values of branch function are: $(1, 1, -99, 1, 1)$, i.e., the fault in the predicate has been detected.

Note that we cannot generate test data so that always P_1 is covered for all years 1997 – 2002 (assuming positive years). We can see that a fault may hide other faults, therefore repeated application of our method is required.

The next step of path validation procedure is to select path P_2 , and execute the series of program inputs $[1997,1989,1.0]$, $[1998,1989,1.0]$, $[1999,1989,1.0]$, $[2000,1989,1.0]$, $[2001,1989,1.0]$, $[2002,1989,1.0]$. Since constraint violation occurs for the third program input $[1999,1989,1.0]$ the path correction method is applied. The test data generation method now results in a new program input $I_2=[2002,1996,1.0]$ for which the required path has been traversed. The program now is executed for the new series of program inputs: $[1997,1996,1.0]$, $[1998,1996,1.0]$, $[1999,1996,1.0]$, $[2000,1996,1.0]$, $[2001,1996,1.0]$, $[2002,1996,1.0]$. The branch function values of predicate c and output values of variable `salary` in the case of the correct code are shown in Table 4/a.

date	F_c	salary
1997	-9	81000
1998	-8	82000
1999	-7	83000
2000	-6	84000
2001	-5	85000
2002	-4	86000

Table 4/a: Branch function- and output values of the correct program

date	F_c	salary
1997	-9	81000
1998	-8	82000
1999	-7	83000
2000	-106	-16000
2001	-105	-15000
2002	-104	-14000

Table 4/b: Branch function- and output values of the faulty program

Now assume that again, the program contains Y2k fault. The branch function values and the values of `salary` of the faulty code are listed in Table 4/b. We can see our method reveals the fault in the branch function again. Since for all the inputs path P_2 is traversed, the Y2k fault for statement 6 can be investigated. The differences of the output functions (here the output itself) are: (1000, 1000, -99000, 1000, 1000), therefore we detect a Y2k failure.

Analyzing this program we can justify the selection of our strong testing criterion. We can see that only instruction 5 contains a Y2k fault while statement 4 does not. Therefore, we have to execute a program path that covers the wrong statement and along which there is no redefinition for `salary` between this statement and an output or predicate that uses `salary`. (In our simple example this requirement is satisfied, but we can easily extend the example when this is not true.) Consequently we have to cover all simple paths in the double-slice.

The complete verification needs to inspect variable `job_begin_year` too in the same way. The 'outputs' `exit('error')` in statements 1 and 2 are also have to be considered.

6 RELATED WORK

Commercial products try to find the location where dates are employed. To do this, some date-manipulation sites can be identified by the places where there is a call to the operating system in the program. An example is when system clock containing current year is read into a variable. However, this variable can be assigned to other variables. Though by applying slicing we can observe each date-related variables and instruction, it cannot be decided whether these instructions suffer from Y2k bug.

Another method is to find date-manipulation sites by searching suspicious variable names. We can use an automated string-searching tool to identify some patterns, for example, `"*date"`, `"*yy"`, `"*year"`, etc., where `"*"` is a wild-card symbol replacing any substring. This method can also be extended by slicing to find other potential places of Y2k bug.

The main problem with these methods is that the number of potential locations of a Y2k fault is much larger than the actual number of Y2k faults in the program. In this way not only the correct code should be modified, but the huge database should also be changed. This is sometimes very difficult, and may entail newer faults. The other problem is that these methods cannot be applied for post-renovation testing, since these methods do not execute programs at all.

The only scientific method to address the Y2k problem has been developed by Reps et al. [9]. Their method is based on path profiling. A path profiler instruments programs so that the different executed program paths can be recognized. After several runs of the program a path spectrum can be obtained and displayed so that the frequency of different executed paths is determined.

This profiling technique is applied for date-dependent variables so that only the value of one date-dependent variable is modified keeping all other variables unchanged. After a lot of program executions, we can compare the path spectra for pre-2000 and post-2000 values. If the spectra are similar for some pre-2000

(post-2000) data, however they differ for pre-2000 and post-2000 data, a Y2k bug is assumed.

This method starts from similar hypothesis, i.e., the values of branch functions are probably different from pre-2000 and post-2000 years. However, there is a significant difference between this and our methods: Even if the values of branch functions are different for two runs, this does not always entail the difference of control flow to be followed. Consider our first example. We can observe that for both pre-2000 and post-2000 years the *else* branch is followed, therefore the path spectra are the same. To address this problem, more runs are required and by applying a threshold ratio we can decide when a Y2k fault is present. This method has two problems. The first is that we have to execute the program many times for both pre-2000 and post-2000 dates. In this case we cannot keep the value of all other variables unchanged. It is very difficult to select many appropriate (i.e. independent) input that would follow similar paths even for the same pre-2000 (post-2000) year. The input selection requires a deep knowledge of both the program and the database, a requirement we would like to eliminate.

Another problem is when the database contains lots of data that would result in the coverage of the *then* branch and only quite a few for which the *else* branch would be followed. In this case, even lots of test cases result in similar path spectra, the Y2k fault remains undetected. If the database is small, then it can occur that there is no data in this database for which the bug would be manifested. In this case path profiling technique surely fails for Y2k bugs. On the contrary, our method needs only 6 inputs to reveal a Y2k fault and this bug is captured for any input that follow a given program path. In addition, our method is more efficient, since we test all the different computations of an output (or predicate) function, while path profiling has no such a strict criterion.

7 CONCLUSION

This paper describes a new method that is able to detect Y2k-bugs automatically by using as few initial information as possible. The method can also be applied as an adequacy criterion to check whether a fixed program is Y2k bug-free or the renovation has to be followed. This is the first scientific method that is both safe and precise. This means that potential faults are revealed, but the code and the related database remain untouched if the method validates their correctness.

Though our method can be applied to reveal Y2k faults, this method is uniquely applicable as an adequacy criterion. All commercial tools can only be applied to possibly reveal Y2k bugs, but testing after the correction of the faults has to be done manually. Since the profiling method is not reliable as was shown in the previous section, our method is the only one which adequately tests programs after Y2k-bug correction. We note that there is no method to surely fix Y2k faults so that the adequacy test can be omitted. Even if all the Y2k variables are modified to 4-digit values both in the database and in the program, there may be some string transformation in the code inducing Y2k bugs. On the contrary, our method is fully automated, i.e., not only the test data are generated, but the validation is also automatic.

A reader knowing the software testing literature deeply might be doubtful

our assertion that the reliability of this method is almost perfect. This reader seems to be right because Howden in [6] showed that not even the all-paths criterion is able to reveal all the faults in a program. This is true. However, consider which type of faults may go undetected even if all-paths criterion has been satisfied.

First, a fault remains undetected if for some data the program would follow a wrong path, however, we select an inappropriate input. This case occurs in our first example if we select a non-pensioner employee that can happen frequently. For this person a correct path has been executed and we cannot find the fault even if all the paths have been executed. Since our program computes branch (and output) functions for 6 different years, our method reveals the bug for any input.

The second case that may result in an unrevealed fault after the satisfaction of all-paths criterion is coincidental correctness. Coincidental correctness occurs if for some badly selected input the incorrect program results in the same value as the correct one. For example, $y = x * x$ and $y = x + x$ result in the same value 4 for $x = 2$. Because we select six subsequent years as input, coincidental correctness cannot occur.

Finally, a program can contain places where some "boundary values" cause failures, while for all other values the program behaves correctly. An example for this type of fault is $y = 1/(10 - x)$, when only $x = 10$ can cause a run time error. However, our boundary value is 2000 that is just selected as input.

Inefficient working of our test data generation algorithm can also cause an unrevealed Y2-fault. However, our algorithm works correctly for linear predicates, and experiences show that most of the predicates (and predicate interpretations) are linear, see [11]. Even if a predicate interpretation is non-linear our algorithm probably finds input to cover a required paths. If not, then finding a local minimum may prevent to generate a selected path. However, we believe that this case occurs very rarely since quadratic (or higher order) date manipulation is usually a nonsense. Note that, however, there may be a predicate at which we should alter the flow of control to reach the selected output, and this predicate is non-linear. In this case other methods based on symbolic execution and non-linear programming can be applied or the path can be covered by human interaction.


Though the whole method has not been implemented yet, we have some preliminary results. The key part of our method (wrt the programming difficulties) is automated test data generation. Our first algorithm can generate a selected path very quickly in the intraprocedural case. (Only some milliseconds is needed by applying a HP 9000/856 computer.) Since the method involves the evolution of branch functions, therefore Y2k-fault detection method has actually addressed in practice.

Nowadays, both forward and backward slices can be determined even for interprocedural case [4, 8, 1]. Because we expect very small slices we can use conservative methods for arrays (and pointers if any). The problem of aliasing has also been resolved, thus we can conclude that our method can be implemented in practice.

References

- [1] Forgács, I. and Gyimóthy, T. An efficient interprocedural slicing method for large programs; Proc. of 9th International Conference on Software Engineering and Knowledge Engineering 279-286 June, 1997
- [2] Frankl, P.G. and Weyuker, E.J. An applicable family of data flow testing criteria; *IEEE Transactions on Software Engineering* SE-14, 8, 1988, 1483-1498
- [3] Hajnal, Á. and Forgács, I. An applicable test data generation algorithm for domain errors; *Proc. of. the ISSTA '98, ACM Software Engineering Notes* 23(2) 63-72, March, 1998
- [4] Horwitz, S., Reps, T., and Binkley, D. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems* 12, 1 (1990), 26-61.
- [5] Howden, W. Methodology for the generation of program test data. *IEEE Transactions on Software Engineering* SE-1, 5, 1975, 554-559
- [6] Howden, W. Reliability of the path analysis testing strategy. *IEEE Transactions on Software Engineering* SE-2, 3, 1976, 208-215
- [7] Korel, B. Automated software test data generation. *IEEE Trans. Softw. Eng.*, 8 (Aug.), 870-879, 1990.
- [8] Reps, T., Horwitz, S., Sagiv, M. and Rosay, G. 'Speeding up slicing' *ACM SIGSOFT Engineering Notes* 19(5) 11-20 December, 1994.
- [9] Reps, T., Ball, T., Das, M and Larus, J. The Use of Program Profiling for Software Maintenance with Application to the Year 2000 Problem. Proc. of the Fifth ACM SIGSOFT Symposium on Foundation of Software Engineering and Sixth European Software Engineering Conference, Zurich, Switzerland, Sept. 1997. and in LNCS 1301, 378-394
- [10] Weiser, M. Program Slicing. *IEEE Transactions on Software Engineering* SE-10, 4, 1984, 432-449.
- [11] White, L.J. and Cohen, E.I. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering* SE-6, 3, 1980, 247-257.

Slide 1




Product Quality Profiling (PQP)

Model Overview
Felix Silva
Hewlett Packard Company
LSG System Test Lab

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 1

Slide 2




Product Quality Profiling (PQP)

Working Definition

A tool to enable the alignment between customer requirements and product (or design) by explicitly correlating key product requirements to customer needs and expectations

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 2

Slide 3




Product Quality Profiling (PQP)

Objectives

- Facilitate the translation of “customer experience” elements into verifiable and quantitative attributes for system development and test
- Help establish commonly agreed upon goals of product performance by clearly understanding product environment constraints
- Contribute to the accurate characterization of customer operational profiles

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 3

Slide 4




Product Quality Profiling (PQP)

Characteristics

- Places value on customer task-related activities rather than product feature set
- Addresses the breadth of customer environments and tasks rather than the depth
- Improves requirements management by means of prioritization, traceability and tradeoff analysis
- Data collection mechanisms for required information already exist and/or require minor changes

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 4

Slide 5



Product Quality Profiling (PQP)

Terminology


Activity:
Weighted compound attribute containing details about target customer product usage model

Customer Usage Scenario:
End user action in support of a task (i.e., Spreadsheet p&p, Word Processing p&p, Drawings/Illustrations p&p)

Task:
Element of work performed by end user (i. e., Mailing Lists, Document Generation, System Administration)

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 5

Slide 6



Product Quality Profiling (PQP)

Terminology (CTND)

Solution:
All of the systems that together meet a customer need

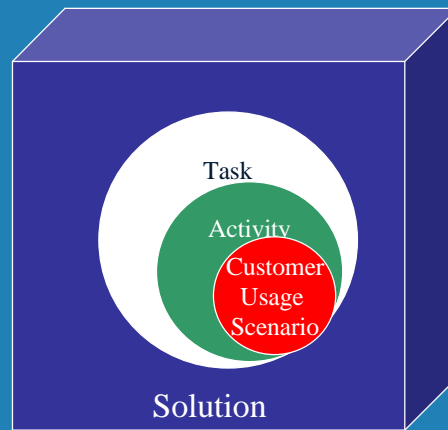
Solution Profiling Factor:
Measurable customer-bound product attribute to be targeted for validation and verification in the context of product development

Product Quality Profile:
Collection of solution profiling factors for a particular product agreed upon as collective goals of product customer fit

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 6

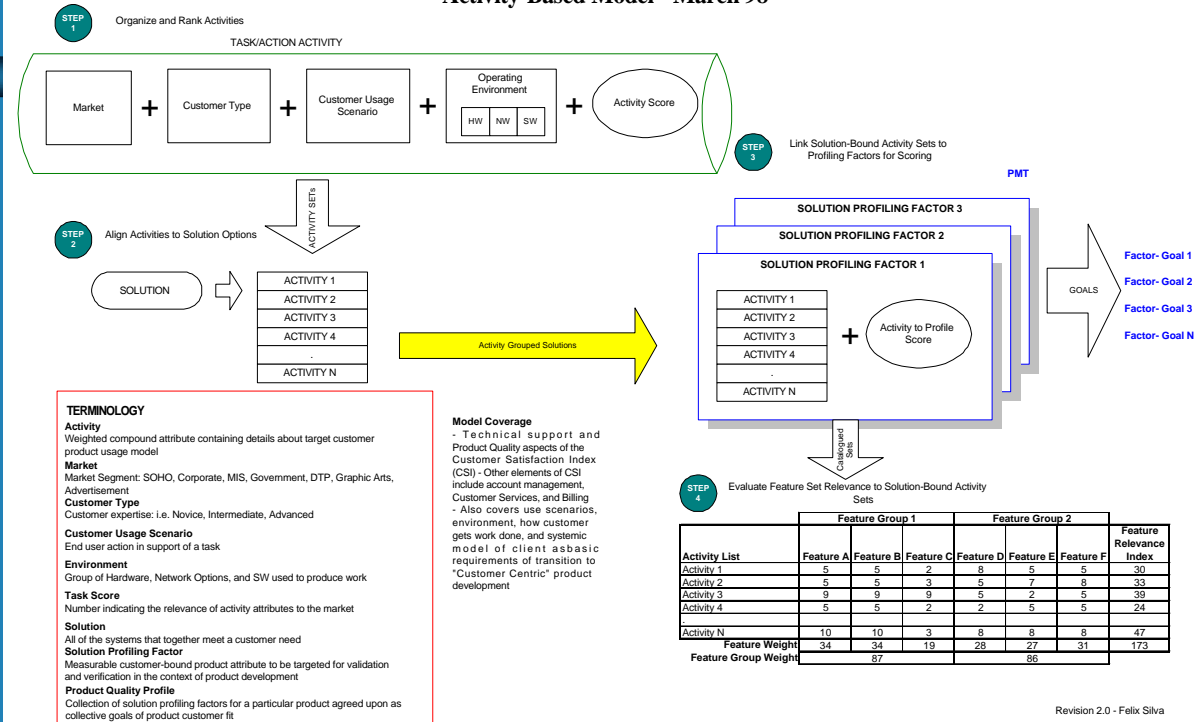
Product Quality Profiling (PQP)

Concept Hierarchy



Model

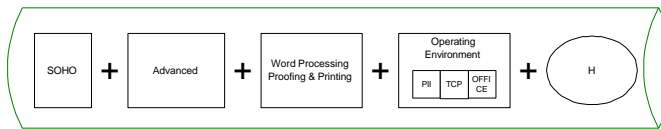
PRODUCT QUALITY PROFILING Activity-Based Model - March 98



Example

PRODUCT QUALITY PROFILING Model Illustration

STEP 1 Organize and Rank Activities

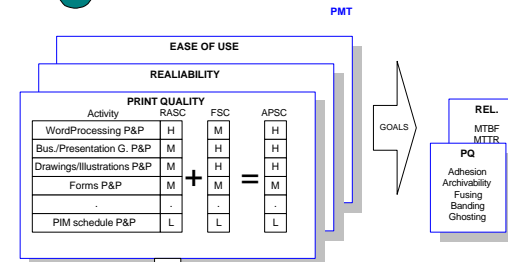


STEP 2 Align Activities to Solution Options

ACTIVITY SETS	WordProcessing P&P	H
DOCUMENT GENERATION	Bus./Presentation G. P&P	M
SYSTEM ADMINISTRATION	Drawings/Illustrations P&P	M
MAILING LISTS	Forms P&P	M
	PIM schedule P&P	L

Activity Grouped Solutions

STEP 3 Link Solution-Bound Activity Sets to Profiling Factors for Scoring




CONSIDERATIONS

- Solutions**
Solutions can be catalogued as an individual task or a connection of related tasks.
- RASC**
Raw Activity Score
- FSC**
Factor Score
- APSC**
Activity to Profile Factor Score

STEP 4 Evaluate Feature Set Relevance to Solution-Bound Activity Sets

Activity List	Print Quality			Paper Handling			Feature Relevance Index
	1200 DPI	RET	Multi-speed fuser	Universal Tray	Duplex	HCI Device	
WordProcessing P&P	5	5	2	8	5	5	30
Bus./Presentation G. P&P	5	5	3	5	7	8	33
Drawings/Illustrations P&P	9	9	9	5	2	5	39
Forms P&P	5	5	2	2	5	5	24
PIM schedule P&P	10	10	3	8	8	8	47
Feature Weight	34	34	19	28	27	31	173
Feature Group Weight	87			86			

Revision 2.0 - Felix Silva




Product Quality Profiling (PQP)

What PQP is:

- A mechanism to balance product technology, marketing, and user experience
- An activity that facilitates defining system requirements based on end user tasks profiles
- A pipeline for adding structure to requirements data already being mined
- A practical tool to help in the requirements elicitation process

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 10



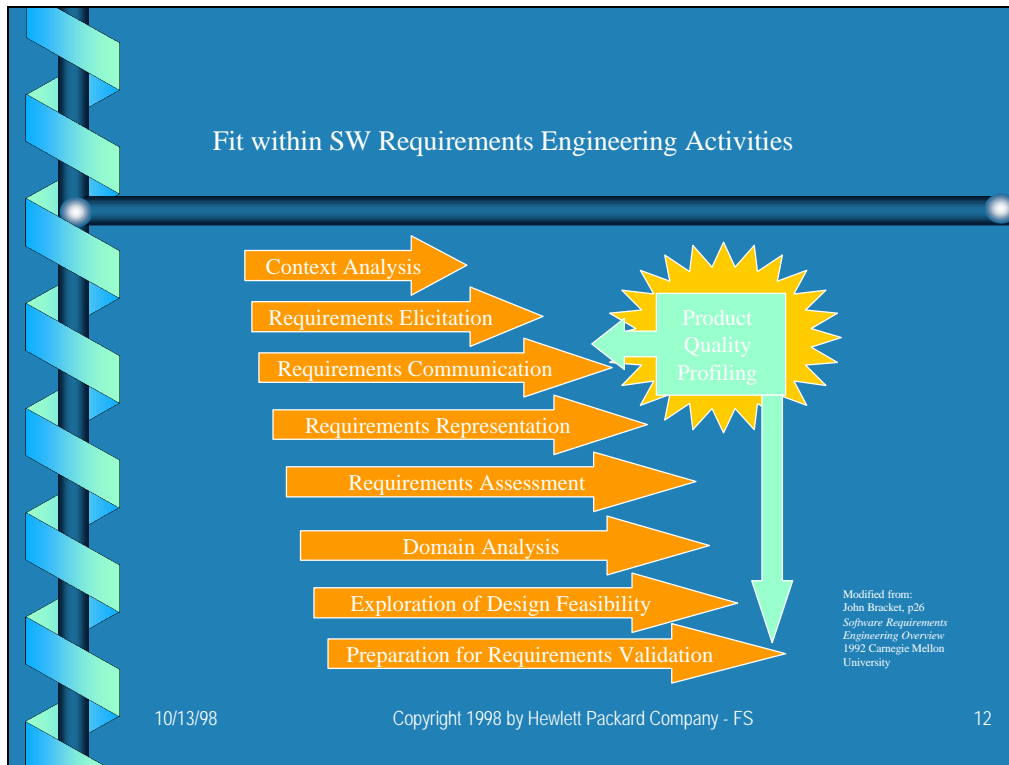
Product Quality Profiling (PQP)

What PQP is not:

- The solution to requirements elicitation
- A substitute for market research
- A guarantee of perfect market-user product match
- A substitute for requirement specifications
- A compulsory prescription for feature development

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 11

Slide 12



Slide 13

- ## Product Quality Profiling (PQP)
- **Advantages and potential**
 - Requires definition of attributes early in development phase
 - Highly leverageable once a baseline is established
 - Instant picture of highest profile environments to help in mitigating risk
 - Useful for derivatives, product rolls, and next generation products
 - Prudently using test resources
- 10/13/98 Copyright 1998 by Hewlett Packard Company - FS 13

Slide 14

Roles and Responsibilities

Cust. Support

- Goal Setting
- Contribute to profiling factor weighting



- Activity data
- Raw Activity Score
- Preliminary Solution Sets
- Contribute to profiling factor weighting
- Goal Setting

Test Lab

- Goal Setting
- Contribute to profiling factor weighting
- Update Model

Manufacturing

- Goal Setting
- Contribute to profiling factor weighting



- Feature to activity weight
- Contribute to profiling factor weighting
- Goal setting


10/13/98 Copyright 1998 by Hewlett Packard Company - FS 14

Slide 15

Product Quality Profiling (PQP)

- **PQP and SW Development**
 - Increased effectiveness of alpha and beta programs - and the like
 - Feature development clearly linked to end-user attributes
 - More effectively scope and deploy development resources
 - Easier to maintain momentum on technological innovation without eroding existing user base

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 15




Product Quality Profiling (PQP)

☀ Effectiveness Measures

- Alignment of user verifiable scenarios into quantifiable goals linked to the Customer Satisfaction Index (CSI)
- Accuracy of usage profiles/simulations
- Ease of structuring activity sets into product solution offerings
- Timeliness of PQP matrix deliverable

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 16



Product Quality Profiling (PQP)

☀ Conclusions

- PQP is an effective tool for structuring customer experience attributes to evaluate product fit
- Tedious at first, but pays good dividends in the long run
- PQP as a discipline helps organizations focus on the practicality of SW solutions to minimize any SW gold-plating

10/13/98 Copyright 1998 by Hewlett Packard Company - FS 17

Product Quality Profiling: A practical model to capture the experiences of software users

Felix M. Silva
Hewlett Packard
LaserJet Solutions Group Test Lab
11311 Chinden Blvd
Boise, ID 83714-1021
USA
e-mail: fsilva@boi.hp.com

Abstract:

When a software development engineer watches a user navigate through software the first time, the reaction is often shock and dismay. This reaction is largely due to the disconnect between how the user sees software and how the development engineer views it. Product Quality Profiling (PQP) is a model to help the system test engineer/organization bridge the gap between development engineers and software users. This paper presents the Product Quality Profiling model as a tool to structure customer experience attributes in alignment with product requirements and other significant quality attributes.

Background

Since the inception of the System Test Lab, it was assumed that end-user usage scenarios, work tasks, and environment (desktop) details were all neatly wrapped up in a package which could be used for generating system level test cases to validate the effectiveness of LSG product solutions. Not soon after making our first commitment to test a product, we realized that gathering this information was going to be a difficult and daunting task. Nonetheless, the research was done, as it was critical to validating configuration attributes for the product. This exercise indicated clearly that collecting customer experience information in this manner would not be optimal if we planned to handle more than one project every six months. Bear in mind that this product was a "simple" printer without many accessories and simplified operating modes.

Could we find a more effective method for gathering customer experience attributes? Initial research on this topic indicated that while our products have been successful and allowed us to maintain high market share, customer experience attributes are very fragmented in the product definition and development process. It has been difficult to

pinpoint one single area where the flow of attributes could be readily observed, catalogued or prioritized for validation. Finding a mechanism or tool to facilitate the collection and organization of these attributes was the thrust for the creation of the Product Quality Profiling (PQP). With the help of a task force, a PQP model was created to meet the needs mentioned above.

Product Quality Profiling (PQP) - Discussion

The PQP model is a tool to enable the alignment between customer requirements and product (or design) by explicitly correlating key product requirements to customer needs and expectations¹. Objectives of PQP include:

- ◆ Facilitate the translation of "customer experience" elements into verifiable and quantitative attributes for system development and test
- ◆ Help establish commonly agreed upon goals of product performance by clearly understanding product environment constraints
- ◆ Contribute to the accurate characterization of customer operational profiles

PQP is geared towards aligning customer actions into logical groups that preserve the essence of these actions and allow the translation of customer views into prioritized descriptive elements useful for product test and development. This translation process embodies the following four major characteristics:

- ◆ Places value on customer task-related activities rather than product feature set
- ◆ Addresses the breadth of customer environments and tasks rather than the depth
- ◆ Improves requirements management by means of prioritization, traceability and tradeoff analysis
- ◆ Organizes data collection mechanisms for required information which already exist and/or require minor changes

The characteristics above are quite clear on highlighting customer or consumer behavior as the key factor to capture for effective product deployment. In addition, PQP preserves a systemic view of the product within the targeted operating environment. As such it provides these value add attributes:

- ◆ A mechanism to balance product technology, marketing, and user experience²
- ◆ An activity that facilitates defining system requirements based on end user tasks profiles
- ◆ A pipeline for adding structure to requirements data already being mined
- ◆ A practical tool to help in the requirements elicitation process
- ◆ As the product is fine-tuned, PQP helps confirm that testing efforts are directed properly and if something changes or is altered, it becomes easier to quantify the impact

PQP - Model Details

The first order of business is to provide an introduction to the model terminology and the relationships among the terms. A *customer usage scenario* represents an end-user action in support of a task (i.e., Spreadsheet proofing & printing, Word Processing p&p, Drawings/ Illustrations p&p). An *activity* is a weighted compound

attribute containing details about target customer product usage model. At the next

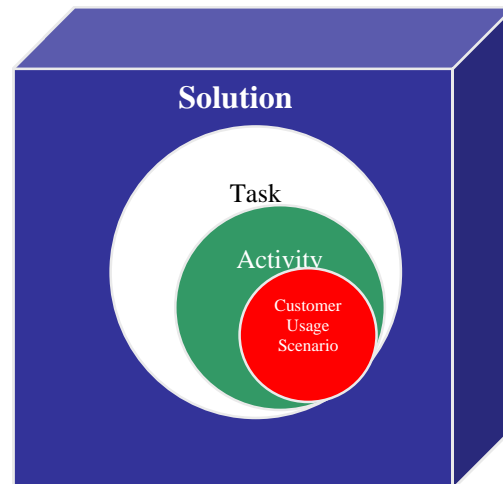


Figure 1 - Concept Hierarchy

level we define a *task* as an element of work performed by the end-user (i. e., Mailing Lists, Document Generation, System Administration). A *solution* represents all of the systems that together meet a customer need. An alternate way of looking at this definition is the set of tasks that must be addressed by the product in the customer environment. A *solution profiling factor* is a measurable, customer-bound product attribute to be targeted for validation and verification in the context of product development. Finally, a *product quality profile* is the collection of solution profiling factors (for a particular product) agreed upon as collective goals for product customer fit.

The drawing in figure 1 represents the overall concept of a *solution*. As such, the atomic level is the *customer usage scenario*. At the next layer, we detect groups of *activities* that contain different customer usage scenarios. Activities are ultimately in support of an item of work or a *task* that the end-user needs to accomplish. At the simplest level, a *task* can represent a *solution*. In reality, a solution is created to support logically grouped tasks (market niche).

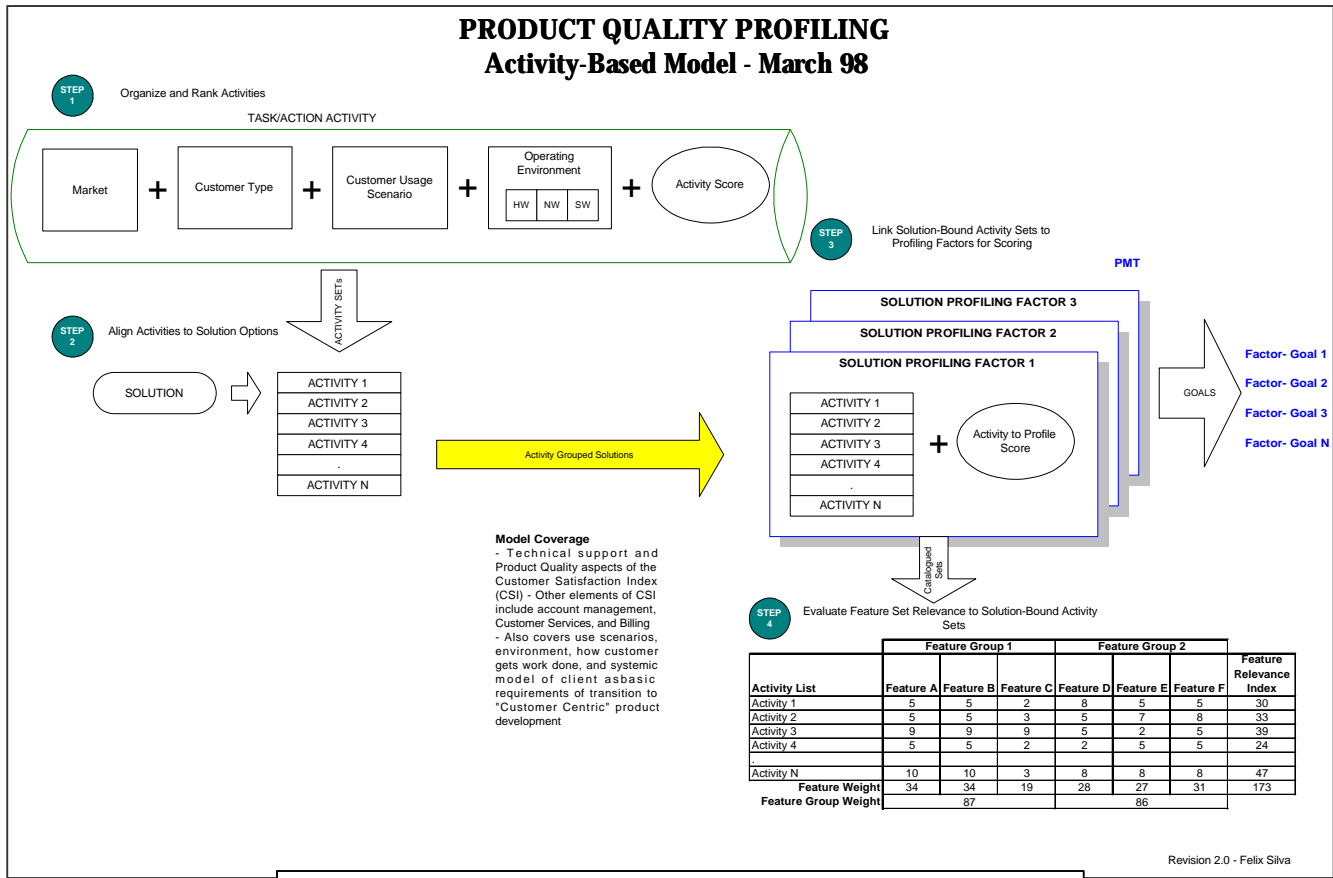


Figure 2: PQP Model Diagram

The concept of activities is key to the PQP model. Activities³ represent the capture of customer experience instances of highest relevance to the product. Consequently, by developing and validating the product against them, we insure that the deployed solution meets task-specific customer needs instead of unfocused feature wants. By focusing on activities, the core set of product features will meet all defined customer usage scenarios with a high degree of reliability.

For example (in the LaserJet arena), the *activity* of supporting insurance travel and real estate offices would be part of the *customer usage scenario* containing word processing proofing and printing. There are many others. Word processing is in turn only part of the work involved in the *task* of generating a report. As with other activities, the entire *solution* the customer uses includes their PC, Printer, monitor, e-mail,

hard copy & file habits and tools. The product is evaluated in terms of the part it should play in this entire sequence.

The Product Quality Profiling Model consists of 4 main steps [See Figure 2 (Model Diagram) for clarity]:

1. Organize and Rank Activities
2. Align Activities to Solution Options
3. Link Solution-Bound Activity Sets to Profiling Factors for Scoring
4. Evaluate Feature Set Relevance to Solution-Bound Activities

Step 1 is quite basic; however, it is one that is often not done in a highly structured manner. The cylinder in the model represents a "core sample" of many possible activities that exist within the realm of product use. As such, the attributes of this core sample represent a very specific consumer action of product use that embodies its associated operating

environment. Assigning each activity a weight (or score) can prioritize these attributes to indicate a relevant rank among all of the other activities that fall within the context of product-related end user tasks. Once a superset of these activities is created, we proceed to Step 2.

Step 2 consists of logically grouping activities into sets for which a solution is created. A different number of activities will be associated with a particular solution. Activities will likely be present in more than one solution, but their relative importance will differ when viewed within the context of that solution. The objective of this step is to line up the activities that make sense as a package to be addressed by a solution offering. The outcome of this step will be a number of prioritized activity grouped solutions.

Step 3 represents an additional level of activity scoring, but within the context of specific solution profiling factors. We begin by examining the relationship of each activity grouped solution set to a solution profiling factor and re-weighting the activity within the context of the factor. This yields a compound score for the activity that is not only relevant in terms of supporting customer usage, but is also relevant along other customer loyalty vectors such as the Customer Satisfaction Index. In the LaserJet Solutions Group (LSG), this added measure of activity evaluation is carried out at the Program Management Team (PMT) level with the expectation of setting goals for each factor during product definition stages. Once this step is completed, the catalogued sets of activities whose relevancy is clearly traceable and self documented is available project-wide.

Step 4 is a suggested way of mapping, evaluating, and tracking the relevance of features to solution-bound activities. Using a matrix similar to this one, it is possible to quantitatively assign a score to each feature that bears a direct relationship to the level of support the feature has on a particular activity. In this way, it is easy to identify errant features and to strengthen weaker ones. In lieu of explicitly detailed requirements documentation, one could use this matrix to distribute resources in

development and test to those areas of highest importance with a high level of confidence that the focus on detailed requirements documentation would solidify resource decisions once it becomes available. Finally and most importantly, the focus of any product decision will both backtrack to customer activity sets and not to features and help maintain the product course based on answering consumer needs and balancing the technology drive.

PQP and Software Development

A great software development challenge is to validate the product against real customer use cases or scenarios. Regarding LaserJets, we make use of programs such as alpha, beta, delta, and multiple install testing (MIT) to help establish a baseline of readiness for product release. While we typically get a "feel good" validation type from these kinds of tests, their deployment and management would be even more effective if we structured some of the elements of these programs along the lines of activities and customer profiling. By proactively organizing, cataloging, and prioritizing consumer use actions, we can select the most effective sites for each of the different test programs.

We have discussed mapping features to activities and scoring them to determine their relevance. This method is routinely used by Microsoft to avoid jumping into defining a product by features instead of customer needs. This concept can be readily applied to any commercial software product in the same manner thus, allowing us to get a better substantially structured knowledge of our customers. This is an asset of major importance to software development since we can clearly scope and focus teams on a core set of features that will effectively meet customer needs while at the same time dedicate teams to explore future customer delight technical features. In other words, we can determine with more accuracy the planning needed to implement the core solutions while getting a leg up on technological innovation.

PQP Benefits

Obtaining "good enough" or adequate levels of quality in a product does not translate into poor quality. For instance, as users become

more familiar with printing technology, they also demand better reliability and simpler product operation. It is based on these premises that PQP methodology offers the following advantages:

- Product attribute definition early in development phase
- High leverage once a baseline is established
- Instant picture of highest profile environments to help in mitigating risk
- Useful for product derivatives, product rolls, and next generation products
- Prudently using test resources
- Determining "finish" point

The creation of activity sets and solution groups for a product are intuitively front-end actions in product development. These key components are fundamental to PQP. As a result, products that consider profiling information as a stated requirement will build validation mechanisms strategies before the first line of code is written. Without a doubt, PQP is a learning process. Initially, it is a bit tedious, but not necessarily difficult to assemble useful profiling information for a single product in a divisional product line. However, it will be quite easy to leverage PQP efforts from one product into similar ones aiming for the same target market.

The development of product quality profiles is an automatic way of systematically prioritizing the most relevant user scenarios and environments where the product will be used. This carries along the additional advantage of providing additional data points to aid in the product risk management process.

One great advantage of customer driven profiles is the ability to preserve a validation environment that can be deployed to validate not only standard development cycle products, but also derivatives and product rolls. Given that the attributes of the PQP are correct and maintained with market trends and user behavior over the project life, the validation of product variants is an effective way of assessing product fit. PQP can act as a customer-experience regression test suite that validates functionality within the context of end-user work activities. It also becomes a leveraged

asset usable by Customer Support, R&D, Manufacturing and Marketing over the life of the product.

Conclusion

PQP is an effective tool for structuring customer experience attributes to evaluate product fit. The PQP methodology is built upon a framework that provides details about user environments, work tasks, and relative prioritization of user activities in support of these tasks. Current trends in customer loyalty indicate that to remain competitive, we can no longer rely on technological innovation alone. It is imperative that we understand what consumers actually do and how our products are used to support their work. PQP provides a channel to grasp this understanding by translating customer experience attributes into practical use cases. The collection of the attributes necessary for PQP implementation already exists in various forms. Implementing the methodology is therefore a matter of establishing the proper partnerships with the customer data gathering organizations (market research, product marketing, technical marketing, Customer Support) and those who will make project decisions based on the profiles (PMTs).

References

- ¹ Tran, T., and Sherif, J., "Quality Function Deployment (QFD): An Effective Technique For Requirements Acquisition."
- ² Norman, D., The Invisible Computer, MIT Press, Cambridge MA, 1998.
- ³ Cusumano, M., and Selby, R., Microsoft Secrets, The Free Press, New York NY, 1995.

APPENDIX A
Product Quality Profiling
Customer Usage Scenario Breakdown

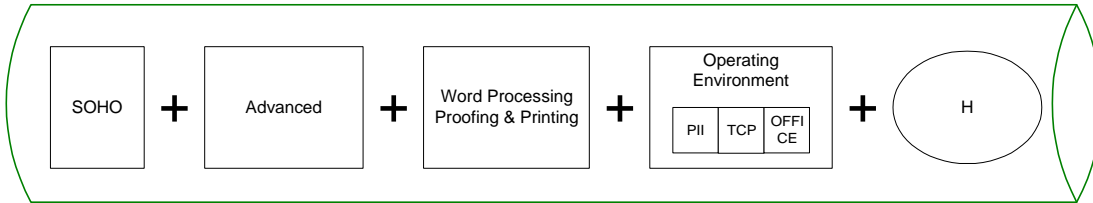
This is a sample list of CUS and related tasks for LaserJets – Each product will have to develop their own sets depending on their markets.

Task	Customer Usage Scenarios	Sub-scenarios	Comments
Mailing Lists	<ul style="list-style-type: none"> ◆ Spreadsheet proofing & printing ◆ Word Processing proofing & printing ◆ Drawings/Illustrations proofing and printing ◆ Forms proofing and printing ◆ Internet printing ◆ Workgroup e-mail printing ◆ Accounting proofing and printing ◆ Project Management schedule printing 	<ul style="list-style-type: none"> ◆ Collation ◆ Multiple copies ◆ Stacking 	This task refers to letters, labels and envelope printing to handle low volume mail mass production.
Document Generation – Composition, Redaction & Distribution	<ul style="list-style-type: none"> ◆ Word Processing proofing & printing ◆ Business/Presentation Graphics proofing and printing ◆ Drawings/Illustrations proofing and printing ◆ Forms proofing and printing ◆ Internet printing ◆ Workgroup e-mail printing ◆ Personal Information Management journal printing 	<ul style="list-style-type: none"> ◆ Collating ◆ Multiple copies ◆ Stacking ◆ Stapling ◆ Duplexing 	Generic task related to the creation of documents such as letters, memos, notes, papers, books, etc.
Telecommunications	<ul style="list-style-type: none"> ◆ Document faxing ◆ Internet printing 	<ul style="list-style-type: none"> ◆ Faxing Lists ◆ Multiple Copies 	
Document Warehousing/Archival	<ul style="list-style-type: none"> ◆ Document scanning ◆ OCR 	<ul style="list-style-type: none"> ◆ Multiple page document scanning 	

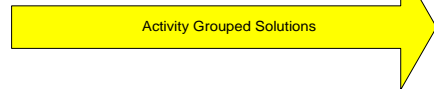
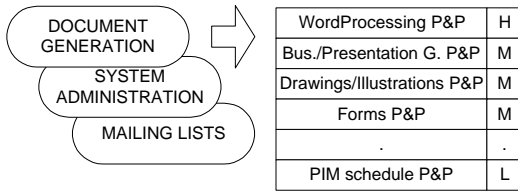
APPENDIX B

PRODUCT QUALITY PROFILING Model Illustration

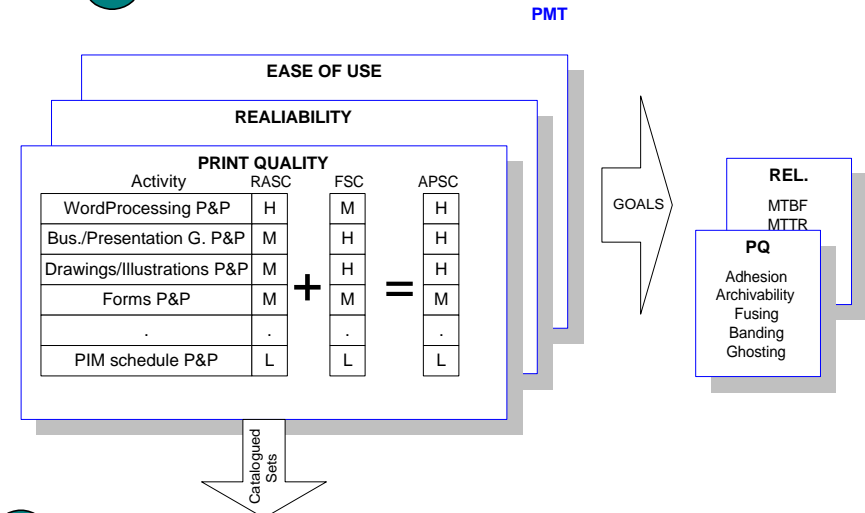
STEP 1 Organize and Rank Activities



STEP 2 Align Activities to Solution Options



STEP 3 Link Solution-Bound Activity Sets to Profiling Factors for Scoring



CONSIDERATIONS

Solutions
Solutions can be catalogued as an individual task or a connection of related tasks.

RASC
Raw Activity Score

FSC
Factor Score

APSC
Activity to Profile Factor Score

STEP 4 Evaluate Feature Set Relevance to Solution-Bound Activity Sets

Activity List	Print Quality			Paper Handling			Feature Relevance Index
	1200 DPI	RET	Multi-speed fuser	Universal Tray	Duplex	HCI Device	
WordProcessing P&P	5	5	2	8	5	5	30
Bus./Presentation G. P&P	5	5	3	5	7	8	33
Drawings/Illustrations P&P	9	9	9	5	2	5	39
Forms P&P	5	5	2	2	5	5	24
PIM schedule P&P	10	10	3	8	8	8	47
Feature Weight	34	34	19	28	27	31	173
Feature Group Weight	87			86			

Revision 2.0 - Felix Silva

APPENDIX C
Quality Profiling Factors – Subset sample of attributes used on LaserJets

Factor	Attributes
Reliability	<ul style="list-style-type: none"> ◆ Hours of trouble-free operation
Design	<ul style="list-style-type: none"> ◆ Sturdy and well-built ◆ Convenient Layout and design ◆ Noise level when printing ◆ Size ◆ Having an attractive appearance
Print Quality	<ul style="list-style-type: none"> ◆ Adhesion, Archivability, and Fusing (a measure how well the toner sticks to the paper) ◆ Banding (a measure of sensitivity of the printing process to mechanical noise) ◆ Ghosting (a measure of the sensitivity of the charging process to the electrophotography process (EP), a measure of the effectiveness of the self-cleaning feature of the drum) ◆ Image placement accuracy and registration (a measure of the firmware and hardware to control the positioning of the image on paper) ◆ Line width (a measure of the effectiveness of the EP to align dots in order to form a line) ◆ Offset (a measure of the effectiveness of the self-cleaning feature of the fuser) ◆ Optical density and Gamut (a measure of the firmware to control and produce correctly colors and shades) ◆ Print Defects
Ease of Use	<ul style="list-style-type: none"> ◆ Easy to understand printer control panel and messages ◆ Easy to replace toner cartridges ◆ Easy to clear paper jams ◆ Clear on-screen messages ◆ Able to print on different sizes of paper ◆ Able to print on different mediums ◆ Easy to print on any printer on network ◆ Ability to manage all users' print jobs ◆ Ability to stop print jobs ◆ Ease of diagnosing and resolving network problems
Prints Consistently	<ul style="list-style-type: none"> ◆ Across different media types ◆ Across related application types

Slide 1

Quality Week Europe '98

**Improved Requirements Engineering
Based on Defect Analysis**

-

Otto Vinter
Manager Software Technology and Process Improvement
Brüel & Kjaer Sound & Vibration Measurement
email: ovinter@bk.dk


© Brüel & Kjaer  Improving the Requirements Engineering Process 1

Slide 2

Defect Analysis from Error Logs

Basic Process Improvement Idea

- Analyze error logs from previous projects to extract knowledge on frequently occurring problems
- Change the development process by defining the 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

© Brüel & Kjaer  Improving the Requirements Engineering Process 2

The PRIDE Background

ESSI-PET (PIE no. 10438)


(The Prevention of Errors through Experience-Driven Test Efforts)

Project Results

- Requirements related problems are our prime cause of bugs (>33%)
- With improved test efficiency, released products could contain as many as 70% bugs related to requirements

Requirements Related Bugs Are Caused by:

- Missing requirements (1/3)
- Misunderstood requirements (1/3)
- Changes to requirements (1/5)

© Brüel & Kjær  Improving the Requirements Engineering Process 3


Defect Analysis from Error Logs

Definitions

- Bugs are anything between serious defects and suggestions for improvements
- Bug reporting starts in the integration phase

Error Logs Analysed

- PC Windows software development project
- Project sizes app. 8 person years
- 200 bugs analyzed from the error logs
- Bug reports covered a period until 12 months after first release

© Brüel & Kjær  Improving the Requirements Engineering Process 4


Defect Analysis from Error Logs

Bug Categorisation

- Based on a bug classification scheme by Boris Beizer:
 - Boris Beizer: *Software Testing Techniques*, Second Edition, Van Nostrand Reinhold
- Comprehensive set of bug categories
- Contains statistics from many projects

Analysis of Requirements Issues

- Error source (forgotten, misunderstood, tacit, wrong ...)
- Interface where the bug occurred
- Quality factor (functionality, reliability, usability, ...)
- Complexity of bug (correction cost)
- What could prevent the bug

© Brüel & Kjær  Improving the Requirements Engineering Process 5

Defect Analysis From Error Logs


What we found:

- 51% of the bugs were requirements related

Major requirements issues:

• Usability	64%
• External Software (3rd party & MS products)	28%
• Functionality	22%
• Other than the above	13%


(NB: there can be several issues in one bug)

© Brüel & Kjær  Improving the Requirements Engineering Process 6

Defect Analysis From Error Logs

Complexity (Correction Costs):

- 92.5% of the requirements related bugs were easy to correct, even when found late in the project.
- Difficult to fix problems (7.5%) were mostly related to external software (third party & MS products).
- However, finding and fixing the difficult problems takes 45% of the time

© Brüel & Kjær  Improving the Requirements Engineering Process 7


Techniques to Prevent Requirements Issues

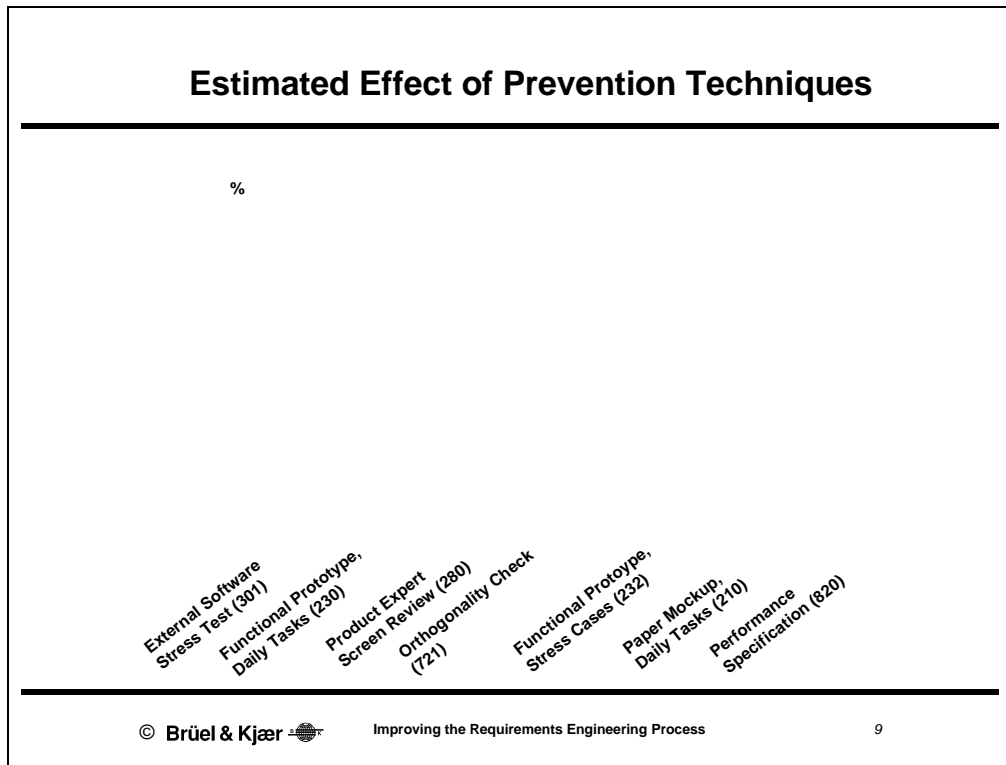
Groups of techniques identified

- 1xx Demand Analysis
- 2xx Usability Techniques
- 3xx External Software
- 4xx Tracing techniques
- 5xx Risk analysis techniques
- 6xx Formal specification techniques
- 7xx Inspection / Checking techniques
- 8xx Other techniques

Some 40 techniques were considered

- well-known
- from check lists
- invented when needed

© Brüel & Kjær  Improving the Requirements Engineering Process 8




Proposed Methodology

Requirements Elicitation and Validation

- **Scenarios (101)**
 - Relate demands to use situations. Describe tasks for each scenario.
- **Usability Test, Daily Tasks, Navigational Prototype (220)**
 - Check that the users are able to use the system for daily tasks, based on a navigational prototype of the user interface.

Verification of Requirements Specification

- **Let Product Expert Review Screens (280)**
 - Let a product expert check for deviations from earlier product styles.
- **External Software Stress Test (301)**
 - Test that the external software fulfills the expectations in the requirements, with special emphasis on extreme cases.
- **Orthogonality Check (721)**
 - Check of requirements specification to see whether an operation or feature can be applied whenever it is useful.
- **Performance Specifications (820)**
 - Check of requirements specification to see whether it contains **performance goals for the requirements.**

© Brüel & Kjær  Improving the Requirements Engineering Process 10


Experimentation with the Elicitation Techniques

Quantitative Results

- **Error logs were analysed and compared to a similar project**
 - same team, same platform, ...
 - person hours within 10 %
 - but almost 4 times increase in new screens

- **Developers were almost 3 times as productive in the development of the user interface**
 - Usability issues per new screen reduced by: 72 %


- **Total reduction in error reports: 27 %**
 - Requirements related reports: 11 %
 - Non-requirements related reports: 37 %

© Brüel & Kjær  Improving the Requirements Engineering Process 11

Experimentation with the Elicitation Techniques

Qualitative Results

- Scenarios gave the developers a clear vision of the user needs for the product
- User interaction with the product was totally changed as a result of the usability tests
- User reactions extremely positive
- Selling steadily more than twice as many
- Product was released in December 1997 and orders could be shipped before the end of the year


© Brüel & Kjær  Improving the Requirements Engineering Process 12

In Conclusion

- Scenarios capture the most important user needs
- Usability tests on early prototypes verify these needs
- The requirements verification techniques will further increase the prevention of bugs

- The impact on the perceived quality of the product is much greater than on the prevention of bugs
- The benefits of using the scenario and usability test techniques have a major impact on our business
- The requirements elicitation techniques are rapidly being adopted by other projects at Brüel & Kjaer

- Defect analysis is a simple and effective way to assess and improve the software development process

© Brüel & Kjaer  Improving the Requirements Engineering Process 13

European System and Software Initiative (ESSI)

An Accompanying Measure to ESPRIT


- The European Strategic Programme for Research and Development in Information Technologies

ESSI Objectives

- Promote Improvements in the Software Development Process in Industry
- Improve Current Practice by Applying State-of-the-art in Software Engineering
- Evaluate State-of-the-art Supports
- Disseminate Experience across Borders and Industrial Sectors

ESSI Lines of Actions

- Process Improvement Experiments
- Dissemination, Education and Training
- Software Best Practice Networks

© Brüel & Kjaer  Improving the Requirements Engineering Process 14


The PRIDE Process Improvement Experiment

**A Methodology for
Preventing Requirements Issues from
Becoming Defects
(PRIDE)**

PRIDE Objectives

- Extract knowledge on frequently occurring problems in the requirements engineering process
- Change the development process by defining the optimum set of methods and tools available to prevent these problems reappearing
- Measure the impact of the changes in a real-life development project


Subcontractor: Copenhagen Business School (Prof. S.Lauesen)

© Brüel & Kjær  Improving the Requirements Engineering Process 15


The Beizer Bug Classification Scheme

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
(incl. Third Party Products)
8. Test Definition or Execution Bugs
9. Other Bugs, Unspecified


Each category detailed to a depth of up to 4 levels

© Brüel & Kjær  Improving the Requirements Engineering Process 16

Defect Analysis from Error Logs		
Category	Our Analysis	Beizer Statistics
1. Requirements	26,0 %	8,1 %
2. Functionality	35,4 %	16,2 %
3. Structural	14,4 %	25,2 %
4. Data	2,2 %	22,4 %
5. Implementation	<u>2,8 %</u>	9,9 % (<u>5,9 %</u>)
6. Integration	0,6 %	9,0 %
7. Architecture (<u>2,7 %</u>)	7,1 %	<u>1,7 %</u>
8. Test	7,1 %	2,8 %
9. Unspecified	4,4 %	4,7 %
TOTAL	100,0 %	100,0 %

© Brüel & Kjær  Improving the Requirements Engineering Process 17

Requirements Related Categories
<ul style="list-style-type: none"> • Requirements and Features as Specified • Functionality as Implemented <ul style="list-style-type: none"> - Requirements misunderstood - Features missing or changed - Domain problems • External Hardware Interfaces and Timing • External Software Interfaces <ul style="list-style-type: none"> - Operating system - Third party software • Test Design <ul style="list-style-type: none"> - Requirements misunderstood

© Brüel & Kjær  Improving the Requirements Engineering Process 18

Defect Analysis From Error Logs

Disclaimer:

- Error reports only show which problems are left after coding and unit testing (e.g. late in the development process)

(A “re-inspection” of the requirements documents only showed a weak correlation with the issues found in the error reports)

Cost / Benefit Calculations

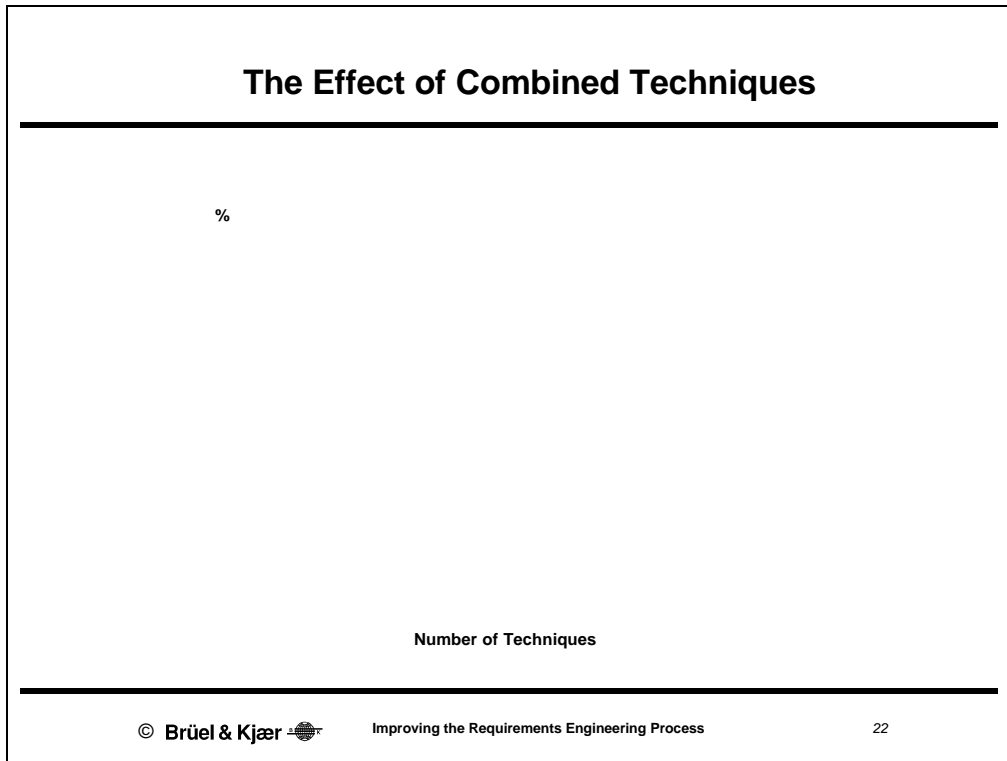
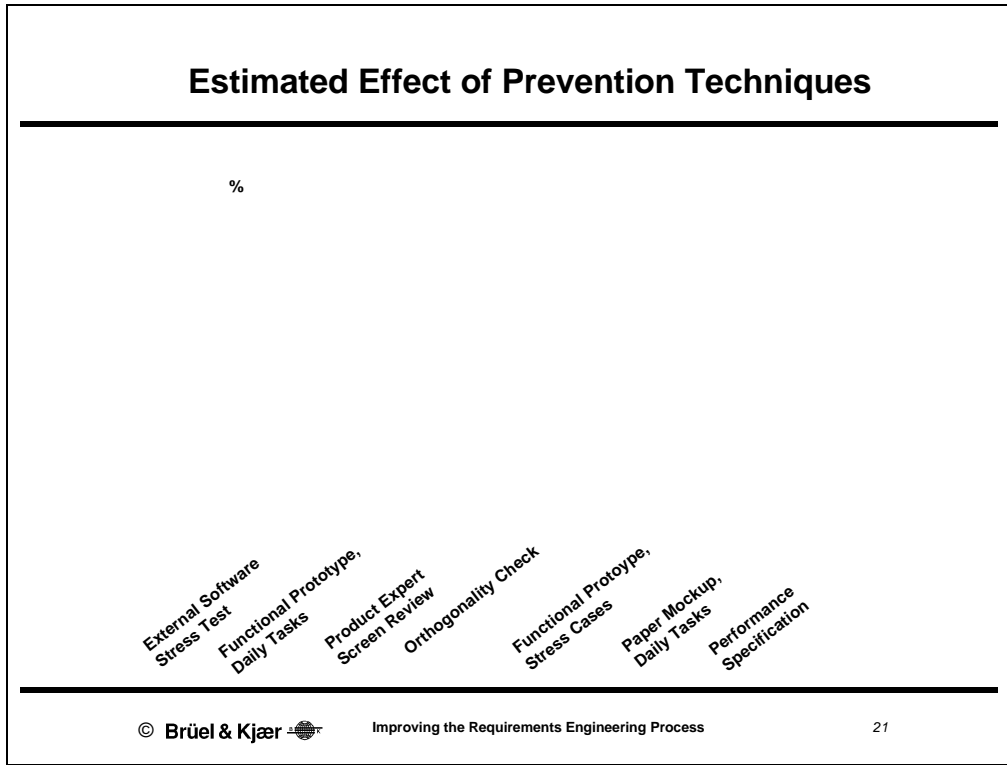
Hitrate_{technique} =

$$\left(\sum_{\text{bugs}} \text{FindProbability}_{\text{bug, technique}} \right) / \text{BugCount}$$

Savings_{technique} =


$$\left(\sum_{\text{bugs}} (\text{FindProbability}_{\text{bug, technique}} * \text{Benefit}_{\text{bug}}) - \text{Cost}_{\text{technique}} \right) / \text{DevelopmentCost}$$

Note: Benefit equals FindFixCost



Techniques Selected for Experimentation

- **Scenarios (101)**
 - Relate demands to scenarios. Describe the tasks in each scenario.
- **Functional Prototype Usability Test, Daily Tasks (230)**
 - Check that the users are able to use the system for daily tasks.
- **Let Product Expert Review Screens (280)**
 - Let a product expert check for deviations from earlier product styles.
- **External Software Stress Test (301)**
 - Test that the external software fulfills the requirements, with special emphasis on extreme cases.
- **Orthogonality Check (721)**
 - Specific check of requirements specification to see whether an operation or feature can be applied whenever it is useful.
- **Initial Value Check (730)**
 - Specific check to see whether objects on screens have meaningful initial contents.
- **Performance Specifications (820)**
 - Specify performance for frequent tasks. Set specific performance goals for affected requirements.


© Brüel & Kjær  Improving the Requirements Engineering Process 23

The Optimum Combination of Techniques

Combining the 4 best techniques on the analysed project:

- would have found 37,5% of requirements related bugs
- and saved 6,6 % on development cost

**The 18 month project could have gained
1 month on time-to-market !**

© Brüel & Kjær  Improving the Requirements Engineering Process 24

Techniques Used in Experiment

Scenarios


Relate demands to use situations. Describe the essential tasks for each scenario.

Write down short descriptions for each known use situation. A report from focus groups is a good source. Otherwise interview product and domain experts.

Usability Test on a Navigational Prototype

Check that the users are able to learn and use the system for daily tasks.

This technique uses a navigational prototype of the user interface, tests it with users simulating daily tasks, revises the design, tests it again, and so on until the result is acceptable.

© Brüel & Kjær  Improving the Requirements Engineering Process 25


Experimentation with the Elicitation Techniques

Quantitative Results

- Error logs were analysed and compared to a similar project
 - same team, same platform, ...
 - person hours within 10 %
 - but 5 times increase in new screens

- Total number of error reports: - 27 %
- Requirements related reports: - 11 %
- Non-requirements related reports: - 37 %

- Requirements Issues
 - Usability issues: + 5 %
 - Usability issues per new screen: - 79 %
 - Other requirement issues: - 36 %


© Brüel & Kjær  Improving the Requirements Engineering Process 26

Results of the Experiment

- Usability tests were performed on a Navigational prototype using MS Word6 Bookmarks
- Three iterations of the prototype was needed


- Scenarios and tasks were included as part of the requirements specification
- The prototype acted as a visual supplement to the requirements specification

- The requirements phase was longer than normal
- The specification and design phase was almost eliminated
- The rest of the development process proceeded as usual

© Brüel & Kjær  Improving the Requirements Engineering Process 27

Important Techniques

- 101 Scenarios
- 21x Paper mockup usability test
 - 210 Daily tasks
 - 212 Stress cases
- 23x Functional prototype usability test
 - 230 Daily tasks
 - 232 Stress cases
- 280 Let product expert review screens
- 301 External software stress test
- 70x Formal inspections
 - 702 Formal inspection of mockup
- 71x Object model checks
 - 710 Improved check of object model
- 72x Consistency reviews
 - 721 Orthogonality check
- 730 Initial value check
- 820 Performance specifications

© Brüel & Kjær  Improving the Requirements Engineering Process 28

Usability Techniques (1/2)

200 User data model

21x Paper mockup usability test


- 210 Daily tasks
- 211 Rare tasks
- 212 Stress cases

22x Screen mockup usability test

- 220 Daily tasks
- 221 Rare tasks
- 222 Stress cases

23x Functional prototype usability test

- 230 Daily tasks
- 231 Rare tasks
- 232 Stress cases

© Brüel & Kjær  Improving the Requirements Engineering Process 29

Inspection / Checking Techniques (1/2)

70x Formal inspection


- 700 Formal inspection of req.spec.
- 701 Formal inspection of object model
- 702 Formal inspection of mockup

71x Object model checks

- 710 Improved check of object model
- 711 Object model with external objects

72x Consistency reviews

- 720 Consistency check
- 721 Orthogonality check
- 722 Uniformity check

© Brüel & Kjær  Improving the Requirements Engineering Process 30


Scenario Example

Production Planning in a Shipyard

Detailed production planning is done by a man in the foreman's office. At times there is a busy traffic of workmen from the shipyard and from subcontractors, who want to know what to do next. Or they may have encountered a problem with a job. The primary job of the detail planner is to ensure that the right person gets the right job instruction. The job instruction is an index card with a description in text and sketches.

Task A: Give a workman the right job to do
Task B: Report job completion with wage information, materials used ...
Task C: Find someone else to do the job if problems are encountered
Task D: Write a letter to a subcontractor
.....

Source: Jens-Peder Vium, IQM

© Brüel & Kjær  Improving the Requirements Engineering Process 31


Reactions to Techniques Practised until now

Scenarios (101)

- "I am in fact deeply surprised. The scenarios made it possible for us to see the flow through the system in a very concrete way."
- "In the beginning of the project I was quite sceptic. I thought it would take too long time. But now I think we get a much more live and exciting requirements specification as a result of the scenarios. It will also make it much easier to make a prototype".
- "It has been an exciting experience to use scenarios. When you had the scenarios then the requirements were dropping out themselves"

Usability Test of Functional Prototype (230)

- "It only took a week to develop the original functional prototype in Visual Basic, and the modifications from the first set of tests to the next was performed overnight in the hotel room"
- "The closer we got to the real users, the clearer became the actual tasks that they performed"
- "We got more information out of the tests than we are able to incorporate in the product. We found features that we had never thought about, as well as features that were irrelevant to the users"

© Brüel & Kjær  Improving the Requirements Engineering Process 32

Scenario example 2: (Road Test)

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.

Scenario example 3: (Noise Source Analysis)

The test object (Air Conditioner, Car engine, Jet engine) is moved into a test cell, and the emitted noise from the object is measured at various points on a surface around the object.

The purpose is to present a contour-map across the surface that shows where the noise comes from, so that the engineer can see exactly where to change the design of the test object, to reduce the noise most effectively.

When interesting behaviour of the noise, e.g. a steep change in noise level or -flow, is observed at certain positions, a finer grid around that position is established. The measurements require that sound intensity probes are moved over the surface manually or controlled by a robot. The measured spectra are inspected manually for each point before the probe is moved to the next position.

When all measurements have been made, the test cell can be used for other test objects. Calculations of the sound power emitted from the test object and presentations of the results are performed separately.

Improved Requirements Engineering Based on Defect Analysis

Otto Vinter

Brüel & Kjær Sound & Vibration Measurement A/S, DK-2850 Naerum, Denmark
Email: ovinter@bk.dk

Søren Lauesen, Jan Pries-Heje

Copenhagen Business School, DK-2000 Frederiksberg, Denmark
Email: slauesen@cbs.dk, pries-heje@cbs.dk

Abstract

The basis for this paper has been a thorough analysis of error reports from actual projects. Error reports which were requirements related have been studied in detail with the aim of finding effective prevention techniques, and try them out in practice. The paper will cover some of the analysis results, a set of effective prevention techniques, and also some practical experiences from using some of these techniques on real-life development projects.

1. Introduction

At SQW96 and QWE97 Brüel & Kjær reported the experiences of a software process improvement (SPI) project where we demonstrated that the introduction of static and dynamic analysis in our software development process had a significant impact on the quality of our products.

The basis for this project was a thorough analysis of error reports from previous projects which showed the need to perform a more systematic unit test of our products. However, the analysis also showed that the major cause of bugs stemmed from requirements related issues.

We have now conducted another SPI project where we have analysed the requirements related bugs in order to find and introduce effective prevention techniques in our requirements engineering process.

The analysis of the requirements related bugs led to a set of techniques that would have been effective on the analysed projects. Some of these techniques were selected for experimentation (validation) on a real-life project. This project is now complete, and the product released. We have analysed the error reports from this project and the practical experiences and major impacts of the techniques used will be reported.

2. Analysis of Requirements Issues

There is no generally accepted way of making a requirements specification. Recommendations like the IEEE Guide [1] and Davis [3] are definitely helpful, but most developers have great troubles following them. What should be included and what not? How can you formulate functional requirements without committing to a specific user interface? How can you formulate "what" without describing "how"?

Through our analysis of error reports in a previous SPI project aimed at improving the efficiency of our testing process [4] [5] we have found that requirements related bugs are the major cause for bugs. We therefore decided to conduct this SPI project [6] aimed specifically at these type of bugs in order to find and introduce effective prevention techniques for requirements problems in our development process.

In both projects we have classified bugs according to a taxonomy described by Boris Beizer [2]. For the current SPI project, however, we have limited the study to those bugs which can be related to requirements issues. We found that requirements related bugs represented 51% of all the bugs analysed.

Furthermore we have 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.

Usability errors also seem to be rather easy to correct even when found late in the development process, e.g. in or after the integration phase. Problems with 3rd party products, however, are generally very costly to correct/circumvent.

This has had an impact on our methodology, tools and training. It had to be much more focused on usability problems, and early verification and validation techniques, rather than correctness, and completeness of requirements documents.

3. Potential Prevention Techniques

When we classified the bugs, we tried to imagine what could have prevented each bug. We started out with a list of known techniques and added to it when no technique seemed to be able to prevent the bug in question. Later when we discussed hit-rates for each technique, we improved and specified each technique further.

Many well-known techniques were considered but dropped, because we could see no use for them in relation to the actual bugs. Initially, for instance, we thought that argument-based techniques could be useful, but the error reports did not show a need for them. Others like formal (mathematical) specifications, seemed of little value. Techniques with focused early experiments on different kinds of prototypes, seemed much better suited to catch real-life bugs.

Many of the proposed techniques are “common sense” techniques that are moved up from the design phase to the requirements phase and formalised. When they are used in this context they will ensure the right quality of the product.

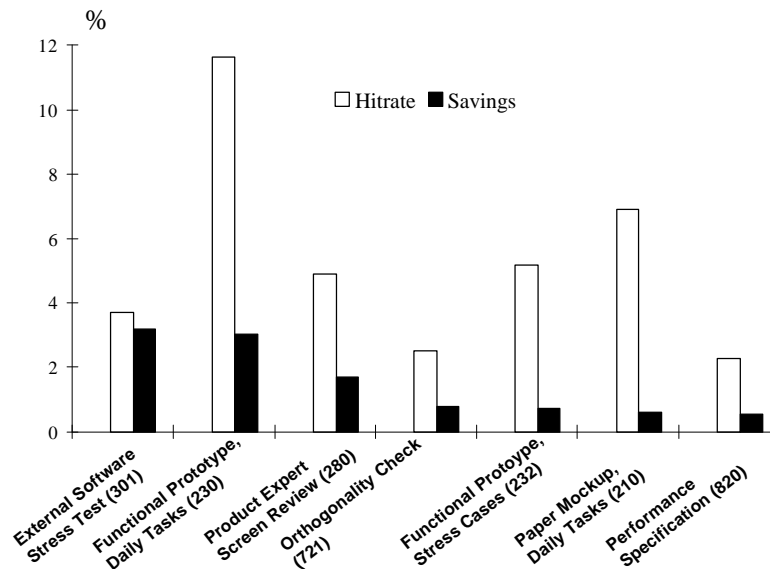
The result was a detailed list of some 40 prevention techniques grouped under the following major subjects:

- 1xx Demand analysis (including scenarios)
- 2xx Usability techniques (including prototypes)
- 3xx Validation and testing of external software
- 4xx Requirements tracing
- 5xx Risk analysis
- 6xx Improved specification techniques (e.g. formal/mathematical)
- 7xx Checking techniques (including formal inspections)
- 8xx Other techniques (including performance specifications)

4. Determining the Optimum Set of Techniques

Each error report was then assigned an estimated hit-rate for each technique, and the estimated effectiveness of each technique was calculated. We also assigned a benefit for preventing each error report, so that we were able to calculate the cost/benefit ratio of each technique, and then select the optimum set of techniques to be employed in our real-life experiment on a baseline project.

The results are shown in the figure below for the top seven techniques with respect to savings. The hit-rates are shown as a percentage of the total number of bugs in the project. The savings are shown as a percentage of the total development effort.



We have chosen to show only the top scorers with respect to savings. Other techniques had hit-rates comparable to the ones shown above, but with lower savings (even negative) because of high costs.

When more than one technique is used at the same time one must be aware, that combining two techniques does not simply add their hit-rates, because the first technique "filters" away some problems, leaving fewer problems for the second technique to detect. And this has an effect on the savings too. In general it will be better to combine techniques that find different kinds of problems. We have applied the principle of dynamic programming to calculate these combined hit-rates and savings and have found the best combinations with respect to savings.

The combination of the four best techniques above on the analysed project would have resulted in a combined hit-rate of 19% of all error reports (37% of the requirements related), and a combined saving of 6 % on the total development effort, which would have been approximately 1 month on the 18 month schedule of the project.

5. The Requirements Engineering Methodology

The results of the analysis were presented to the members of a new development project. Based on the detailed list of techniques and hit-rates/savings, the team took part in the final decision on the techniques of the methodology.

The techniques of the methodology are:

- **Requirements Elicitation and Validation**

- *Scenarios (101)*
 - Relate demands to use situations. Describe the essential tasks in each scenario.
- *Navigational Prototype Usability Test, Daily Tasks (220)*
 - Check that the users are able to use the system for daily tasks based on a navigational prototype of the user interface.

- **Verification of the Requirements Specification**

- *Let Product Expert Review Screens (280)*
 - Let a product expert check screens for deviations from earlier product styles.
- *External Software Stress Test (301)*
 - Test that the external software fulfills the expectations in the requirements, with special emphasis on extreme cases.
- *Orthogonality Check (721)*
 - Check the requirements specification to see whether an operation or feature can be applied whenever it is useful.
- *Performance Specifications (820)*
 - Check the requirements specification to see whether it contains performance goals for the requirements.

The analysis of error reports had not found that technique 101 (*Scenarios*) was effective in itself. However, since usability was such an important issue, we needed a technique to define the tasks that users should perform during the usability tests. We therefore included 101 as one of the techniques in the methodology, also because other sources indicated that scenarios were quite effective to improve developer understanding of the domain.

Originally the team focused on technique 230 (*Functional Prototype Usability Test, Daily Tasks*) as the choice of prototype for usability tests, because they were worried that they would not get enough out of a paper mockup (210) or a navigational prototype (220). On the other hand we were worried that a functional prototype would not be available for usability tests until too late to allow for the changes to requirements that would be uncovered by these tests.

What actually happened was that after the use situations (scenarios) had been described, the team could not wait for a functional prototype to be developed. In only two weeks they developed a first prototype with navigational facilities (screen mockup). Both VisualBasic and the Bookmark feature in Word 6 was used to develop further prototypes. These navigational prototypes were immediately subjected to usability tests, and the amount of issues found and corrected convinced the team that a navigational prototype would be sufficient.

Thus, the technique that is included in our methodology is not 230 (*Functional prototype usability test, daily tasks*), but another of the analysed techniques: 220 (*Screen mockup usability test, daily tasks*). Even though this technique is estimated to have only half the hit-rate and much lower savings than technique 230.

6. Experiences with the Techniques During the Experiment

The development team picked up the scenario and usability test techniques with great enthusiasm. In interviews, statements like the following were heard:

Scenarios (101):

I am in fact deeply surprised. The scenarios made it possible for us to see the flow through the

In the beginning of the project I was quite sceptic. I thought it would take too long time. But now I think we get a much more live and exciting requirements specification as a result of the scenarios. It will also make it much easier to make a prototype.”

It has been an exciting experience to use scenarios. When you had the scenarios, then the requirements popped up by themselves”

Screen Mockup Usability Test, Daily Tasks (220):

It only took a week to develop the original prototype in Visual Basic, and the modifications from the first set of tests to the next were performed overnight in the hotel room”

The closer we got to the real users, the clearer became the actual tasks that they performed”

We got more information out of the tests than we are able to incorporate in the product. We found features that we had never thought about, as well as features that were irrelevant to the users”

The other techniques of the methodology were never applied by the team in practice. Introducing so many new techniques at the same time on a project turned out to be too ambitious.

The requirements engineering process took longer than expected, but the specification and design phases were reduced, so the resulting delay was not considered critical. The rest of the development process was conducted in the usual fashion, following the normal procedures for development of

software at Brüel & Kjær. The baseline project was completed in December 1997 and the product was released.

7. Results of the Experiment

We have analysed the bugs from the completed project in the same manner as we did in the original analysis reported in chapter 2-4. We compare the results to another project previously developed by the same team on the same platform and under similar circumstances. The actual number of person months on the two projects is within 10%.

The major difference between the two projects is that the project used for experimentation was expected to be very user interface intensive. Actually it contains almost 4 times as many new screens as the project we compare it to.

The application it was intended to support had previously resulted in a lot of 'shelfware' products both from B&K and our competitors, because the customers were unable to grasp the intricacy of the measurement standard that should be followed. The usability techniques to be experimented with would seem especially suited for the experiment.

7.1. Effect on Requirements Related Issues

We have found an overall reduction in error reports of 27% from the previous generation of the product to the experimented product. The reduction in the number of requirements related error reports was 11%. According to our analysis, the actually used techniques (101 and 220) were estimated to achieve a combined hit-rate of 8% of all error reports and 15% of the requirements related.

When we study the distribution of requirement issues according to quality factors, we see a slight increase in usability issues (5%), whereas other requirements issues (functionality etc.) have been reduced by 36%. The immediate reaction to this is that the usability techniques employed have not reduced usability issues.

However, the impact of usability techniques is closely linked to the complexity of the user interface. The baseline project had almost 4 times as many new screens as the previous project we compare it to, all of comparable complexity. If we adjust for this difference, we actually have achieved a 72% reduction in usability issues per new screen, which is quite extraordinary.

Furthermore, the baseline project only spent 33% more person months to deliver almost 4 times as many new screens of comparable complexity. This almost 3 times difference in productivity can be explained by the design and development of the user interface being a stable process once the navigational prototype (screen mockup) had been validated in usability tests. In the previous project the new screens were constantly subject to change all through to the end of the project.

Finally, we have analysed the error reports from the baseline project to study hit-rates and savings in order to find further techniques that could have been employed with effect on the remaining bugs. We have found that none of the usability test techniques on prototypes are any longer among the top 7 candidates with respect to savings.

This shows that the usability test techniques have been effective in preventing requirements related bugs, and that using a navigational prototype (screen mockup) instead of a functional prototype seems to be adequate to prevent this type of bugs. This is important since the cost to build a navigational prototype is lower than building a functional prototype and can be performed much earlier in the development life-cycle.

Furthermore the requirements verification techniques of our methodology are still on the top of the list with respect to savings, and ranked relatively as follows: 280 (*Let Product Expert Review Screens*), 820 (*Performance Specifications*), and 721 (*Orthogonality Check*). This shows that these verification techniques of the methodology could have been an important supplement to the validation techniques actually used.

The baseline project did not use external software, so technique 301 (*External Software Stress Test*) has not been validated. Our original analysis showed that this technique would have been very effective in preventing problems that are difficult to fix on projects with new external software.

7.2. Other Effects

What is also surprising is that not only did we experience a reduction in bugs related to requirements issues, we found an even higher reduction in other bug categories (37%). We have been very puzzled about this unexpected result. We have thought of several causes that might have influenced the result.

1. The primary effect of the used techniques
2. Derived effects of the used techniques
3. Focus on a team improves their productivity no matter what else is changed.
4. Random effects
5. A change in the experimenters' evaluation of the reports
6. Differences in the team/culture/domain/project

Since we are comparing the results with another project previously developed by the same team, within the same domain, and under similar circumstances, we can eliminate cause 6.

Two out of three persons on the evaluation team (cause 5) have taken part in all the analyses and comparisons of error reports. The analysis reported in chapter 2-4 took place two years ago, but we have had to revisit some of the original error reports during the present comparison of results, and we found a reasonable agreement with our previous analysis.

We cannot completely rule out random effects (cause 4). However, the observed differences are within standard confidence limits so the reduction cannot be attributed to random effects only.

Nor can we rule out the Hawthorne effect (cause 3), which states that merely focusing on a team improves their productivity no matter what else is changed. But statements from developers suggest that the primary and derived effects of the techniques (cause 1 and 2) are the main causes for the reduction in error reports.

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 reduced uncertainty and indecision among the developers on what features to include and how they should be implemented and work. As one developer said during an interview:

“After the scenarios had been written they were often used in design discussions in the project group. The one that was best at relating to the scenarios won the discussions.”

Another developer mentioned that this was the first project he had experienced without turbulence from uncertain requirements.

8. Business Benefits

We have seen a significant reduction in error reports due to the scenario and usability test techniques. 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 user interaction with the product was totally changed as a result of the usability tests. The first prototype raised incredibly many issues, especially compared to how little time it took to develop. A completely revised second version of the prototype still raised many issues during usability tests, so a third version of the prototype was made. This acted as a validation of the changes that had been made to the user interface.

When Brüel & Kjær experts were presented with the prototype, the response was: “That is really smart, why haven't we done that before, why haven't we focused on getting the measurements in the box instead

The product was released in December 1997 just in time for orders to be shipped before the end of our financial year. It is the opinion of the project team that it would have been impossible to achieve this goal, if they had not used the new techniques. Too many problems would have been discovered too late.

The product has now been on the market for more than 7 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.

9. General Observations

Analysis of error reports is a cheap and effective way for companies who wish to get started on a software quality process improvement programme. It is not necessary to perform comprehensive measurements on development activities, and wait until enough data has been collected.

We have found that using the already available information on bugs in the company has provided us with enough information to create real attention with management to initiate change programmes even before the effects of the proposed changes were substantiated.

Setting up a process improvement programme is now an experience-driven incremental task where measurements are only performed when experience shows that there is a real need (problem) for the data to make an informed decision on how to change part of the development process.

This approach to process improvement will guarantee constant management attention because of immediate results, and acceptance among developers since only important measurements need be collected by them.

10. Acknowledgements

Our SPI projects have been funded as Process Improvement Experiments by the Commission of the European Communities (CEC) under the ESSI programme: European System and Software Initiative. The goal of the ESSI programme is to promote improvements in the software development industry so as to achieve greater efficiency, higher quality, and greater economy. This is accomplished by applying state-of-the-art in software engineering in a wide range of industries.

We also want to acknowledge persons at Brüel & Kjær Kai Ormstrup Jensen (now with DELTA) and Per-Michael Poulsen (now with NOKIA Telecommunications), who have been deeply involved in the analysis, definition, and introduction of the prevention techniques. And the project manager of the baseline project Flemming Petersen, who accepted the challenge to experiment with the new techniques, and stood by his decision when schedule pressure increased.

11. References

- [1] ANSI/IEEE, Guide to Software Requirements Specifications, ANSI/IEEE Std. 830, 1984.
- [2] Boris Beizer, Software Testing Techniques. Second Edition, Van Nostrand Reinhold NY, 1990.
- [3] A.M. Davis, Software Requirements, Analysis and Specification, Prentice-Hall, 1990
- [4] Otto Vinter, Per-Michael Poulsen, Knud Nissen, Jørn Mørk Thomsen, The Prevention of Errors through Experience-Driven Test Efforts. ESSI Project 10438. Final Report,"Brüel & KjærA/S, DK-2850 Nærum, Denmark, 1996. (<http://www.esi.es/ESSI/Reports/All/10438>).
- [5] Otto Vinter, Per-Michael Poulsen, Knud Nissen, Jørn Mørk Thomsen, Ole Andersen, The Prevention of Errors through Experience-Driven Test Efforts,"DLT Report D-259, DELTA, DK-2970 Horsholm, Denmark, 1996.
- [6] Otto Vinter, Søren Lauesen, Jan Pries-Heje, 'A Methodology for Preventing Requirements Issues from Becoming Defects. ESSI Project 21167. Final Report,"Brüel & KjærSound & Vibration

Measurement A/S, DK-2850 Nærum, Denmark, 1998. (<http://www.esi.es/ESSI/Reports/All/21167>)
(To appear when approved by CEC).

Generating Test Cases From Use Cases Automatically

Robert M. Poston



Test Cases from Use Cases - 1



Test Cases from Use Cases

- ◆ Purpose: To describe the specification-based testing process of generating test cases from use cases
- ◆ Outline:
 - ◆ Testing in the life cycle process
 - ◆ Back end testing
 - ◆ Specification-based testing
 - * Front-end defect prevention
 - * Requirements in text
 - * Requirements in use cases
 - ◆ Generating test cases from use cases



Test Cases from Use Cases - 2



Testing in Life Cycle Models

- ◆ All contemporary life cycle models allocate **30% or more** of project time and resources to testing.¹

Requirements Process		Design Process	Implementation Process			Testing Process
10% Analyze Need/ Problem	10% Specify Rqmts.	20% Design Application	10% Design Classes	10% Code Classes	10% Test Classes	30% Test Application



Test Cases from Use Cases - 3



Back End Testing

- ◆ Traditionally testing is *back end* work.
- ◆ The testing phase includes two kinds of work:
 - ◆ Testing - Failure detection and risk determination
 - ◆ Reworking - Failure elimination

						Testing Process
Analyze Need/ Problem	Specify Rqmts.	Design Application	Design Classes	Code Classes	Test Classes	ReWorking (Debugging)
						Testing



Test Cases from Use Cases - 4



Specification-Based Testing

	Specification-Based Testing	Product-Based Testing
Test cases designed from	Specification or Model	Code
Earliest starting phase	<i>Front End</i>	Back End
May also be called	Black Box	Code-Based or White Box



Test Cases from Use Cases - 5



Automated Specification-Based Testing

- ◆ Automated specification-based testing reduces work time in two ways:
 - ◆ **Defect prevention**
increases quality and reduces rework
 - ◆ **Test automation**
increases productivity

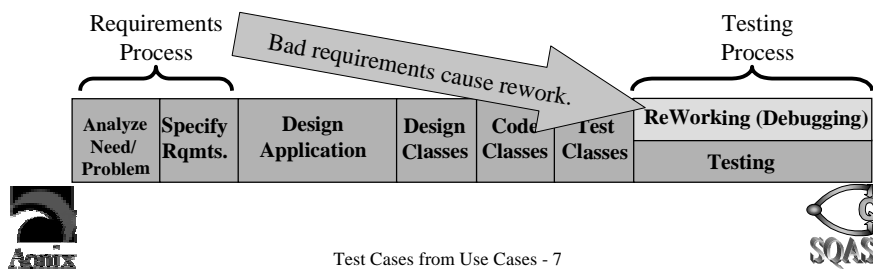


Test Cases from Use Cases - 6



Defect Prevention

- ◆ **55% or more** of software failures discovered by end users and system testers are caused by problems with requirements.²



Requirements Defects

- ◆ The most probable defects in requirements are well known:³
 - ◆ Ambiguous words and phrases
 - ◆ Incomplete statements
 - ◆ Inconsistent functions
 - ◆ Untestable functions
 - ◆ Untraceable functions
 - ◆ Undesirable design impositions



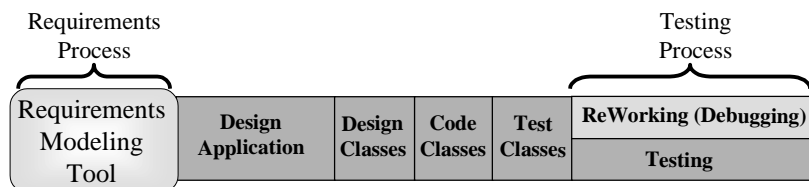
Preventing Requirements Defects

- ◆ The methods for preventing the most probable defects in requirements have been widely published. ^{4, 5, 6}
 - ◆ Rigorous definitions prevent ambiguities.
 - ◆ Usage checkers prevent inconsistencies.
 - ◆ Checklists prevent incompleteness.
 - ◆ Standards prevent testability problems.
 - ◆ Standards prevent traceability problems.
 - ◆ Rules prevent undesirable design impositions.



Preventing Requirements Defects

- ◆ Tools can incorporate all methods for preventing most probable defects in requirements.



Preventing Requirements Defects

- ◆ A requirements modeling tool must
 - ◆ model in a language that end users can understand with little training
 - ◆ capture all information a tester needs to produce test cases
 - ◆ prevent most probable defects



Test Cases from Use Cases - 11



Preventing Requirement Defects

- ◆ The Use Case Modeling Notation meets the three tool requirements when

Restricted to system-level models

System-level models do not show modules, subsystems, or internal objects.

Extended to enable test-ready models

Test-ready models contain sufficient information for test generation.

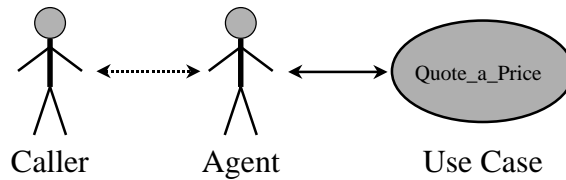


Test Cases from Use Cases - 12



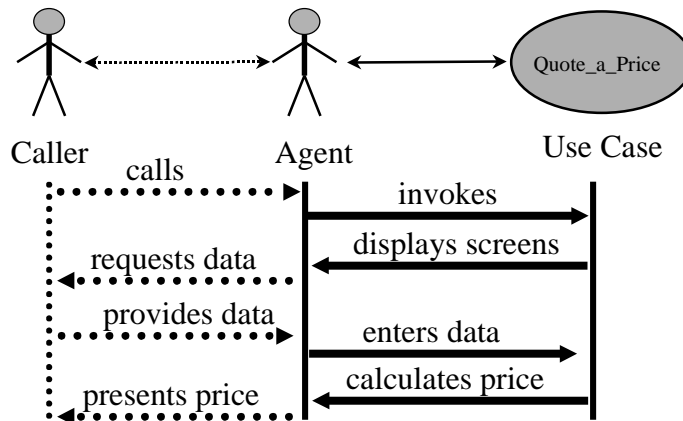
Requirements in Text

- ◆ Here is a natural language requirement.
 - ◆ The Customer Service System (CSS) shall help a Sales Agent calculate a price quote for a product requested by a Caller.
- ◆ With its graphical equivalent.



Requirements in Use Cases

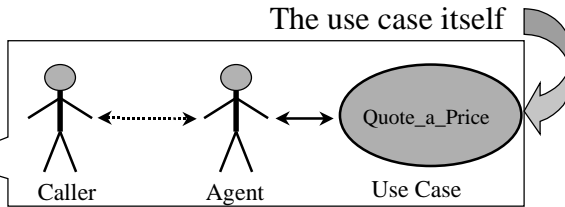
- ◆ Requirement: The Customer Service System shall help a Sales Agent calculate a price quote for a product requested by a Caller.



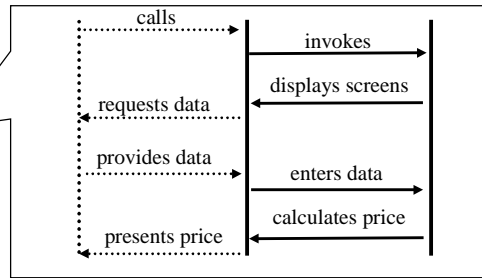
Use Case Model

A Use Case Model has three parts:

The Use Case Diagram



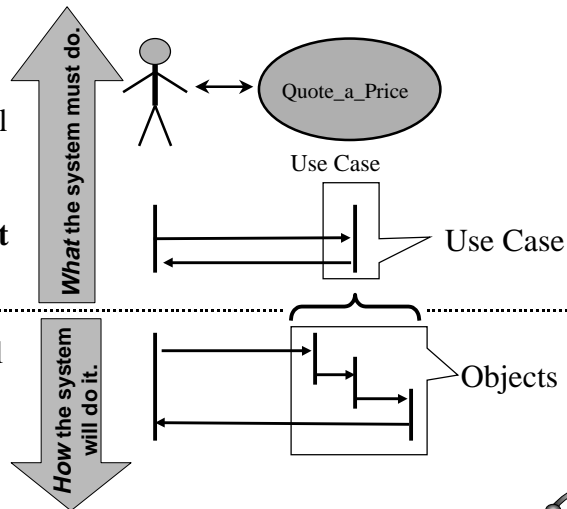
The Event Trace or Sequence Diagram



System-Level Use Case Model

System-Level Use Case = Requirement

Object-Level Use Case = Design

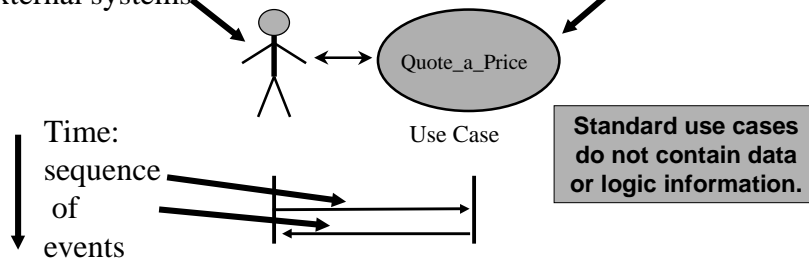


Test-Ready, System-Level Use Cases

Test-ready means that a use case contains sufficient information for test generation.

Actors: end users and external systems

Action

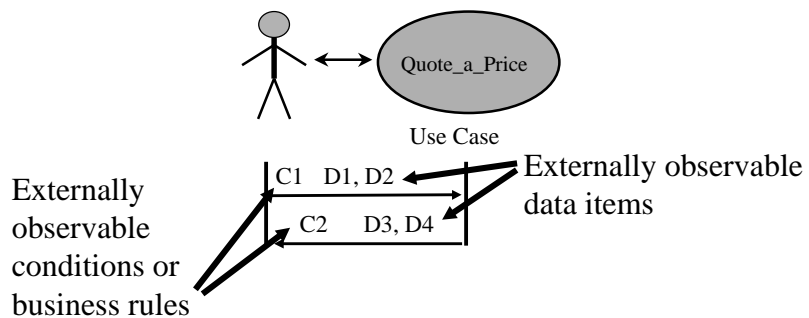


Test Cases from Use Cases - 17



Test-Ready, System-Level Use Cases

To be test-ready a model must contain actor, action, data, logic, and time information.



Test Cases from Use Cases - 18



Test-Ready, System-Level Use Cases

Example externally observable data items:

Text

Range - [A-Z] 1,36

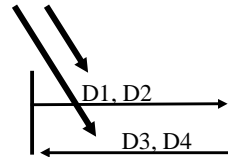
List - Jones, Smith, Won

Numeric

Range - min=19, max=65, res=1

List - 12, 14, 18, 21

Structures



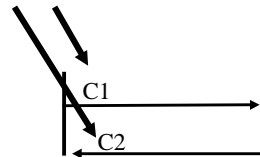
Test-Ready, System-Level Use Cases

Example externally observable conditions or business rules:

dataitem A == dataitem B

Deposit > 0

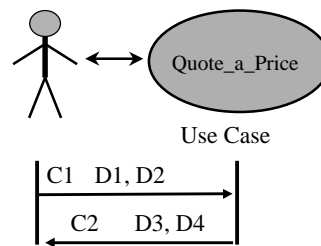
Year A > Year B



Specification-Based Testing

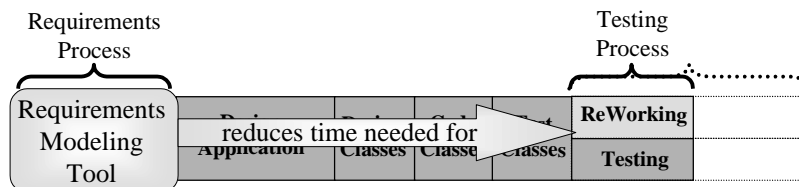
Specification-Based testing requires system-level, test-ready models which contain

actor,
action,
data,
logic,
and
time information.



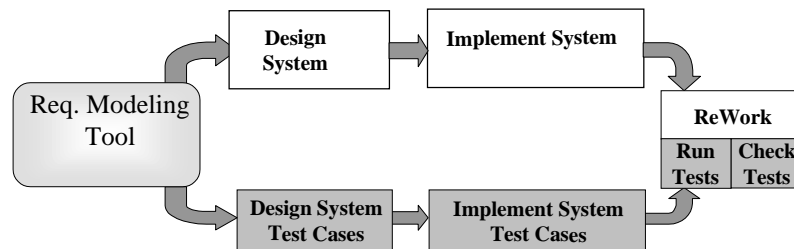
Preventing Requirements Defects

- ◆ Using a tool to create test-ready, system-level use cases in the requirements phase prevents requirements defects and reduces rework time.



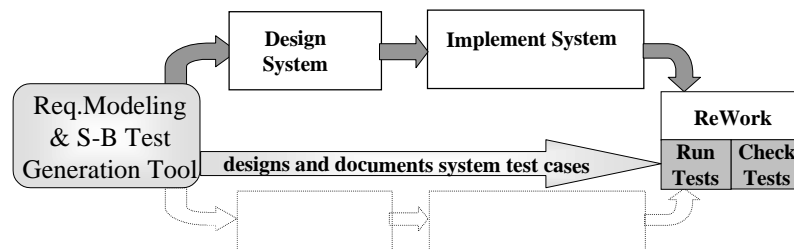
Front-End Testing

- ◆ Preventing defects in requirements makes requirements stable enough to begin testing earlier in the lifecycle.



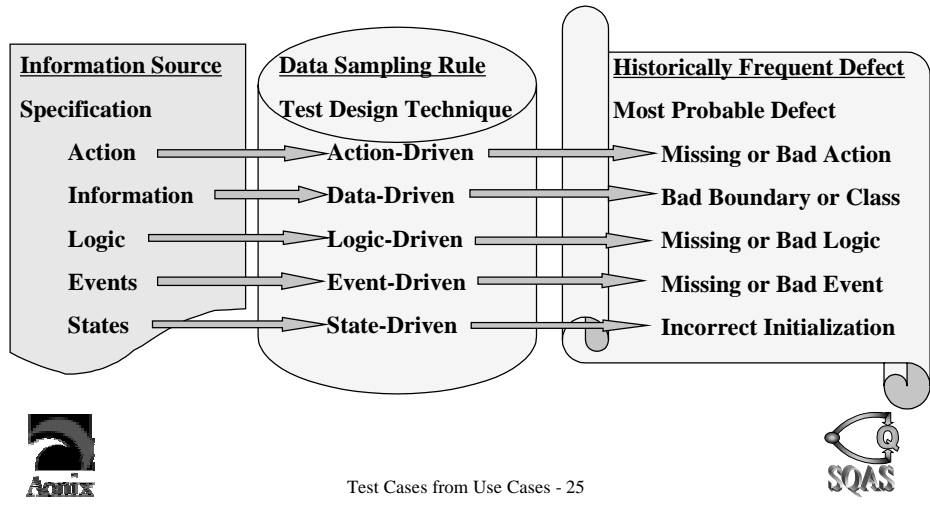
Test Generation from Use Cases

- ◆ A specification-based test generator produces test cases and reduces work.



Generating Test Cases from Use Cases

Select samples of stimuli that will find failures.



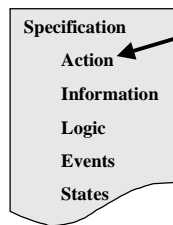
Test Case Design⁷

- ➔
- ◆ Test Design Step 1:
 - ◆ For each Requirement (Action)
 - ◆ *Sample* Stimuli (Causes or Inputs)
 - ◆ Test Design Step 2:
 - ◆ Combine *samples* to produce test cases



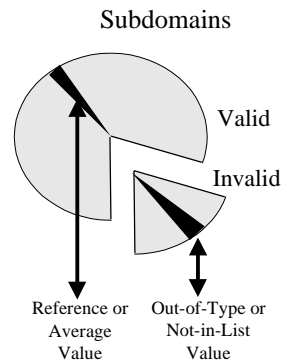
Action-Driven Test Design

Functional Testing



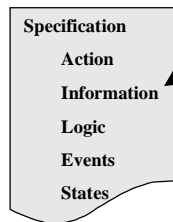
Rule: Select valid and invalid samples for each stimulus or input that is used by an action.

Reason: To find defects of incorrect and missing actions.



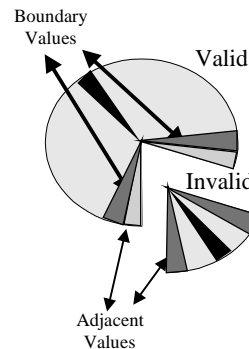
Data-Driven Test Design (1)

Boundary Value Analysis



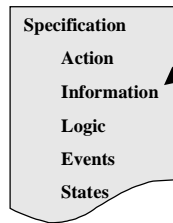
Rule: Select samples on boundaries and just below and just above the boundaries (first & last, fastest & slowest, highest & lowest).

Reason: To find defects of boundary handling, such as off-by-one errors.



Data-Driven Test Design (2)

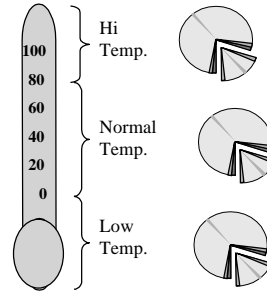
Equivalent Class Analysis



Rule: Select *one* member from a class of values when all members of the class receive *equivalent* treatment by the code.

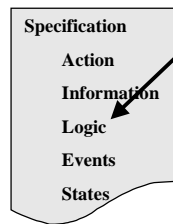
Reason: To find defects of incorrect class handling.

Temperature



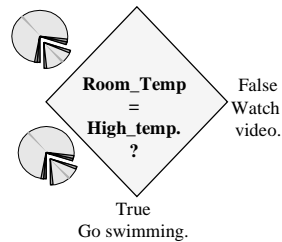
Logic-Driven Test Design

Cause-Effect Graphing aka Business Rule Testing



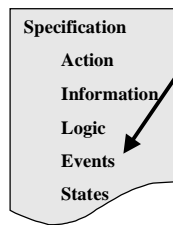
Rule: Select data samples that will cause each logical expression to be exercised with a true and a false evaluation.

Reason: To find defects of incorrect logic processing.



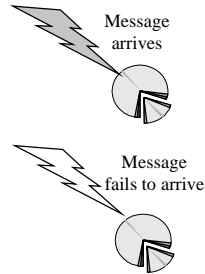
Event-Driven Test Design

Performance Testing



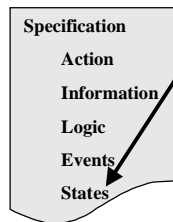
Rule: Sample each event as a Boolean item to indicate that the event has or has not occurred.

Reason: To find defects of incorrect time synchronization.



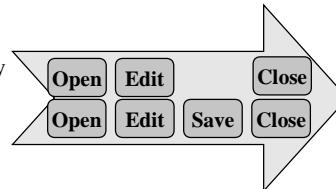
State-Driven Test Design

Initialization-Reinitialization Testing



Rule: Select input data samples that will cause each state transition to occur in every state where the transition should occur and in one state where the transition should not occur.

Reason: To find defects of incorrect action sequences.



Test Case Design

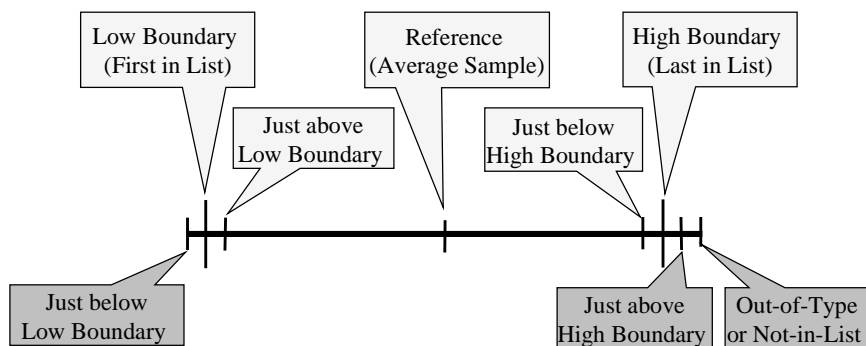
- ◆ Test Design Step 1:
 - ◆ For each Requirement (Action)
 - ◆ *Sample Stimuli* (Causes or Inputs)
- ◆ Test Design Step 2:
 - ◆ Combine *samples* to produce test cases



Test Design Step 2: Combine Samples

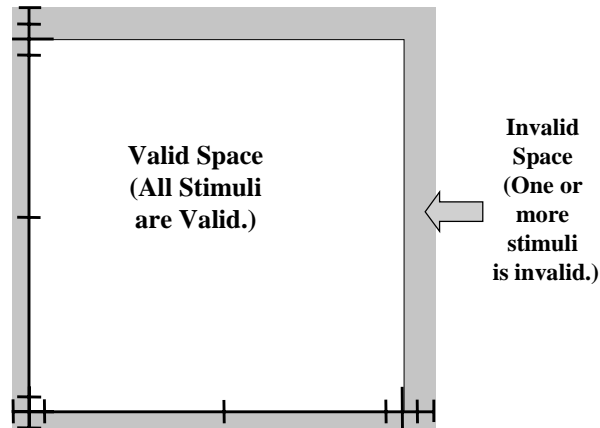
Construct the sample or testing space.

Start with a sample or number line.



Test Design Step 2: Combine Samples

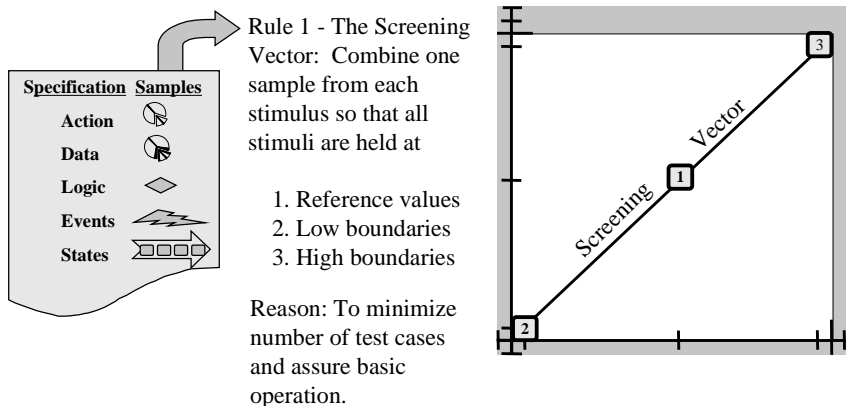
Combine the number lines to form the sample or *testing space*.



Test Cases from Use Cases - 35



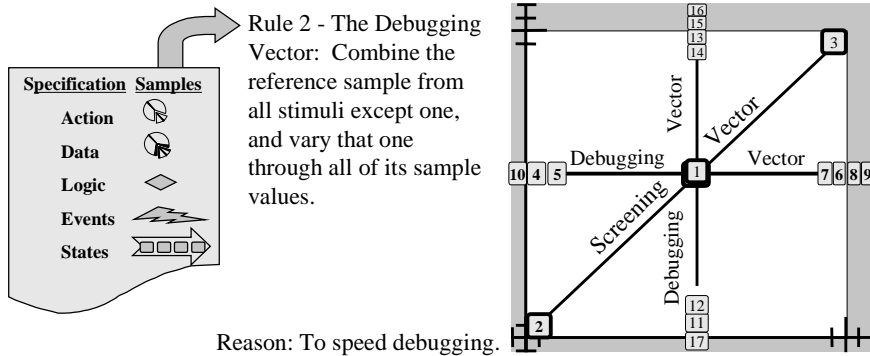
Test Design Step 2: Combine Samples



Test Cases from Use Cases - 36

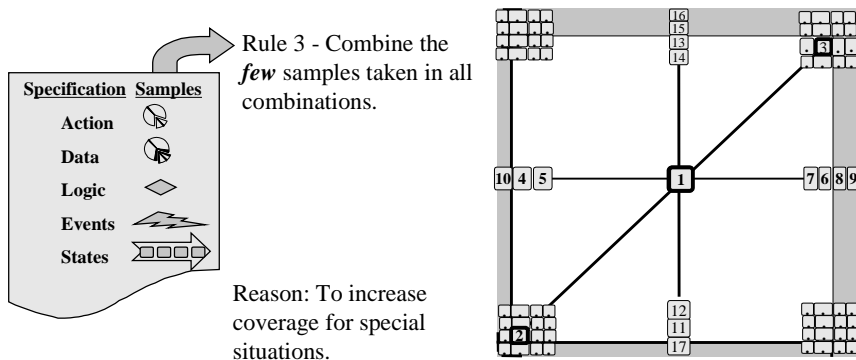


Test Design Step 2: Combine Samples



Test Design Step 2: Combine Samples

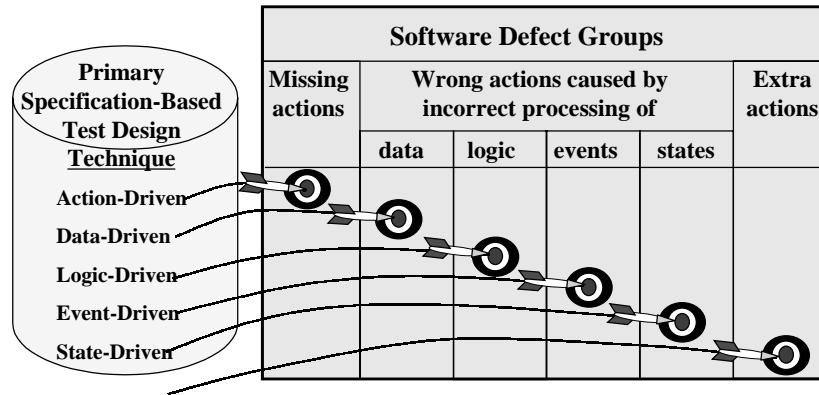
Robustness Testing



Note: Logic-, Event-, and State-Driven samples *MAY* add additional test cases.



Test Design Techniques and Defect Groups



Run specification-based test cases and measure code coverage. Any code not covered is extra.



Summary - Experiences

- ◆ Manual Specification-Based Testing: Case Study 1
Defect count dropped 94% from
1.2 failures per 1,000 lines of code to
0.072 failures per 1,000 lines of code. ⁸
- ◆ Automated Specification-Based Testing: Case Study 2
Productivity increased 100 fold from
100 test cases in 20 days to
1,000 test cases in 2 days. ⁹



References

- ¹ Jones, Capers, Software Quality: Analysis and Guidelines for Success, International Thompson Computer Press, Boston, MA, 1997, p. xxiv.
- ² Davis, Alan, "System Testing: Implications of Requirements Specifications," *Information and Software Technology*, Vol. 32, No. 6, July/August 1990, pp.407-414.
- ³ ANSI/IEEE Standard 830, **A Standard for Requirement Specifications**, IEEE, New York, NY, 1984, 1988, 1992 .
- ⁴ Ibid, ANSI/IEEE.
- ⁵ Ibid, Davis.
- ⁶ Poston, Robert, M., "Preventing The Most Probable Errors in Requirements," IEEE Software, September 1987, Vol. 4, Num. 5, pp. 86-88.
- ⁷ Poston, Robert M., Automating Specification-Based Software Testing, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 25.
- ⁸ Poston, Robert M. "Counting Down to Zero Software Failures," Automating Specification-Based Software Testing, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 230.
- ⁹ Adhikari, Richard, "Development Process is a Mixed-Bag Effort," Client/Server Computing, February 1996, pp. 65-72.



IBM



Off-The-Shelf Vs. Custom Made Coverage Models, Which Is The One for You?

Shmuel Ur, Avi Ziv

Yael Shaham-Gafni - Presenter

IBM Research Lab in Haifa, Israel

email: {sur, aziv,gafni@vnet.ibm.com}

November 1998



Outline

- ➡ Introduction
- ➡ Benefits and risks of using coverage
- ➡ Code coverage
- ➡ Functional coverage
- ➡ Comparison between code and functional coverage
- ➡ Comet - general purpose tool for functional coverage
- ➡ General guidelines for coverage usage
- ➡ Conclusions



What Is Coverage?

Coverage:

Any metric of completeness with respect to a test selection criteria

☞ Testing is based on samples

☞ Definitions:

- **Coverage Task** - A boolean predicate on a test
- **Coverage Model** - A set of coverage tasks
- **Complete Coverage** - Covering all the (legal) tasks in that model



Benefits of Using Coverage

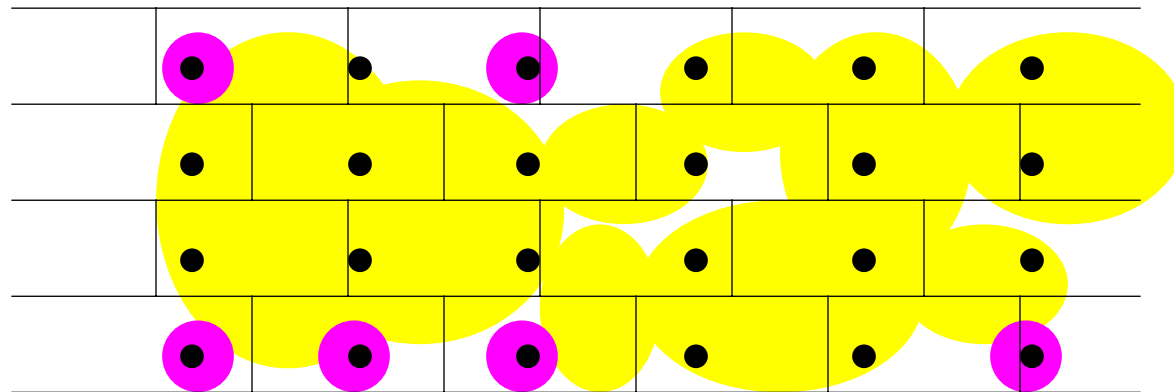


-
- ➔ Measure the “quality” of a set of tests
 - ➔ Supplement test specification by pointing to untested areas
 - ➔ Help in creating compact and comprehensive regression suites
 - ➔ Help in definitions of testing requirements and specifications
 - ➔ Enable better understanding of tested program



Risks of Using Coverage

- Using coverage without commitment to use results
- Generating simple tests to cover specific uncovered tasks
 - ➔ The painted wall analogy



- Some coverage models are ill suited to deal with common problems
 - ➔ Missing code
- Low coverage goals



Code Coverage

-
- Measures the execution of tests against the source code of the program
 - Required in many standards
 - Easy to use
 - Many available tools
 - ➔ At least one tool for almost every language and OS
 - Many existing models
 - ➔ Control flow models - statement, branch, multi condition, ...
 - ➔ Fault-based models - mutations
 - ➔ Dataflow models - C-use, P-use, ...
 - Measure coverage uniformly on the whole program

Functional Coverage



-
- ☞ Coverage is based on the function of the design

 - ☞ Coverage models are specific to a given design

 - ☞ Models cover
 - The inputs
 - Internal states
 - Scenarios
 - Parallel properties
 - Bug Models

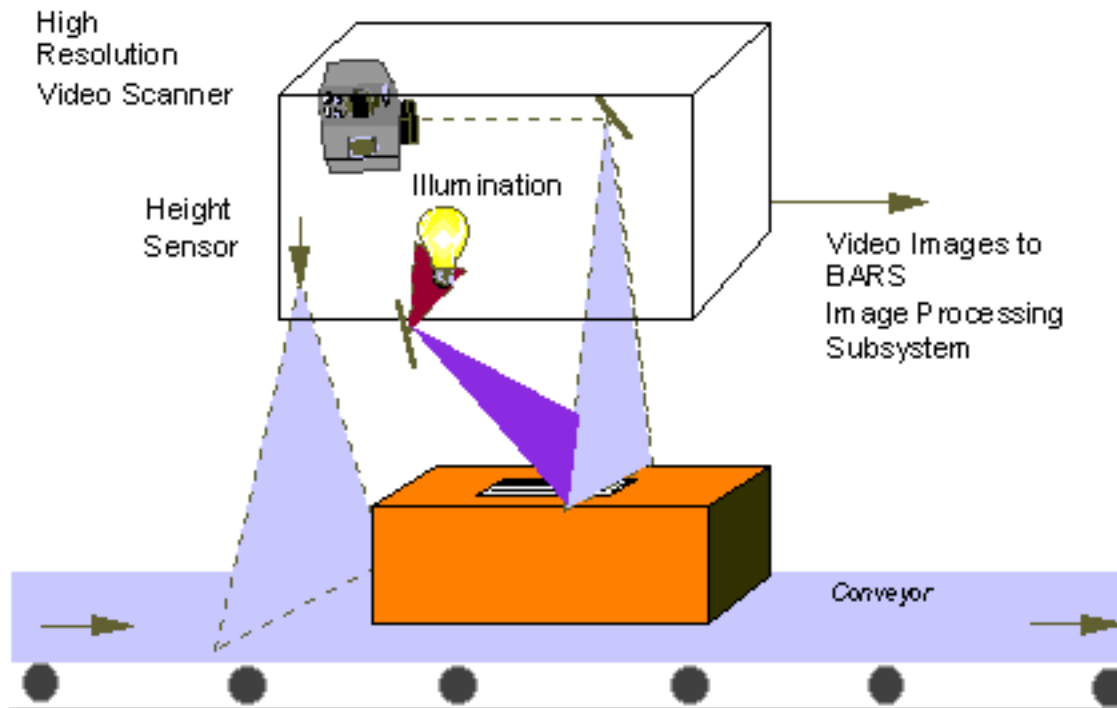


The Functional Coverage Process

- **Defining the domains of coverage**
 - Where do we want to measure coverage
 - What attributes (variables) to put in the trace
- **Defining models**
 - Defining tuples and semantics on the tuples
 - Restrictions on legal tasks
- **Collecting data**
 - Creating traces
 - Processing the traces to measure coverage
- **Coverage analysis and feedback**
 - Monitoring progress and detecting holes
 - Refining the coverage models
 - Generating regression suites



Functional Coverage Example - Parcel Sorting System





Parcel Sorting System - Model Definition

Attribute description and values:

- Height - (Height of parcel in millimeters)/100+1 (max 6)
- Width - (Width of parcel in millimeters)/100+1 (max 6)
- Length - (Length of parcel in millimeters)/100+1 (max 6)
- Master - 1 if parcel processed by master else 0
- Num_processors - Number of busy processors when the parcel started (1-6)
- Num_addresses - Number of addresses are on the parcel (0-4)
- Flush - Name of the stage prior to Flush
- Finish - Name of the stage prior to Finish
- Success - confidence above 0.8 in recognition

Restrictions:

- There are at most six busy processors at a time
- If the parcel was sent to the master all slaves were busy



Code vs. Functional Coverage

☞ Availability:

- **Code:** many available commercial tools
- **Functional:** almost no commercial tools available

☞ Cost:

- **Code:** cost of tools is usually low, no additional costs
- **Functional:** higher cost to develop tools and models

☞ Ease of use:

- **Code:** easy to use. Plug and play
- **Functional:** need expertise in definition and implementation of models



Code vs. Functional Coverage

👉 Learning curve:

- **Code:** users can benefit from tool almost immediately
- **Functional:** need some time to learn how to define and implement models

👉 Experience gathered:

- **Code:** vast experience on usage, coverage targets, etc.
- **Functional:** little experience, harder to reflect from one project to another



Code vs. Functional Coverage

☞ Focus:

- **Code:** Uniformly spread on all the entire program
 - + Covers the entire program
 - Cannot focus on area of concern
- **Functional:** Focuses on areas of concern
 - + Check coverage where most important
 - Impossible to use on the whole program

☞ Adapting to testing resources:

- **Code:** Hard to adapt because of hard coded models
- **Functional:** Easy to adapt. Can use right number of models and of the appropriate complexity



Code vs. Functional Coverage

☞ Missing code:

- **Code:** Not fitted to deal with missing code
- **Functional:** Can deal with missing code

☞ Impact on project:

- **Code:** Impact only on testing and testing requirements
- **Functional:** Many areas of impact
 - Better understanding of program and environment
 - Helps in test specification and requirements
 - Impacts from early stages of design



Which Coverage Method to Use

☞ Functional Coverage Should be used on:

- High risk area
- Complex, error prone areas
- Changes to existing code (maintenance)

☞ Code Coverage Should be used:

- On all code written given sufficient resources are available
- As a criteria for finishing unit testing



Comet -

General Purpose Tool for Functional Coverage

Objective:

*A generic tool for **measuring coverage** with respect to user-defined functional coverage models*

Inputs:

- Coverage models (events, relations)
- Event traces (e.g., test programs, simulation traces)

Outputs:

- Coverage reports
 - Coverage statistics: what is covered
 - Coverage progress
- Regression test suite with coverage properties



Comet Methodology

☞ Emphasis on Functional Coverage

- The coverage is on the function, not the program
- The models can be as focused and detailed as desired

☞ Separate coverage models from coverage measurement tool

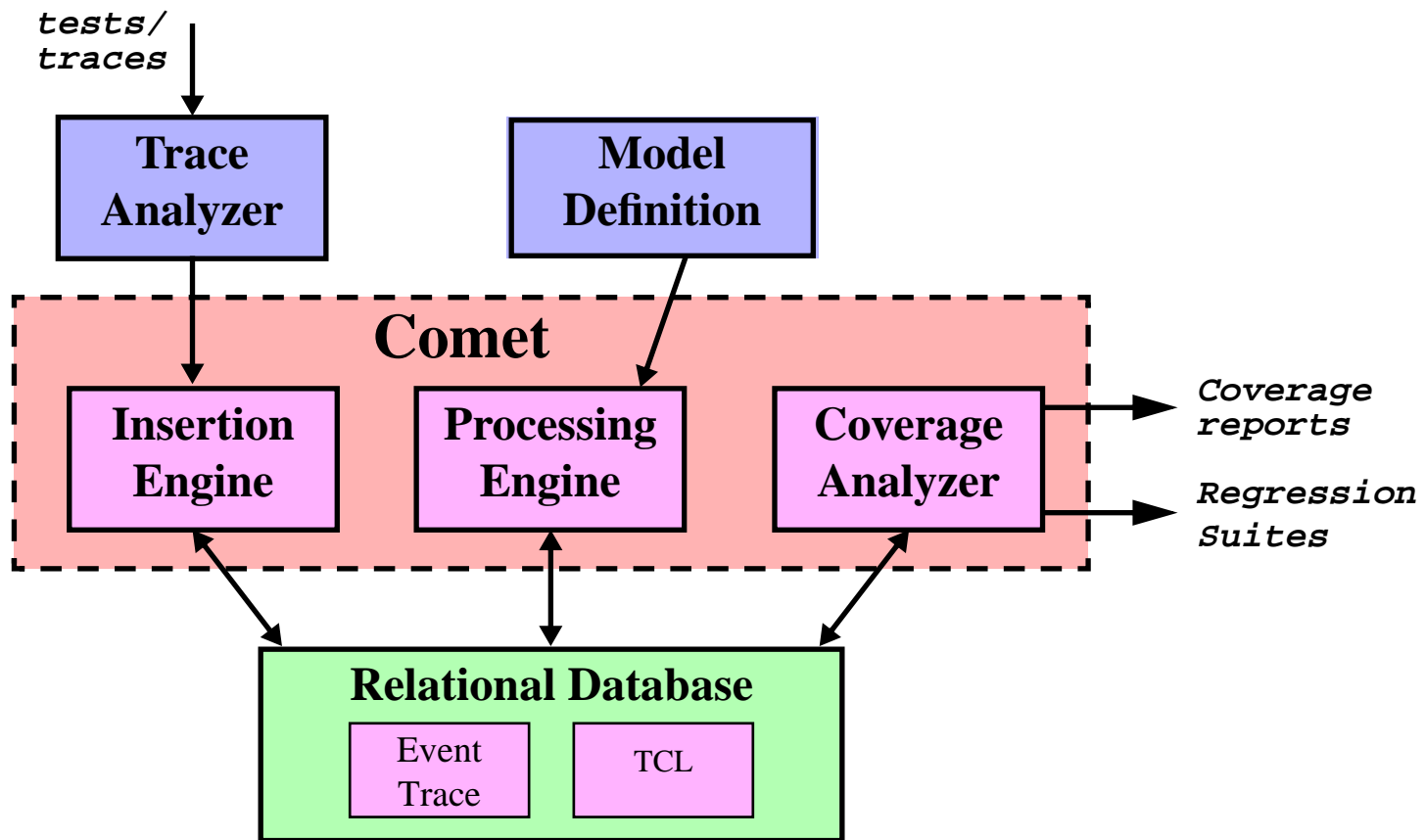
- The same tool supports many different coverage models

☞ User defines coverage models as a part of the test plan

- Less test requirements to write by hand



Comet - Overall Structure





Coverage Guidelines

- ➡ Look for the most complex, error prone part of the application
- ➡ Create the coverage models at high level design. This will improve the understanding of the design and automate some of the test plan.
- ➡ Create the coverage model hierarchically. Start with small simple models and combine them to create larger models.
- ➡ Before you measure coverage check that your rules are correct on some sample tests.
- ➡ Try to generalize as much as possible from the data; X was never 3 is much more useful than the task (3,5,1,2,2,2,4,5) was never covered.



Conclusions

- ➡ Code coverage is easier to use, cheaper and more available than functional coverage
- ➡ Functional coverage is an excellent way to improve the testing of risky, error prone areas
- ➡ Functional Coverage can help in all design stages. Functional coverage models should be created at high level design
- ➡ Combining Code and functional coverage can lead to high quality verification.
- ➡ For more information
<http://oop.cs.technion.ac.il/236804-Fall-1997/papers/pub.html>

Off-The-Shelf Vs. Custom Made Coverage Models, Which Is The One for You?

Shmuel Ur and Avi Ziv
IBM Haifa Research Lab
MATAM
Haifa 31905, Israel
email: {sur, aziv@vnet.ibm.com}

Abstract

We compare the benefits and costs of using off-the-shelf coverage tools vs. application specific coverage tools. We start with an overview of coverage, its benefits and its risks. We elaborate on relatively unfamiliar functional coverage as a way to create custom made coverage models. We explain what functional coverage models are, how to create them and their great benefits with an example from our experience. We provide guidelines on how to decide if coverage should be used at all and whether code based or functional coverage (or both) should be used.

1 Introduction

Testing is one of the biggest problems of the software industry. The cost of testing is usually between 40-80% of the development process (70% for Microsoft) as compared with less than 20% for the coding itself [4]. The practice of letting the users find the bugs and fixing them in the next release is becoming dangerous and costly for three main reasons: reputation and brand-name are harmed, replacing the software can be very costly when there is a large install base, and litigation can be expected if the software error caused harm to the user. Therefore, one has to be certain that testing resources are used efficiently and that the testing is thorough.

The main technique for demonstrating that the testing has been thorough is called test coverage analysis [10]. Simply stated, the idea is to create, in some systematic fashion, a large and comprehensive list of tasks and check that each task is covered in the testing phase. Coverage can help in monitoring the quality of testing, assist in creating tests for areas that have not been tested before, and help with forming small yet comprehensive regression suites [5].

Coverage, in general, can be divided into two types: code or functional. Code coverage concentrates on measuring syntactic properties in the execution, for example, that each statement was executed, or each branch was taken. This makes code coverage a generic method which is usually easy to measure, and for which many tools are available. Examples include code coverage tools for C [9], C++ [13], and Java [14]. Functional coverage, on the other hand, focuses on the functionality of the program, and it is used to check that every aspect of the functionality is tested. Therefore, functional coverage is design and implementation specific, and is more costly to measure. Currently, functional coverage is usually implemented manually or by using custom made tools.

In this paper, we describe functional coverage and how it should be used in the testing process, starting from test requirements and specifications, and going all the way until and beyond the release of the product. We discuss the advantages and disadvantages of functional coverage and compare it to code coverage. Based on this comparison, we provide guidelines on which type of coverage best fits the user's needs and how it should be used.

Since functional coverage models are derived from the specifics of the program specifications and implementation, it is impossible to find off-the-shelf tools that fit the user's coverage models. Moreover, the development of tools for specific models is usually too expensive and time consuming. The result is

that functional coverage tools are usually simple and do not have the rich functionality of code coverage tools. To overcome this problem, we developed, at IBM's Haifa Research Laboratory a new functional coverage tool that separates the coverage models which are defined by the user, from the tool itself. The user can define custom made coverage models that fit the design in the best way, while enjoying all the benefits of a coverage tool, such as data collection and processing, creation of coverage reports, and generation of regression suites with high coverage.

The rest of the paper is organized as follows: In Section 2, we provide a short background on what coverage is and what the benefits and risks of using it are. In Sections 3 and 4, we describe code and functional coverage. In Section 5, we describe Comet, a coverage tool that was developed at IBM's Haifa Research Laboratory. Comet enables the user to define her own coverage models and still enjoy the rich functionality of an off-the-shelf coverage tool. In Section 6, we compare the two coverage methodologies and provide some guidelines on who should use each methodology and how to use it. In Section 7, we provide some general guidelines for the use of coverage. Section 8 concludes the paper.

2 The Benefits and Risks in Using Coverage

Coverage is defined as any metric of completeness with respect to a test selection criteria [3]. Many such metrics have been suggested in the past [3], of which statement coverage is the most common. Full statement coverage means that every statement in the program has been executed by the tests. Coverage is one of the more systematic ways to check that the testing has been thorough. When using any coverage model, of which many are available [11], a metric is created against which the quality and completeness of the testing is measured.

The most commonly used coverage metrics are based on the control flow of the program, such as statement coverage and branch coverage, however, many other metrics exist. Some coverage metrics are based on the data flow of variables, like define-use [3], while others are not based on the program code but on the inputs or the specifications.

Coverage is usually used to find new testing requirements that have been overlooked in the test plan. Many times the test requirements are written during the design and do not take into account the details of the implementation. For example, the implementation of a sorting function might use two different algorithms, depending on the size of the array sorted, a detail which is not in the specifications. In this case, statement coverage might show that the inputs never included the case of a short array, that use one of the algorithms, and that a new test is needed. Working with coverage as a guide to improve the quality of testing has been shown to be a cost effective use of resources [12].

Another application of coverage that is commonly used, is generation of regression suites [10]. Generation of regression suites has to deal with two contradictory requirements; the suite must be small so that it is economical to execute it after every design change, yet it must be comprehensive in order to find the bugs that were introduced. Coverage enables us to find a relatively small set of tests which is comprehensive in the sense that it covers the required metric [5].

Besides these uses, coverage provides other benefits to the testing process that are often overlooked. One such benefit is the use of coverage or, more specifically, functional coverage, to assist in defining testing requirements and specifications. Another benefit of functional coverage is that it helps to achieve a better understanding of the tested program during definition of the coverage models.

While the use of coverage as an aid to the testing process has a lot of benefits, centering the testing process around coverage has its own risks. A common misconception about coverage is that the testing methodology should be to decide on an appropriate coverage metric and then generate a set of tests that covers it. This is not advisable for a number of reasons, the main one is demonstrated by the following analogy to wall painting.

Assume that one uses a process which paints a wall with an inch of protective paint. A quality assurance process (coverage) chooses a hundred points at random at which to drill and measure

the paint thickness. If at all of the points there is an inch of paint, then we have a reasonably good assurance that the wall is well painted. However, what do we do if at 50 points the layer of painting is not thick enough? If we tell the painter the location of these 50 points and he fixes them, the fact that the 100 points are now covered no longer guarantees that the wall is covered because our painting (test generation) process and our coverage process are not independent. One way to overcome this difficulty is to inform the painter that some region of the wall is not well painted. This is equivalent in testing to generating a test requirement (as opposed to a specific test) from an uncovered point.

Another drawback of coverage is that many coverage models are ill suited to deal with many common problems. For example, control flow models, such as statement and branch coverage, are ill suited to deal with missing code. If, for example, a case statement should have six cases but, in practice, it has only four, statement or branch coverage will not help you find it. One way to overcome this difficulty is to use several coverage models, which are derived from different domains, so that one model will cover the weaknesses of another model.

A different risk in using coverage is setting low coverage goals. It has been shown that using coverage to assess quality with a lower coverage target (50%-90%) is not useful [12]. The reason is that the probability of having bugs in hard-to-cover areas tends to be larger than the probability of bugs in well covered areas. Therefore, it is better to use simpler coverage models with high coverage goals than more complex models with lower coverage goals.

3 Code Coverage

Code coverage, usually just called coverage, is a technique that measures the execution of tests against the source code of the program. For example, one can measure whether all the statements of the program have been executed. The main uses of code coverage are assessing the quality of the testing, finding missing requirements in the test plan and constructing regression suites.

A number of standards, as well as internal company policies, require the testing program to achieve some level of coverage, under some model. For example, one of the requirements of the DOA standard [15] is 100% statement coverage.

Many coverage tools that support all major programming languages exist. Every tool implements a number of coverage models for a particular combination of operating system, compiler and programming language. Most of them work by instrumenting the source code and adding counters which can later be used by the tool's user interface to show the status and progress of the coverage in some detail. To apply such a tool, one typically has to recompile the software with the tool and execute the tests. After the tests are executed, there is usually some interface that highlights the parts of the program that were not covered.

Almost all coverage tools implement the statement and branch coverage models. Multi-condition coverage, a model that checks that each part of a condition (e.g. A or B and C) had impact, is also implemented by many tools. Fewer tools implement the more complex models such as define-use, mutation, and path coverage variants [6].

The main advantage of code coverage tools is their simplicity of use. The tools come ready for the testing environment. No special preparations are needed in the programs and understanding the feedback from the tool is straightforward. The main disadvantage of code coverage tools is that the tools do not "understand" the application domain. Therefore, it is very hard to tune the tools to areas which the user thinks are of significant.

4 Functional Coverage

Unlike code coverage, where the execution of tests is measured against the program source code, functional coverage focuses on the functionality of the program, and it is used to check that every aspect of

the functionality is tested. Therefore, functional coverage is design and implementation specific, and is harder to measure. Currently, functional coverage is mostly done manually.

Functional coverage is considered by some to be black-box testing [6], since it involves models based on the specifications of the application. We believe that functional coverage is much more varied. Functional coverage models can be based on the specifications of the application, but they can also be derived from the implementation. Functional coverage models have many flavors. Models can cover the inputs and outputs of the program or they can look at the internal state of the program (e.g., values of variables). Functional coverage models can be snapshot models, that look at the state of the program at a certain time, or they can be temporal models that deal with scenarios. Usually, functional coverage models involve looking at several properties in parallel. Our experience shows that many bugs can be found only when a number of events happen concurrently [1]. Therefore, covering each event on its own is not sufficient. A simple example for a snapshot model is covering all the possible values of the input parameters of a function. An example for a temporal model is looking at the changes in the values of global variables between consecutive activations of a function. Thread interleaving and synchronization in a multi-threaded system is a source of many bugs. Therefore, a coverage model that looks at all the reasons for thread switching is a good example for a coverage model that is based on a bug model (A bug model is a set of requirements for finding bugs of a type that have been uncovered before.)

The first and most important step in the functional coverage process is deciding what to cover or, more precisely, on what coverage models to measure coverage. In order to make coverage successful and use it to influence the testing process, it is important to choose the correct types of coverage models. First, it is important to choose coverage models for areas which the user thinks are risky or error prone. Next, the size of the model should be chosen in accordance with the testing resources. The model size (number of tasks) should not be too large, making it impossible to cover the model, given the testing time and resources available. From our experience, we found that the best way to create effective models is to start from small models and later refine them or combine them to create bigger and more complex models.

Often, some of the tasks in the coverage model are *illegal tasks*, that is, coverage tasks that should not occur. The reasons could be limitation on the inputs - the sum of angles of a triangle is 180 - or implementation details - two threads that write to the same resource should never be in the write stage of the semaphore at the same time. Specifying the illegal tasks is an important part of coverage model definition.

After definition of the coverage models, the next step is data collection. There are two broad categories of coverage data collection techniques. The first, which is used in most of the code coverage tools, is to create a counter for each task and modify it on the fly whenever the task is found. The second is to print the necessary data to a trace file that is later processed.

One of the historical problems of functional coverage, which stems from the fact that the models are implementation specific, is lack of automation. However, although each model is unique, the coverage processes of different models have much in common. Tasks have to be updated in tables, regression suites have to be created, coverage reports on sub-models and on progress have to be made. We have created a coverage tool, named Comet [7], that handles all the common requirements, and we have created over a hundred coverage models for a number of customers for widely varying applications (mainly in hardware). A more detailed description of Comet and the methodology behind it is given in Section 5.

The ability to focus on points of concern, which is one of the main advantages of functional coverage, carries a risk. The risk is that only functional coverage will be used, and therefore, *only* the parts which are of concern will be tested thoroughly. Since creation of coverage models requires some effort, the parts which are not of special concern could be neglected. For these parts, it is better to use generic, off-the-shelf, code coverage models than nothing at all.

4.1 Example - Parcel Sorting System

We demonstrate applying a functional coverage to a parcel sorting system that was developed at IBM's

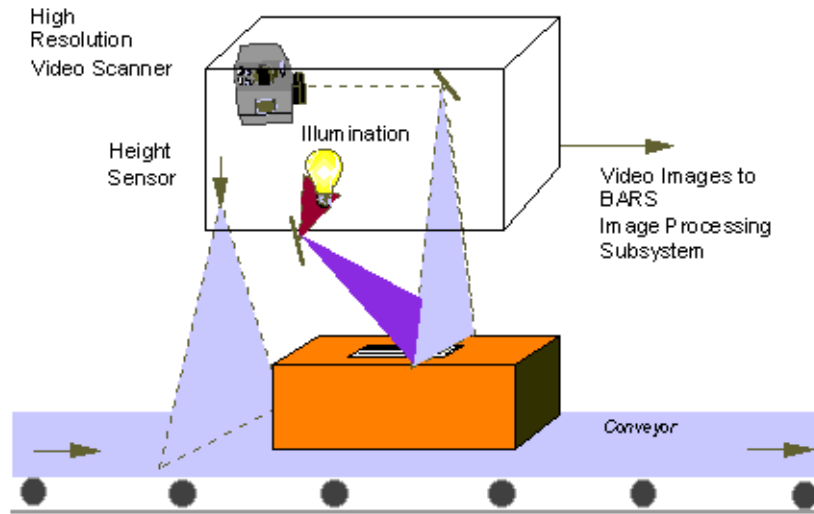


Figure 1. Structure of the parcel sorting system

Haifa Research Laboratory. In this system, a 6 PowerPC SP system (a multi processor system) is used to sort parcels. The parcels are sorted in a number of phases. In the first phase, the top surface of the parcel is scanned, its dimensions are measured, and the lenses are focused. Then, a processor is assigned to the parcel. In the second stage, a high resolution image of the top surface is created, the image is decoded so that the address and bar-code on it are understood. In the third stage, the weight of the parcel is acquired. The information including the size, weight and address is then delivered to the host. The parcel sorting system is depicted in Figure 1. This system contains many technologies, such as voting between two powerful OCR engines, fuzzy dictionary search for address validation, parallel processing and leading edge image data transfer (50MB/sec).

The coverage model described below is designed to check that the system has been stressed by different inputs. The model checks that every type of parcel size and every type of address, is encountered. The model verifies that a parcel has been internally processed in every way and by every processor, and that the processing can be correctly interrupted at any stage. The correlation between success (correct decoding of addresses), parcel type, and system load is measured.

A coverage model is composed of a list of tasks. A convenient way to specify tasks is as tuples of variables or attributes. The list of tasks in the coverage model is the cross product of the lists of all the possible values for each attribute. For example, the coverage model that we illustrate here, has attributes of a single parcel and the load of the system during the processing of that parcel. Figure 2 gives the list and description of the attributes for the parcel sorting system model. Some of these attributes, such as `Num_addresses`, are values of variables in the program. Some, like `Width`, are calculated from a single variables, and others, such as `Num_processors` and `Flush`, require processing of the trace. Not all the tuple values are possible. Restrictions on the possible tuples are imposed by limitations on the inputs caused by the environment, or by the implementation. Examples of restrictions for the parcel sorting model are:

- There are no more than six busy processors at one time
- When a parcel is sent to the master, all the slaves are busy.

Note that the process of finding restrictions is an iterative process. An initial set of restrictions is

Attribute list:

(Height, Width, Length, Master, Num_processors, Num_addresses, Flush, Finish, Success)

Attribute description and values:

- Height - (Height of parcel in milimeters)/100+1
- Width - (Width of parcel in milimeters)/100+1
- Length - (Length of parcel in milimeters)/100+1
- Master - 1 if parcel processed by master else 0
- Num_processors - Number of busy processors when the parcel started (1-6)
- Num_addresses - Number of addresses are on the parcel (0-4)
- Flush - Name of the stage prior to Flush
- Finish - Name of the stage prior to Finish
- Success - confidence above 0.8 in recognition

Figure 2. Attribute list for parcel sorting system model

Model 1:Parcel Size - (Height, Width, Length)
Model 2:Number of addresses - (Num_addresses)
Model 3:Load - (Master, Num_processors)
Model 4:Messages - (Flush, Finish)
Model 5:Success - (Success)
Model 6:All Parcel types - Model 1 + Model 2
Model 7:Load and Messages - Model 3 + Model 4
Model 8:Messages and Success - Model 4 + Model 5
Model 9:Complete model - Model 6 + Model 7 + Model 5

Figure 3. Models hierarchy for parcel sorting system

generated. After looking at the traces, exceptions to some of the restrictions are found. Each exception signals a bug in the implementation, a problem in the trace generation, or an incorrect restriction, all of which happen in practice. Also, holes in the coverage measurement that are found may indicate restrictions that were previously overlooked.

Functional coverage models are usually built in a hierarchical way, one on top of each other. We start with small models that examine specific areas, such as the dimensions of a package in the parcel system. Later on, these small models are merged to create larger models. Figure 3 shows the hierarchy of the models for the parcel sorting system. The advantage of using hierarchical models is that problems can be found at an earlier stage of the testing. New rules on sub-models imply new rules on the containing models. Another advantage is that the coverage process is tested first on simple models which makes it easier to debug and cover. Note that not all the models in the hierarchy need to be implemented explicitly. Some models, especially very simple ones, like Model 2 in Figure 3, can be viewed as projections of larger models. Usually, smaller models are implemented if we want to use them to find new restrictions and estimate the size of the big model, so that we will be able to assess if the large model is feasible.

Whenever the parcel sorting system is operating, it creates, as a by-product, a trace of the messages between the processes in the system. This trace includes information such as dimensions for every

incoming parcel. It also includes messages which contain the arrival and departure time of a parcel, which processor processed it, and intermediate status of the parcel processing. This trace is processed and, from the information relating to each parcel, we create a tuple that is used in the coverage.

After collecting coverage information for some time we look for coverage holes which we do not succeed to find when generating tests. We first look at each variable separately as there are a fewer tasks to look at this way. We may find out, for instance, that we never saw a parcel on the master processor (`Master = 1`). After we make sure that we have full coverage for each variable, we look at combinations of some of the variables.

There are several possible reasons for holes in the coverage. We have already discussed missing restrictions, in which case we simply add new restrictions. Bugs in the application or the coverage process are also a possibility. However, the most common cause, and the reason for coverage in the first place, is that some tests are missing. We use coverage holes as pointers to areas which need more testing. We iterate the process until the coverage is complete or, more commonly, resources or time are exhausted. Eventually the restrictions will be almost accurate.

In many cases, the number of tasks in the coverage model seems daunting. For example, for the coverage model in Figure 2 with no restrictions, the number of tasks is $6*6*6*2*6*5*7*7*2=1,375,920$. In practice, the number of tasks is usually not a problem. The first reason is that the restrictions in many cases reduce the number of tasks by a number of orders of magnitude. In [7] we discuss a model in which the restrictions reduce the number from 30,000,000 to 1,500. The second reason is that one does not start with the full model but with partial models which have a smaller number of tasks.

5 General Purpose Tool for User Defined Coverage Models

One of the properties of functional coverage is that the coverage models are application specific. On the other hand, a common property of most coverage tools is that the coverage models which the tool is designed to handle, are hard-coded into the tool. Therefore, in order to make the tools applicable to many users, the models that are implemented in these tools are generic. The result is that there are almost no commercially available coverage tools for functional coverage in general, and specifically for the coverage models that the user needs.

This leaves a user who wants to use functional coverage with two options: to do coverage manually, or to build her own coverage tool for her models. The first option makes the coverage work tedious and time consuming and creates a severe limit on the size of the models that can be implemented. The second solution may require a large development effort for a tool that might be used once for a single program. Usually, tools that are built for a single use are less elaborate in the functionality they provide to their users. Such tools also tend to be error-prone. The result is that such tools cost much more than commercial tools and provide much less functionality.

In [7], we propose a different methodology for functional coverage. This methodology calls for separation of the coverage model from the coverage measurement tool. The idea behind this methodology is that most of the functionality provided by existing coverage tools, such as data gathering and coverage reports, is independent of the coverage models, and is thus similar in all tools. The main difference between coverage tools is the models they implement. Therefore, a single, general purpose tool, that will be oblivious to the coverage model and provide all the functionality of existing tools, can be used to provide all the coverage needs of a user both for generic and specific models.

To provide all the needed functionality of a coverage tool to a specific model, the tool has to be aware of the exact specifications of the model. Therefore, one of the inputs to the tool is the definition of the models. The definition of a model has to be done in a language which is simple enough for a user to use, yet rich enough to describe all the models that the user wants to cover. Our experience shows that a language that contains the predicates used in first order temporal logic (*and*, *not*, *exist*, *for all*, *before*, *after*, etc.) combined with the use of simple arithmetic (natural numbers, arithmetic and relational operators), is

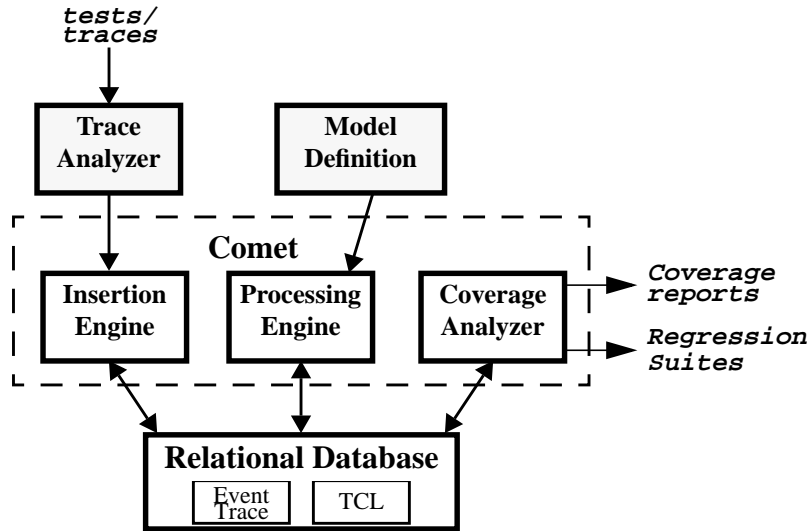


Figure 4. Comet Structure

sufficient.

The advantages of using such a general purpose coverage tool are enormous. First, it allows its users to define their own models, according to their specific needs, and still enjoy all the functionality of a dedicated coverage tool without the need to develop such a tool. It therefore provides users with a means to measure coverage on models which were not available to them before. The tool also allows organizations to use a single tool for most of their coverage needs, and not be forced to buy and maintain a set of tools, one for each coverage model. Another advantage of explicit and external model definition is that it enables sharing of coverage models between projects. Finally, the tool enables its users to adapt their coverage models to their testing resources, and refine these models during testing. For example, users that can afford only quick and dirty testing can define coarse grain models with a small amount of tasks, while users that want to do comprehensive testing can define finer grain models.

Based on the methodology described above, we have developed a coverage measurement tool named Comet. Comet enables users to define their own coverage models, gather and process traces to measure the coverage of these models, and generate coverage reports on the results of the measurement. Comet relies on a relational database in order to supply a comprehensive and stable environment needed for the coverage measurement process itself and an analysis of the coverage results.

Comet consists of three major parts, as shown in Figure 4: the *Insertion Engine* whose task it is to insert events from input traces into the database, the *Processing Engine* whose task is to process the traces in the database in order to detect coverage tasks according to the model definition, and the *Coverage Analyzer* which analyzes the measurement results and prepares coverage analysis reports according to the user definition.

Since Comet is oblivious to the coverage model and it can receive many types of traces, it requires two user-provided additions that do not exist in other tools. The first addition is the definition of the coverage model which is done using Comet's GUI. The second addition is a trace analyzer that converts the format of the traces to a standard format that Comet can handle. A more detailed description of Comet can be found in [7].

6 Comparison Between Code and Functional Coverage

In this section, we compare the advantages and disadvantages of code coverage and functional coverage. The comparison is based on the following criteria: availability of tools, cost, ease of use, learning curve,

portability, ability to focus on areas of concern, experience gathered, and ability to cover the entire design. Based on this comparison, we suggest several guidelines on where and when these coverage methods should be used. The summary of the comparison is shown in Table 1.

Property	Code coverage	Functional coverage
Availability	Many available commercial tools	Almost no commercial tools available
Cost	Cost of tools is usually low, no additional costs	Higher cost to develop tools and models
Ease of use	Easy to use. Plug and play	Need expertise in definition and implementation of models
Learning curve	Users can benefit from tool almost immediately	Need some time to learn how to define and implement models
Experience gathered	Vast experience on usage, coverage targets, etc.	Little experience, harder to reflect from one project to another
Focus	Uniformly spread on all the entire program	Focuses on areas of concern
Adapting to testing resources	Hard to adapt because of hard coded models	Easy to adapt. Can use right number of models and of the appropriate complexity
Missing code	Not fitted to deal with missing code	Can deal with missing code
Impact on project	Impact only on testing and testing requirements	Many areas of impact

TABLE 1. Comparison between code and functional coverage

Code coverage is a well established method. Many tools are available for almost any programming language. Code coverage has been extensively used in the industry. It is estimated that 20-30% of the testers have used it at least once. Most of the code coverage tools are inexpensive, easy to use, and do not require any (manual) changes in the code. The results they provide are easy to understand. We have had experience in the past in which we have found bugs using branch coverage tools less than an hour from the time the tool was introduced. These tools give a general sweep of the entire code for areas which were not used or not sufficiently used. The cost of the tools is relatively small, and their running time overhead is usually between 20% to 100%.

Another big advantage for code coverage is the vast experience that has accumulated from using it. There are many experts and publications, such as [3] and [10], that provide guidelines on how to deploy coverage, which coverage models should be used, what the accepted levels of coverage for each model are, and how to interpret coverage results.

Code coverage tools have some built-in faults. The first is that these tools do not find missing code. Another problem is that a coverage tool is not available for some environments (e.g., language, operating system) and therefore, since code coverage tools are expensive to create, in these environments coverage is not used.

Traditionally functional coverage was outside of the commercial tools domain. When functional coverage was needed, a domain specific tool was written in order to check it. Tools written for single applications are usually not very elaborate as far as their report generation capability or graphical user interface are concerned. The main advantage that these tools have is that they do exactly what the user wants. The disadvantages are cost, as they do not have the economy of scale, and that the tools are relatively simple and bug prone. With the introduction of functional coverage tools, such as Comet, functional coverage becomes a viable option in many more circumstances. The advantages of such a general purpose coverage tool over domain specific tools are enormous, as explained in Section 5.

More effort, more time, and more expertise are required in the deployment of functional coverage. The technical work of implementing a functional coverage model takes several hours, and requires a better knowledge of the tool than the use of a code coverage tool. Moreover, the knowledge on how to create good functional coverage models requires additional expertise and special training. This means that the learning time for using functional coverage is much longer than for code coverage.

As far as ease of use, learning curve, and cost are concerned, code coverage has the advantage. However, it is our experience that functional coverage models have many advantages as well. Creating a functional coverage model, as opposed to using an off-the-shelf tool, gives the application engineer a fresh point of view on his applications. It is our experience that creating the coverage models by itself, without even collecting the information, is considered worthwhile by our customers. We have found, through many examples, that the people who design and code the program cannot predict which configurations or scenarios are possible in the programs. Analyzing possible scenarios will, in some cases, find bugs. For this reason, functional coverage is now being incorporated into the high level design stage of a very large project that we work with.

Functional coverage methodology enjoys a number of other advantages over code coverage. The first and most important is the ability to create coverage models for areas of concern. For example, in maintenance where most of the total cost is spent, small changes are made to existing, tested, software. One would like to be able to focus the tests and coverage only on the areas of these changes. By using functional coverage, one can focus the resources where they are needed. Other examples include applications which contain a complex logic part, such as a scheduler in an operating system. One would like to ensure that the level of testing for such sections of the application is very high.

The ability to focus on certain areas also helps with adapting the coverage to the available testing resources. With code coverage, if you do not have the amount of resources needed to do statement coverage you will probably not do coverage at all. If your resources are more than sufficient to cover everything your coverage tool marked, you will not be able to use your additional resources. Functional coverage models can be as detailed or as high-level as desired. This advantage has an accompanying disadvantage: if you want to test the application, and there is nothing specific you want to focus on, then code coverage is better than functional coverage.

Functional coverage is less vulnerable to the problem of missing code than code coverage, since it is usually derived from the specification, not the code itself. Missing specification is still an area of concern but it can be circumnavigated in most cases if the coverage model is well defined.

Code coverage should be used for a uniform check on the test plan. Whenever the testing resources are enough to complete the test plan, code coverage is a cost effective way to find missing requirements which should be added. Functional coverage, on the other hand, should be risk driven. A few areas of the application which either contain risk or are very complex and error prone should be identified and functional coverage should be applied to them. If there is no specific area which is of concern, then only code coverage should be used. If the resources are not enough for using coverage for the entire code base, but there are areas of specific risk, only functional coverage should be used. This can be the case when maintenance is done where only small changes in tested software need to be tested thoroughly. For these kinds of changes, specific models can be made and the testing will focus on the changes.

7 General Guidelines for Usage of Coverage

In this section, we try to share some of our experience as to how coverage should be applied. Coverage should not be used if the resources used for it can be better spent elsewhere. This is the case when the budget is very tight and there is not enough time to even finish the test plan. In such a case, designing new tests is not useful as not all the old tests will be run. Coverage should be used only if there is a full commitment to make use of the data collected. Measuring coverage in order to report coverage percentile is practically worthless. Coverage points out parts of the application that have not been tested and guides

test generation to these parts. Moreover, it is very important to try to reach full coverage or at least set high coverage goals, since many bugs hide in hard-to-reach places. It is usually worthwhile to create an automatic test generator instead of generating all the tests manually. The combination of an automatic test generator and coverage is very potent as the tests can be biased in the direction in which coverage is missing.

Coverage is a very useful criteria for test selection for regression suites. Whenever a small set of tests is needed, the test suite should be selected so that it will cover as many requirements or coverage tasks as possible.

When coverage and reviews are used for the same project reviews can put less emphasis on things that coverage is likely to find. For example, a review for dead code is unnecessary if statement coverage is used, a review for boundary error in loops is redundant if the appropriate mutation coverage model [8] is used, and manually checking that some values of variable can be attained is not needed if the appropriate functional coverage model is used.

When there is a coverage expert who helps a number of projects, such as in our organization, an effort should be made to make the coverage reports as succinct as possible. It is more useful to have feedback of the form function SORT was never called than line 2 in function SORT was never executed (as well as lines 5,8,11...). For functional coverage this comment is even more important. A functional coverage model is usually an N dimensional space over some variables. The information that a single point has not been covered is not as useful as the information that a sub-space (as large as possible) was not covered.

The following guidelines are applicable only to functional coverage:

- First, the most complex, error prone part of the application has to be identified. Coverage models should be written for these parts at the high level design stage. The benefits are a fresh look at the design, a good start for the automation of the test plan and creating a model that will fit the implementation as it evolves, assuming that it still implements the same design in contradiction to code coverage, where coverage can start having an impact only after tests have been executed.
- Coverage models should be created hierarchically. One should start with simple small models and combine them to create larger models. Figure 3 illustrates a hierarchy of functional coverage models. Using hierarchical models enables us to debug the coverage process on simple cases and find problems in the coverage models early.
- Restrictions should be created at model creation time. It is our experience that programmers have a very hard time specifying the correct restrictions (They never do.) but that they find that they learn a lot about the design. Before coverage is measured, sample traces should be looked at to check that the restrictions are correct.

8 Conclusions

In this paper, we compared functional coverage to code coverage. We have shown that each has its own merits and drawbacks. It has been our experience, as well as the experience of anyone that we know has used coverage, that coverage is worth doing. Almost anyone, and under any timing and budget consideration, can benefit from some form of coverage. However, one has to commit to coverage and use it properly.

Functional coverage is a more powerful testing technique than code coverage, since it can focus on areas of concern, and contribute to the design and verification processes in many more ways. On the other hand, functional coverage is more complicated and requires more resources than code coverage. We therefore recommend that it should be reserved to those parts of a program that are of special concern. Code coverage, on the other hand, can and should be uniformly applied to the entire application.

References

- [1] Y. Abarbanel-Vinov, and S. Ur. Processor Bug Classification and Modeling, IBM's Haifa Research Lab internal document, 1996.
- [2] J. Baumgartner and R. Raghavan. Method to compute test coverage in complex computer system simulation. *IBM Technical Disclosure Bulletin*, 40(3):1-4, March 1997.
- [3] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990.
- [4] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1995.
- [5] E. Buchnik and S. Ur. Compacting regression-suites on-the-fly. In *Proceedings of the 4th Asia Pacific Software Engineering Conference*, pages 385-94, December 1997.
- [6] S. Cornett. Software Test Coverage Analysis, <http://www.bullseye.com/webCoverage.html>
- [7] R. Grinwald, E. Harel, M. Orgad, S. Ur, and A. Ziv. User defined coverage - a tool supported methodology for design verification, to appear in *Proceedings of the 35th Design Automation Conference*, June 1998.
- [8] W.E. Howden. Weak mutation testing and completeness of test sets, *IEEE Transactions on Software Engineering*, 8(4):371-379, July 1982.
- [9] J.R. Horgan, S. London and M.R. Lyu. Achieving software quality with testing coverage measures, *Computer*, 27(9):60-69, September 1994.
- [10] B. Marick. *The Craft of Software Testing, Subsystem Testing Including Object-Based and Object-Oriented Testing*. Prentice-Hall, 1995.
- [11] C. Kaner. Software negligence and testing coverage, In *proceedings of STAR 96: the Fifth International Conference, Software Testing, Analysis and Review*, pages 299-327, June 1996.
- [12] R. Stewart. Unit test coverage as leading indicator of rework, *EuroSTAR 97*, November 1997.
- [13] C-Cover - Test Coverage Analyzer for C/C++, <http://www.bullseye.com/webCcover.html>
- [14] DeepCover for Java, <http://www.rstcorp.com/DCJava.html>
- [15] Software test and evaluation guidelines, Department of the Army, Pamphlet 73-7

Practical Approach To Using Software Metrics

Presented by
Howard Chorney

**PROCESS SOFTWARE
CORPORATION**

Quick Quote

“Numbers are only numbers, and
alone cannot tell you anything
unless you know what your looking
for.”

**PROCESS SOFTWARE
CORPORATION**

PSC

Internetworking

Overview

- Why use metrics?
- Sizing the test effort
- Practical metric set
- Getting Started

PROCESS SOFTWARE CORPORATION

PSC

Internetworking

Why Use Metrics

- Vehicle to check what is planned against reality.
 - Checks status of test effort.
 - Flags any potential problems.
- Tracks condition of product throughout development cycle.
- Takes the emotion out of the ship decision.
- Helps plan follow on projects.

PROCESS SOFTWARE CORPORATION

Internetworking

PSC

How Are Metrics Most Effective

- By setting project goals
- Understanding project goals
- Measuring goals against reality

PROCESS SOFTWARE CORPORATION

Internetworking

PSC

Sizing The Test Effort

- Goals of my organization
 - Find and identify defects!!
- How do you know what your planning for?
- How do you know your there?
- Challenges and variables
 - Coding styles
 - Product type
 - Functional complexity of components

PROCESS SOFTWARE CORPORATION

Internetworking

PSC

Defect Predictions

- What is it, why use it, how can it help us?
 - Method of predicting defects in a software development project
 - Gives product team a baseline for testing
- Calculating defects
 - Defects per KLOC x estimated LOC / 1,000
- How do we know we did it right?

PROCESS SOFTWARE CORPORATION

Internetworking

PSC

Associated Risks

- Need some up front data.
- Not an exact science disparities will exist between prediction and reality. Could be viewed as consulting the magic oracle.
- Takes time to refine the prediction method.
- May be view as a quota system.
- May not be applicable for all test efforts.

PROCESS SOFTWARE CORPORATION

Internetworking

PSC

Defect Find Rate

- Provides high level view of defect discovery on a cumulative basis.
- Flags anomalies in the test effort.
 - Discovery rate decreasing to early in the project
 - Need to re-evaluate test strategy.
 - Front end coding practices paying off.
 - Discovery rate not decreasing at end of project
 - High quantity of defects, may need to re-evaluate release decisions.
 - Need to re-evaluate test strategy early on in project.
 - Need more testing resources.

PROCESS SOFTWARE CORPORATION

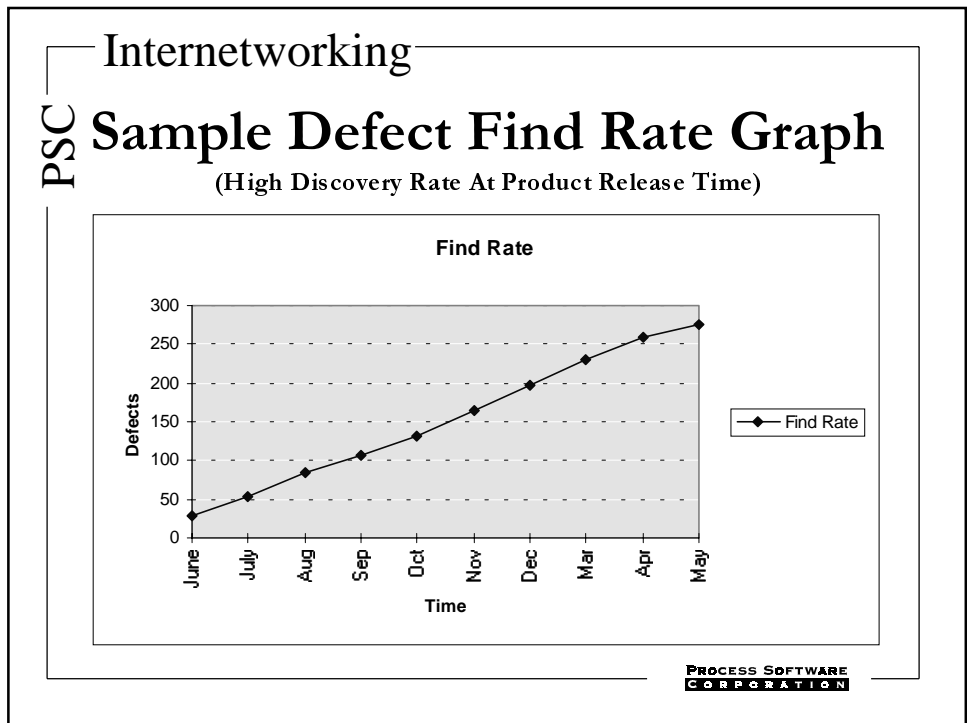
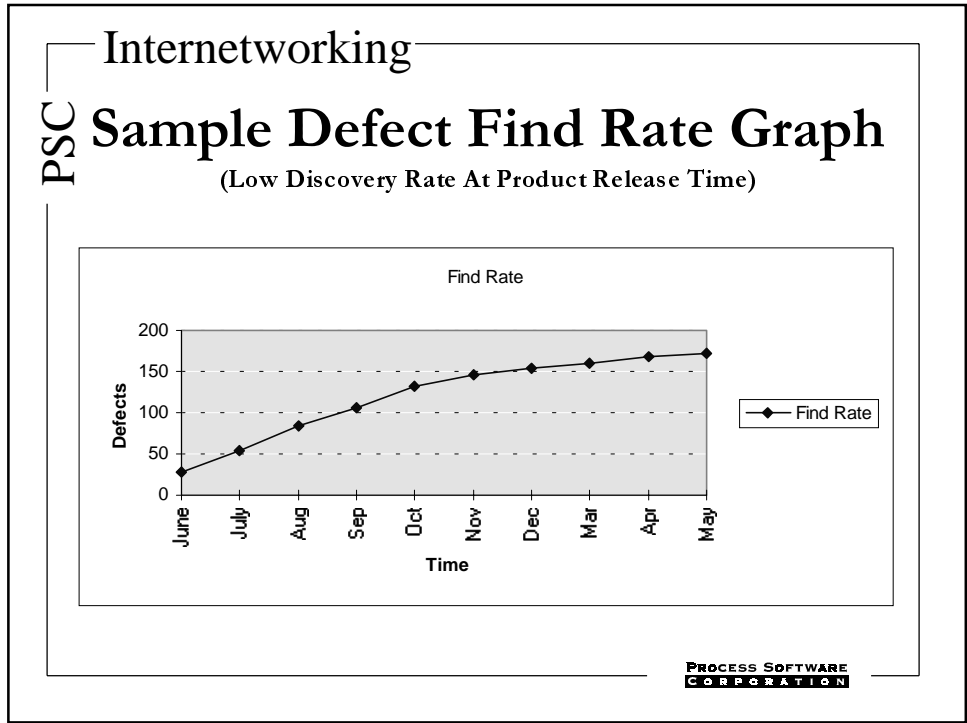
Internetworking

PSC

Defect Find Rate (Continued)

- Used to verify defect prediction metrics.
 - Flags amount of defects remaining in product.
- Can be tailored to component level.
- Can be used on a weekly or monthly level.
- Not effective for reporting detailed spikes.

PROCESS SOFTWARE CORPORATION



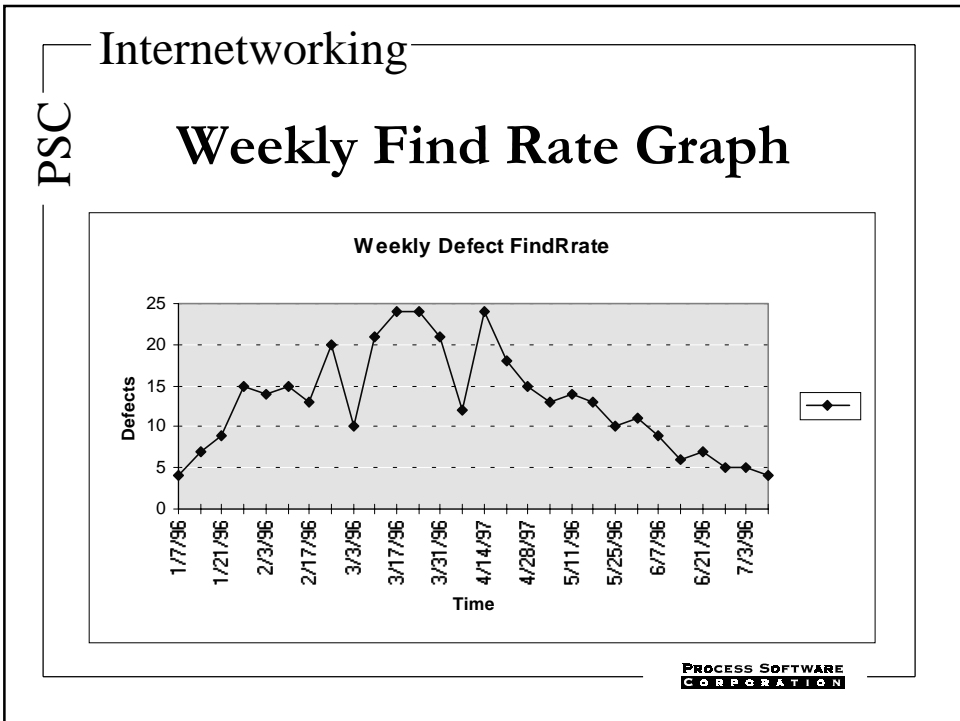
Internetworking

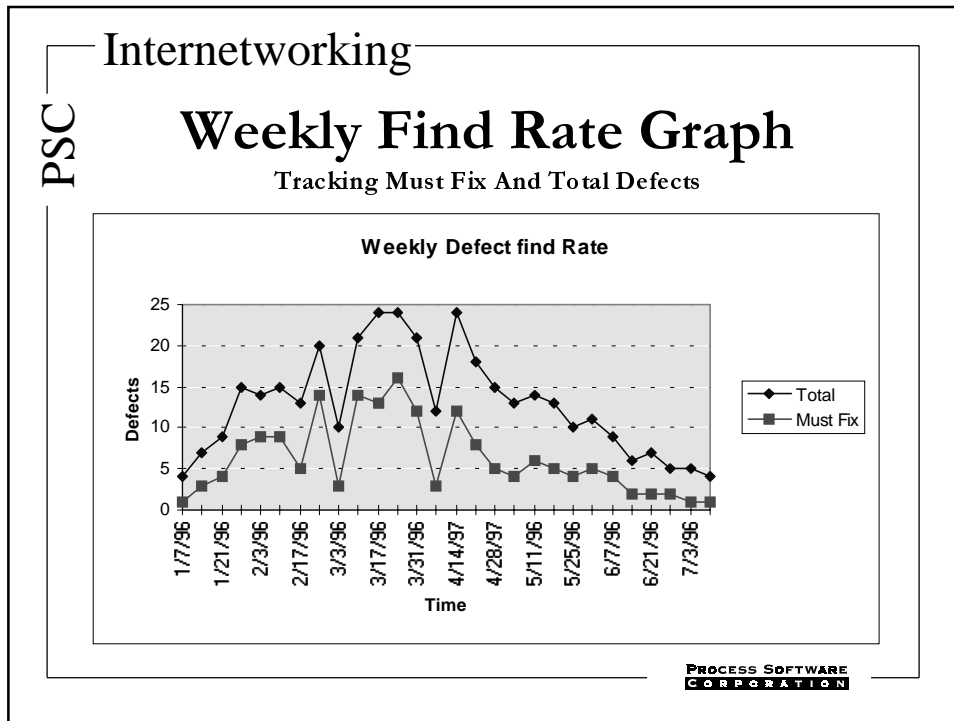
PSC

Weekly Defect Find Rate

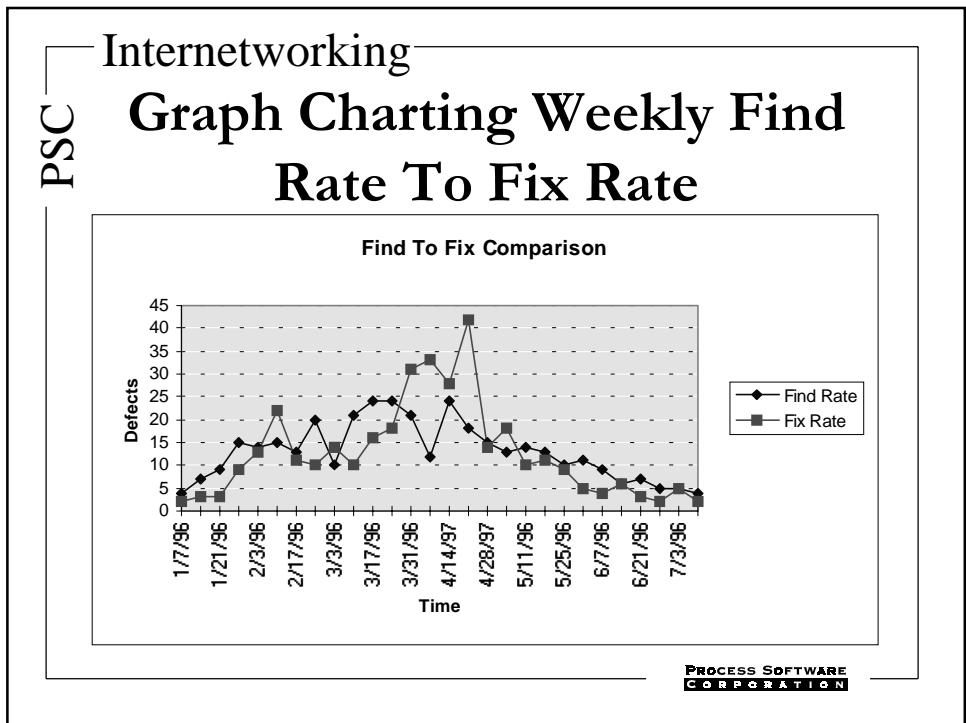
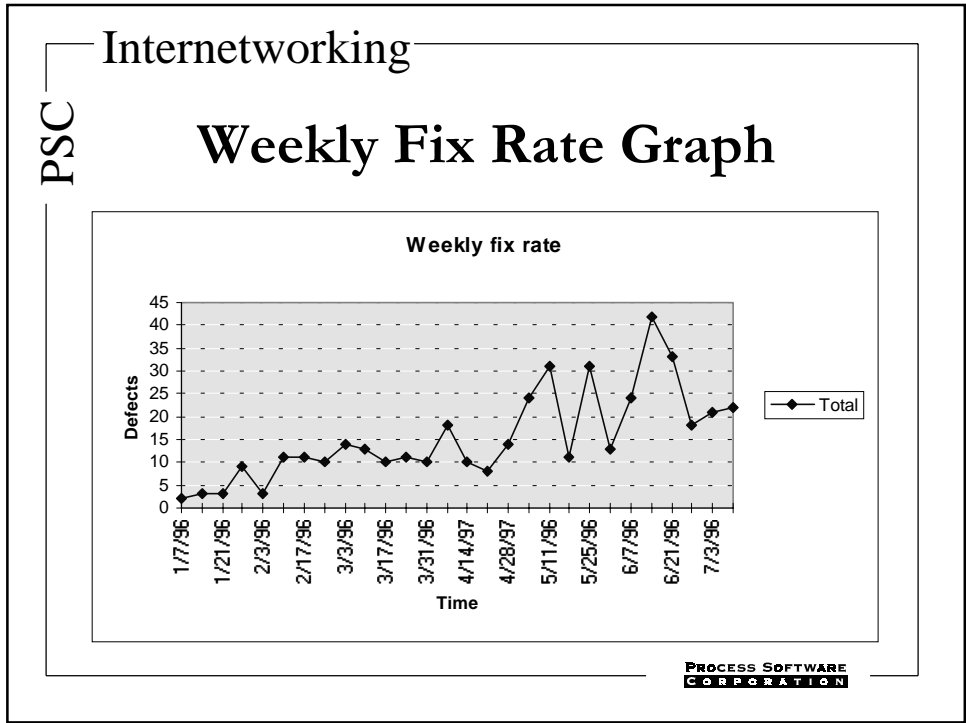
- Charts defects on a week by week basis.
 - Effective in monitoring test effort
 - Can detect more detailed spikes during test effort.
 - Effective in monitoring product condition.
 - Details find rate throughout project.
 - Measures test effort against project goals.

PROCESS SOFTWARE CORPORATION





- Internetworking
- PSC
- ## Weekly Fix Rate
- Monitors defect fix rate on a weekly basis.
 - Assists in assessing the defect regression strategy.
 - Understanding when the majority of fixes are done, can assist in planning when resources will be needed to verify fixes.
 - Assists in understanding product stability issues.
 - If fix rate is high at end of development cycle, how can product be stable?
 - Can be combined with Weekly Find Rate graph for comparing incoming defects to fixed defects.
- PROCESS SOFTWARE CORPORATION



Internetworking

PSC

Unresolved To Resolved

- Unresolved
 - Open
 - Acknowledged
 - Answered
 - InWork
 - Deferred
- Resolved
 - Fixed
 - Closed
 - Rejected

PROCESS SOFTWARE CORPORATION

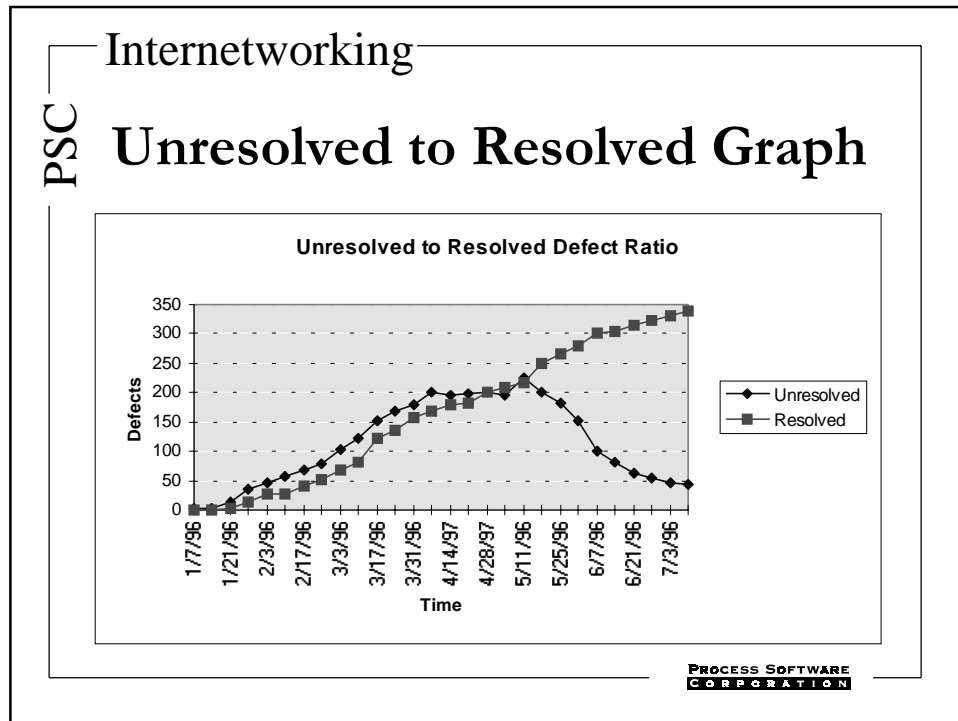
Internetworking

PSC

Unresolved To Resolved (continued)

- Monitors the amount of unresolved defects to resolved defects
 - Show product development team the relationship between unresolved and resolved defects.
 - Helps assess the work to be done to clean up unresolved cases.
- Caution!!
 - Cross over line can fluctuate, should monitor for set pattern before making condition judgment.

PROCESS SOFTWARE CORPORATION



- Internetworking
- PSC
- ## Getting Started
- Ask yourselves these question
 - Are metrics something that might be useful to your product development team.
 - What is your current situation and what information does your product development team need.
 - Does the product development team understands it's goals.
 - What tools will help us monitor our goals against reality.
- PROCESS SOFTWARE CORPORATION

Internetworking

PSC

Getting Started(Continued)

- Five Basic Steps
 - Assess
 - Assess what you need to do.
 - Research
 - Many publications are available with many different methods.
 - Choose
 - Choose the metrics that give you the information you need
 - Implement
 - Refine
 - It will take time to get what you need, be prepared to work on it.

PROCESS SOFTWARE CORPORATION

Internetworking

PSC

Interesting Reading

- Capers Jones, *Software Quality Analysis And Guidelines For Success*. International Thompson Computer Press, 1997
- William Perry, *Effective Methods For Software Testing*. Wiley-QED, 1995
- Tom Demarco, *Controlling Software Projects: Management, Measurements and Estimations*. Prentice Hall, 1982
- Stephen H. Kan, *Metrics And Models In Software Quality Engineering*. Addison Wesley, 1995
- Robert B. Grady, Deborah L. Caswell, *Software Metrics: Establishing A Company Wide Program*. Prentice Hall, 1987
- Robert B. Grady, *Practical Software Metrics For Project Management And Process Improvement*. Prentice Hall, 1992

PROCESS SOFTWARE CORPORATION

A Practical Approach To Using Software Metrics

Howard Chorney
Software Quality Manager
Process Software Corporation
Framingham, Massachusetts
chorney@process.com

1.0 Introduction

As Development Managers, Quality Managers and Project Leaders in today's rapid time to market environment, we face many interesting challenges. During my experience, I have found some of the more interesting challenges to be: how do we track the condition of a software project under development? How do we avoid unpleasant surprises a week before we are about to ship? How do we set quality goals for the product and how do we measure that we are meeting the goals? How do we know when a product's condition is stabilizing? Most importantly, how do we know we can ship? Other important questions are how do we scope the test effort for the project? What's the scope of the test effort? What are we capable of testing in the time frame allotted? How can resources be allocated or re-allocated to meet the testing needs? How do we do a better job from a testing standpoint on the next project?

It has been my personal experience throughout my career that many organizations have made these decisions based solely on gut feeling and emotions. I have especially found this to be true in a smaller company with little or no resources dedicated to software quality processes or initiatives. Unfortunately the consequences of adhoc planning and making uninformed decisions have resulted in the inability to make product ship dates for causes such as product quality problems or inadequate testing. These types of decisions have also resulted in products being released prematurely, resulting in high product rework costs and customer satisfaction issues. High product rework costs cannot only affect an organization financially, they can also cause delays in follow product work, which affects time to market concerns.

Fortunately these problems can be avoided through the use of software metrics. There are many metrics that can be used to provide the project development team with the information needed to track the condition of the product and the condition of the test effort during the development cycle. This information can be used to aid the team to make informed decisions about the product. There are also metrics that can be used by the test team to aid in the planning stage and execution stage of the test effort.

A Practical Approach To Using Software Metrics

The most difficult part of the use of metrics is not interpreting metrics themselves, it is determining the information needed from the metrics and then choosing the correct metric to yield the desired information. This paper will focus on using a specific set of software metrics I have used as a software quality manager during the development cycle, and what information the metrics have provided me. It will also discuss the individual metrics within the set, including what each metric reports, where the metric is best used in a project, the advantages and disadvantages (if applicable) of each metric, and how certain metrics cross check each other. Finally, it will focus on how to implement the use of software metrics.

Metrics are tools at our disposal to aid us and flag us when things are not going along as we had planned. Metrics are not foolproof and results may vary from time to time. However, Metrics are an excellent tool to look for trends. When a metric is deviating from a stated or predicted trend it gives us the opportunity to go investigate the reason for the deviation. In some cases we may discover real problems and may need to address something in our product development effort. In other cases we may find that the trend we have predicted is inaccurate and may need to adjust our information. In any case it is better understand potential problems early when something can be done, then when it is too late.

2.0 Sizing the test effort

In all of the organizations I have been associated with, the overall charter of the group has been to find and identify defects in the product. Based upon this charter a few of the challenges in the planning effort have been to understand where to look for defects, understand how to look for defects, and understand if all the defects have been found. Additional challenges are figuring out how to measure the fact that we are getting anywhere close to finding all the defects we need to find, where are the risk areas in the product, and how much time is it going to take to find all the defects.

When the Q.A. manager or project leader is sizing up the test effort for an upcoming project, there are many factors that need be taken into account when trying to figure out all the defect information discussed above, and planning the testing resources to find them. One basic element to resource planning is if you do not understand how many defects you have to find, how can you plan for finding them? I have not found one surefire method to provide all the information I need to estimate this work accurately, nor do I believe there is one. However I would rather use a method that gives me some measurable information over an adhoc guessing method. In choosing a method for measurable resource planning one must be willing to accept the fact that no method is completely accurate and each method will most likely have to be used and refined throughout multiple project cycles.

A Practical Approach To Using Software Metrics

2.1 Planning Challenges

When looking at the factors and addressing the challenges, there are variables in the planning process depending upon whether the project is based upon new product development or based upon making enhancements to an existing product. One also has to look at variables in the coding methodology, such as whether the development effort consists of porting existing code to a new platform, enhancing existing code or developing brand new code.

Other challenges are based upon the complexity of the functionality of the components being developed. Looking at the functional specification should aid in determining the functional complexity of the component, and aid in designing functional tests for the component. However, the question remains, will a design from a functional spec ensure the ability to exercise all the code paths and meet all the test requirements needed to adequately test the component and find all the defects within the component? Code coverage tools are an excellent aid to ensure code path coverage and are highly recommended when ever possible. However, older operating systems and legacy applications often do not lend themselves very well to code coverage tools and many do not have any tools associated with them.

Once coverage is planned from a functional specification, will there be enough information to define the total time needed to test the component and if so, how can one measure success or failure from a time standpoint? One way to track this information is to make time estimate based upon “x” amount of time per command and have the tester executing the tests keep a record of the time spent to verify the accuracy of the estimation. To ensure some consistency the functional test matrix could include a time section with specific parameters defined around the recording of time, to handle exceptions such as time taken to reproduce problems, etc. Note that the data out is only as good as the individuals keeping the records. Another factor, is keeping the records can be very time consuming and interfere with hands on test time. The data could also vary if different testers are inputting the data, regardless of the process parameters. This form of time estimation would have to be done with every component in the product, which could be a very time consuming effort in itself.

When taking all these factors into account and attempting to formulate a somewhat accurate resource plan, it is helpful to find a method that takes all factors into account in an overall sense. The method to be discussed allows a project planner to come up with a measurable mean to predict the number of resources needed during a project by attempting to understand the amount of defects to be found. The method can also use other metrics discussed further on in this paper to track the number of defects found throughout the project life cycle.

A Practical Approach To Using Software Metrics

2.2 Defect Prediction Metric

The defect prediction metric is usually derived at the beginning of a project. The defect prediction method is based upon a fairly simple premise, it estimates the number of defects that might be present in a software product. Then it calculates how long it takes your organization to find the defects. As discussed earlier, no method is totally accurate. However, during my experiences I have found that this method provides a good baseline for resource estimates, and it can be refined and updated from project to project. When first starting out with the defect prediction method it may be a good idea to use it in conjunction with an organization's existing resource methods for comparison purposes.

In order to really implement the defect prediction method, it is best to have some sort of existing product history in the company. This may be a difficult approach to use for a company producing their first ever product. It also may be difficult to use if your company has not practiced the recording of defects when a defect is found. One of the more important things to remember when using the defect prediction method is an organization needs to start somewhere.

2.3 Calculating Defects

Defects prediction is based upon known or estimated lines of code and measured defects per KLOC. (KLOC = 1,000 line of code). It obviously is not possible to get the defect per KLOC numbers on a new project. However, by looking at historical data in previous product development efforts, a baseline for defects per KLOC can be formed for initial use in the defect prediction effort. If an organization is practicing code reviews, it is important to understand if the defect information is recorded and included in the KLOC number when planning test resources. It has been my experience, that an estimate of 50 defects per KLOC is a good starting point to work from, when initial data is unavailable.

Another caveat is that predicting the lines of code to be produced is not an exact science. The lines of code prediction will have to be an estimate and the estimate needs to come from the engineering development team. In the past I have learned, when a developer specs out their work, the developer is usually pretty close at estimating their lines of code. I have also learned that using a 10% plus or minus factor in the estimate is a safe practice.

Once the numbers of lines of code are predicted, and the average defects per KLOC have been established, divide the total lines of code by 1,000 and then multiply that number by the average defects per KLOC number. This provides the baseline number for predicted number of defects in the product. This baseline can be set for the entire product for one component. Once the baselines are set, using the defect find rate curve for defect tracking can be measured against the baseline during the life of the product development project. One note of caution, after a baseline has been set it is important to

A Practical Approach To Using Software Metrics

work with the test team to ensure the defect prediction metric is not viewed as a quota system. If the rate is achieved and defects are still being found, work should still continue in the defect finding efforts.

The defect find rate curve can also be broken down by components and used with the defect prediction method to measure the test effort via component. There is an advantage to tracking the defect rate by component. It allows the test team to understand if a particular test strategy is effective by measuring defects found against the baseline. If a test strategy has been defined, a baseline has been established and the defect yield has peaked far before the baseline has been reached, the test team now has a flag that the test strategy may have to be re-evaluated. Re-evaluation of the test strategy could consist of changing the test tactics, or it could mean that the original defect prediction rate is incorrect. In either case it alerts the test team to potential issues in the component.

2.4 Tracking Defect Prediction Results

Using the defect find rate curve along with the defect prediction method allow the product development team to track several things. As in component level tracking, it flags potential problems with test strategies, if the find rate curve levels out far before the defect prediction rate has been met. It allows the team to track defect prediction accuracy and helps the team adjust it's defect prediction methods. If the defect find rate is still rising, the defect prediction metric has not been met and if the product is close to it's scheduled ship date, it flags potential quality risks with the product. If this is tracked on a weekly basis, actions could be taken to add or shift resources to different areas of the product.

2.5 Utilizing Defect Prediction Results

How can the defect prediction method be used in resource projections? Once the baseline is established, the product team needs to understand what it will take to find the amount of projected defects. From a human resource standpoint it is advantageous to understand the rate at which the test team can find defects. This can include all methods of finding defects from code reviews to automated test development to test execution. Understanding this will allow the product team to understand how many defects can be found by existing test resources in the time allotted for the product development project and if additional resources will be needed to find the rest of the defects. If no additional resource can be added, it allows the product development team to understand potential quality risks come product release time.

Summarizing the perceived advantages of understanding and using the defect prediction method:

- Combined with a Defect-Find-Rate-Per-Hour metric, the defect prediction metric helps the product development team understand the scope of the test effort and how many resources will be needed to test the product.

A Practical Approach To Using Software Metrics

- Provides the product development team an estimate of how many defects remain in the project at any one point in time.
- Combined with the Defect-Find-Rate Curve metric, the defect prediction metric can provide the product development team information to determine if the product test strategy for identifying software defects is on target.

Summarizing the perceived risks of using the defect prediction method:

- Defect prediction is not 100% precise; some disparities will exist between predictions and reality.
- May be viewed as a quota system and the defect finding effort might diminish after all the predicted defects are found.
- This metric might not work perfectly the first time it is used. It might take a few product releases to learn how to more accurately estimate product defects.

3.0 Defect Find Rate Curve Metric

The defect find rate curve metric displays the total amount of defects found during the product development life cycle. This metric is a cumulative find rate of total defects over time. This metric is charted by a graph displaying the number of software defects over time. How does this metric provide value to the product development team? By using this metric the team can monitor defect discovery rates, and cross check defect prediction metrics as previously discussed.

3.1 Defect Discovery Rates

The defect find rate curve aids the product development team in observing at what point during a project the majority of defects are found. It can be argued that this metric is not totally accurate and the slope of the defect discovery rate will increase and decrease depending upon how the test effort is planned. However, by understanding trends and setting defect discovery goals, this metric can measure the test effort against the planned goals, and raise flags when things are not going as planned.

One disadvantage of this metric is that it can not pickup detailed spikes and display peaks and valleys to look at more detailed information; using a weekly defect find rate metric would be more beneficial. The weekly defect find rate metric will be discussed in the next section.

A summary of potential information this metric can highlight:

- Defects are found as predicted and the product development team believes the test effort is on track.
- The defect find rate curve has not flattened out in the final stages of product development, indicating more defects are still in the product.
- A quantity of defects remains based upon the original Defect Prediction Metric.

A Practical Approach To Using Software Metrics

- The defect find rate curve has flattened out too early in the product development effort and alerts the product development team to re-adjust the test effort to identify the remaining defects.
- Indicates where the bulk of the testing effort is most effective.

3.2 Examples of Defect Find Rate Graph

Looking at the graph, in theory the find rate climbs steeply during the early to middle stages of development. As product development enters the final stages, defects should be harder to find and the curve should flatten out. Figure 1 displays a project where the defect discovery has decreased over time and the defect find rate curve has flattened out.

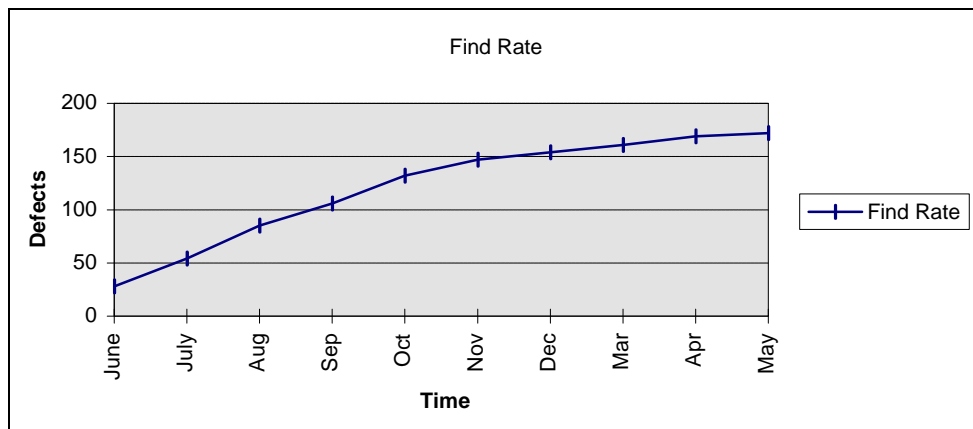


Figure 1

Figure 2 displays a project where the defect discovery has failed to decrease over time and the defect find rate curve continues to climb.

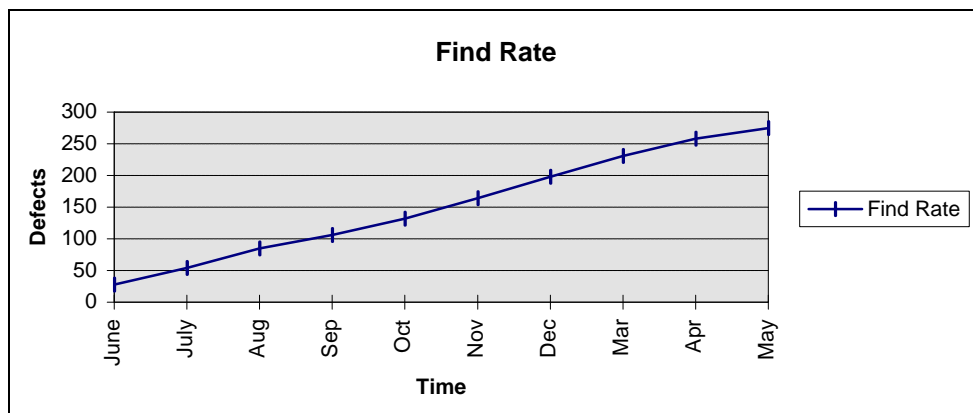


Figure 2

This metric can also be designed to display a comparison between total defects versus must fix defects as shown in Figure 3.

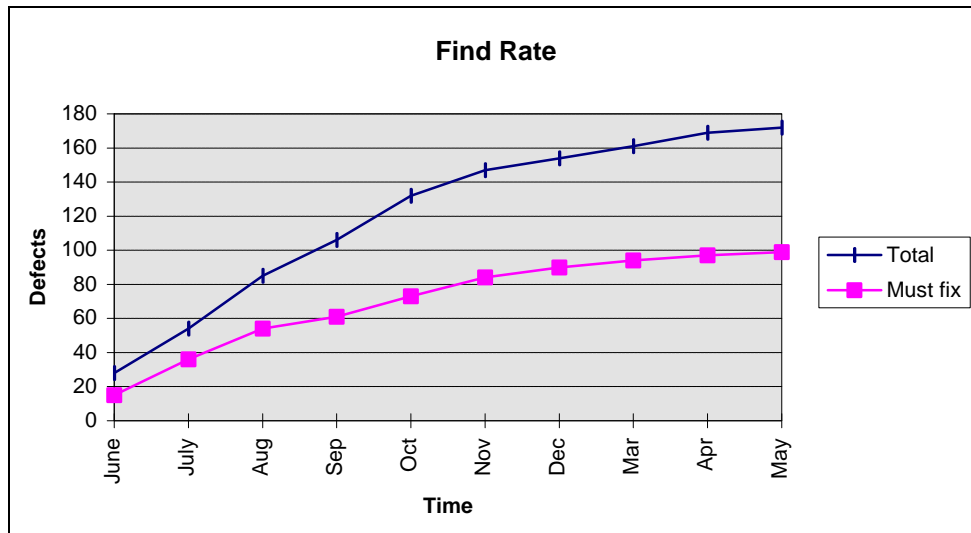


Figure 3

4.0 Weekly Defect Find Rate

This metric charts defects found on a week-by-week basis. This metric plots defects found on a weekly basis and is charted by a graph displaying the number of software defects found over time. How does this metric provide value to the product development team? This metric is an excellent tool for monitoring the test effort and condition of the product.

4.1 Monitoring The Test Effort

By understanding trends and setting defect discovery goals, this metric can measure the test effort against those goals and raise flags when things are not going as planned. For example, when planning the test effort it should be expected that the defect discovery levels will be higher when testing code new to the product, either in the form of new features, or enhancements to existing features. Using the weekly defect find rate, a product development team can monitor the test effort to see if defects are actually being found when the testing is scheduled to take place.

4.2 Monitoring The Product Condition

This metric can also be used to help determine product quality. In theory the find rate should stay high during the early to middle stages of development. As product development effort enters the final stages, defects should be harder to find and the weekly find rate should decrease.

Summary of potential information this metric can highlight:

- Defect find rates are decreasing during final stages of product development.

A Practical Approach To Using Software Metrics

- High rate of defects are still found during final stages of product development.
- The time needed to find problems during all stages of the test effort.
- Identifies proper test effort is taking place during product development.

4.3 Examples Of Weekly Defect Find Rate Graphs

Figure 4 displays a graph where the majority of the defects have been found during the early to middle stages of the test effort. Some of the spikes may indicate time required to develop test suites, time to reset test beds, or personnel issues.

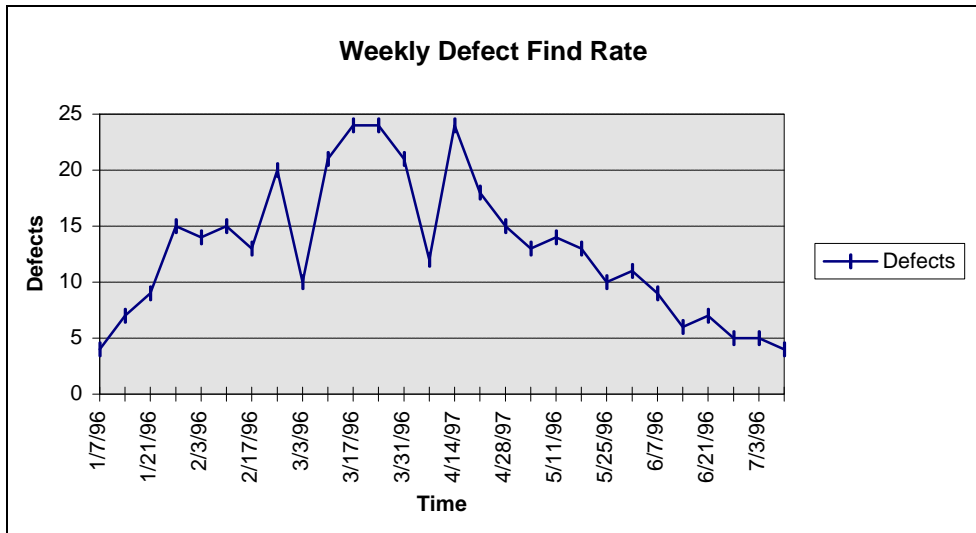


Figure 4

A Practical Approach To Using Software Metrics

This graph is also easily tailored to depict the software defect find rate by defect severity levels as displayed in Figure 5.

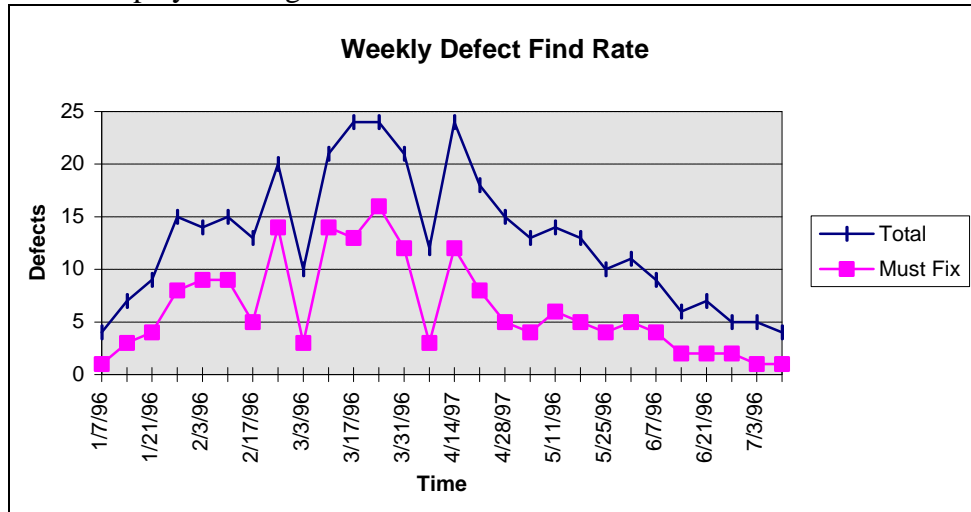


Figure 5

5.0 Weekly Closure Rate

This metric charts the software defect closure rate on a week-by-week basis. The metric is charted by a graph displaying closed software defects over time. How does this metric provide value to the product development team? It is an excellent tool for monitoring the progress of defect fixes, assessing the defect regression strategy and understanding the product's stability.

5.1 Monitoring The Progress Of Defect Fixes

By charting the fix rate on a weekly basis the product development team can easily assess the progress of the Open Defect fix rate. By understanding trends and setting goals for defect fixing, the weekly defect find rate can flag the product development team when things are not going as planned. For example; if the fix rate is low and there are a number of outstanding defects that need to be addressed, an adjustment of resources can be made to increase the defect fix rate.

5.2 Assessing The Defect Regression Strategy

By charting the fix rate on a weekly basis the product development team can assess the amount of defects that have been fixed, and plan the resources needed to verify the fixes. If resources cannot be spared to verify the fixes, the potential risk factor can be assessed at an early stage, and contingency plans can be made. Note that, at first using the defect fix rate in this capacity does not allow for much up-front planning, however over a period of a few product releases the product development team can observe patterns and plan resources for follow on product accordingly.

A Practical Approach To Using Software Metrics

5.3 Understanding Product Stability

In theory, at the beginning of product development, the software defect closure rate will be low. As time goes on, the weekly closure rate should increase as more software defects are being fixed. During the final stages of product development, the weekly closure rate should decrease. By charting the fix rate on a weekly basis and understanding trends and goals the product development team can be flagged when things are not going along as planned. Figure 5 is a sample graph of a product becoming stable as time goes on.

5.4 Find To Fix Comparison

Comparing the weekly defect find rate with the weekly fix rate can also be a useful aid when determining product stability. This gives the product development team a quick snap shot of the condition of the product under development.

Summary of potential information this metric can highlight:

- A low closure rate versus a high rate of opened defects indicates that the product might not be very stable. Doing this on a week-by-week basis should call attention to this problem early, so it is possible to:
 - ⇒ Make adjustments in resource allocation to correct the problem.
 - ⇒ Determine if the product development schedule is at risk.
- If the closure rate is low and the known defect rate is low, then the product is fairly stable.

5.5 Examples Of Weekly Defect Find Rate Graphs

Figure 6 displays a graph where the majority of the defects have been fixed at an early enough time to allow for regression testing to take place on a fairly stable product.

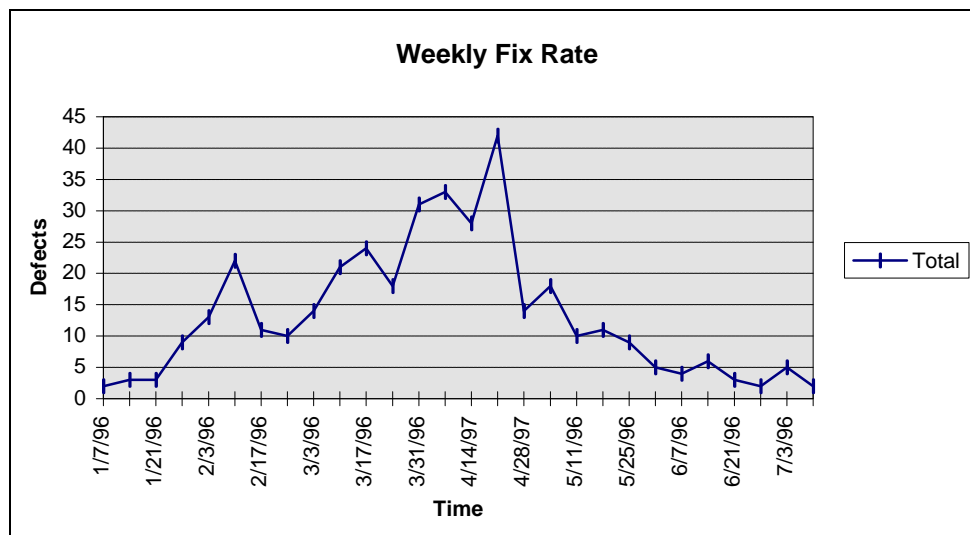


Figure 6

A Practical Approach To Using Software Metrics

Figure 7 is a sample graph of a product failing to become stable as time goes on. Please note the amount of activity at the end of the project. All the last minute changes never allow the product to become stable.

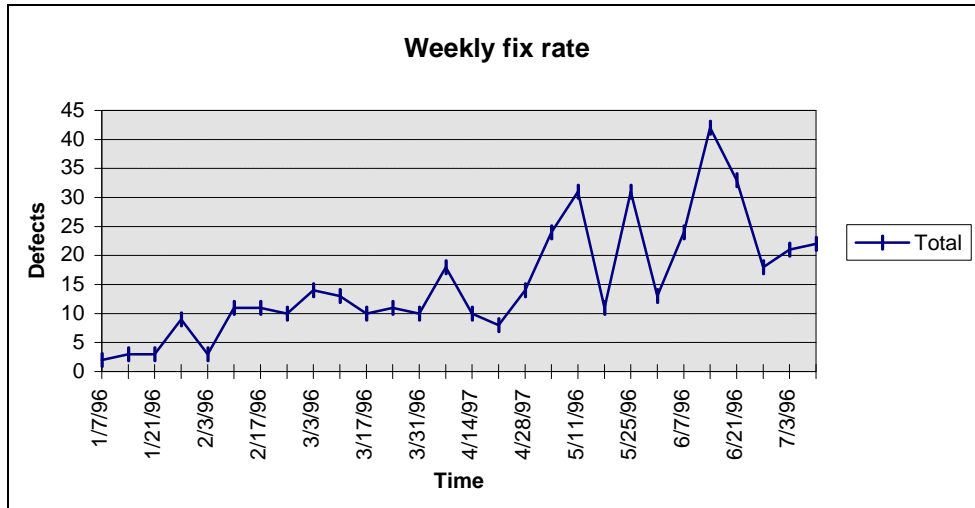


Figure 7

This graph is easily tailored to depict the weekly closure rate by defect severity levels as displayed in Figure 8.

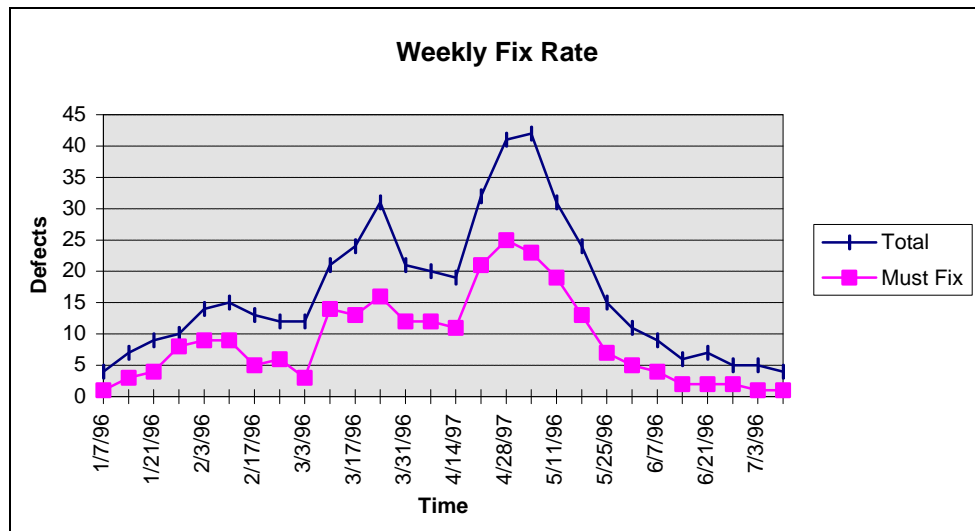


Figure 8

A Practical Approach To Using Software Metrics

Figure 9 depicts a graph comparing the weekly defect find rate with the weekly fix rate.

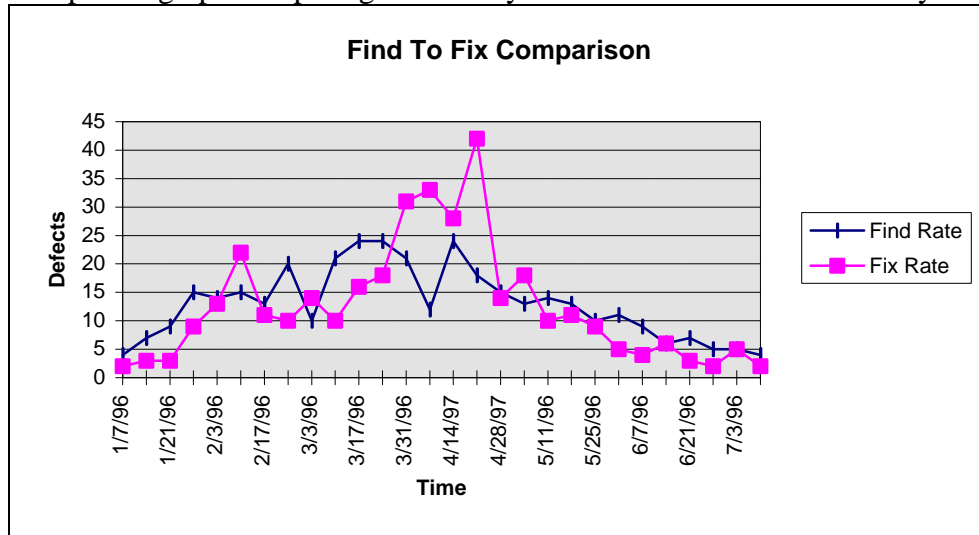


Figure 9

6.0 Total Resolved-to-Unresolved Defect Rate

This metric compares the total number of open and unresolved software defects against the total number of resolved software defects. For this case unresolved defects refer to open. Deferred defects, resolved refers to fixed, closed and rejected defects. This metric is charted by a graph displaying unresolved and resolved defects over time. How does this metric provide value to the product development team? It metric is a tool for displaying the gaps between the amount of total unresolved defects to amount of total resolved defects.

6.1 Comparing The Gaps

In theory, as the product development effort progresses the amount of unresolved defects should exceed the amount of resolved defects during the early stages of development. Eventually, the amount of resolved cases should exceed the amount of unresolved cases. At this point, the quality of the product potentially should be improving.

This metric is best used as a high level snapshot when projecting product quality. On it's own this metric should not be used solely to project product quality. In many cases the amounts of unresolved to resolved defects may fluctuate for a period of time. It is best to observe the gap between unresolved and resolved defects widen, before projecting any quality claims. This metric has increased value when used in conjunction with the weekly find rate and fix rate metrics.

A Practical Approach To Using Software Metrics

6.2 Examples Of Unresolved to Resolved Graphs

Figure 10 depicts a project where the unresolved defects and resolved defects fluctuate before the gap widens.

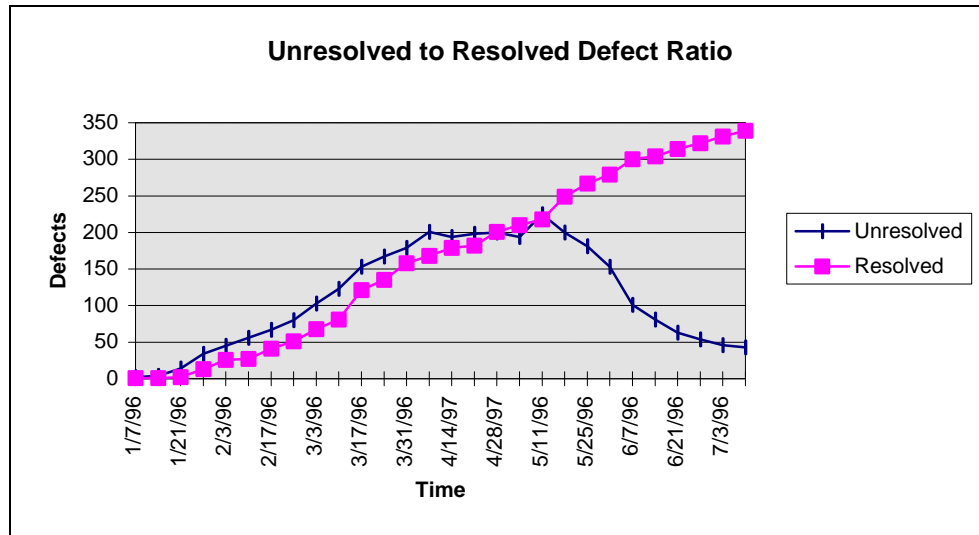


Figure 10

7.0 Taking the First Step To Get Started

Now that a basic set of metrics have been discussed, the next question is; is this information something that would be valuable to your product development team? If the answer is yes, what are the steps your team needs to take to move forward? Every product development team's needs are different. A recommended first step to any product development team is getting together and analyzing the current issues occurring during the product development cycle, then formulating a plan to address each issue.

As far as using metrics are concerned, the first and foremost step is to formulate quality and test goals around the product development effort. If the product development team does not understand what it needs, a metrics program cannot provide much information. There are many publications on the market today that discuss setting quality and testing goals and using metrics to check the actual results against the goals. It is highly recommended, before embarking on any program. To review at least one publication that may be beneficial to a product development team wanting to implement quality metrics.

Finally, whether your product development team chooses to use the methods and metrics discussed in this paper, or it chooses a completely different set of metrics, it is absolutely worthwhile to take the time and implement some form of software quality metrics. Immediate results may not be achieved, and it may take some time to understand

A Practical Approach To Using Software Metrics

how to set goals and use the information, but eventually the time invested will payoff, and the overall payoff will be developing a better software product.

Some Interesting Reading

Capers Jones, *Software Quality Analysis And Guidelines For Success*. International Thompson Computer Press, 1997

William Perry, *Effective Methods For Software Testing*. Wiley-QED, 1995

Tom Demarco, *Controlling Software Projects: Management, Measurements and Estimations*. Prentice Hall, 1982

Stephen H. Kan, *Metrics And Models In Software Quality Engineering*. Addison Wesley, 1995

Robert B. Grady, Deborah L. Caswell, *Software Metrics: Establishing A Company Wide Program*. Prentice Hall, 1987

Robert B. Grady, *Practical Software Metrics For Project Management And Process Improvement*. Prentice Hall, 1992

ESSI PIE No. 27839

HYPER

Quality Assurance Technologies for the EURO Conversion

Industrial Experience at Allianz Life



Brigitte Klein
Allianz Lebensversicherungs-AG
Reinsburgstr. 19
D-70178 Stuttgart

Lionel Briand, Bernd Freimut,
Oliver Laitenberger, Günther Ruhe
Fraunhofer IESE
Sauerwiesen 6
D-67661 Kaiserslautern

Session Agenda

- **Introduction** - The Company and its IT Department
- **EURO Conversion** - Characteristics and challenge
- **HYPER Project** - Motivation and objectives
- **Improvement Approach** - Software process improvement program and results
- **Technology** - Perspective-based inspections, statistical quality control, goal-oriented measurement
- **Procedure** - People, process, and technology
- **Initial Results** - Initial evaluations and assessments
- **Lessons Learned** - Initial experiences
- **Outlook** - Further actions

Introduction

The Company: Alliance Life Assurance

- member of the world-wide Allianz Group
- market leader of life assurances in Germany
- premium income DEM 12.5 billion, insured sum DEM 305 billion
- number of insurance policies: 7,723,000
- currently 5.500 employees
- further information on web-page: <http://www.allianz-leben.de>



The IT Department

- more than 500 employees, 350 of them application developers
- development of commercial software, mainly online transaction applications with very large databases (about 8500 programs)
- building up a new generation of client-server applications
- our goal: to reach level 3 to 4 of the SEI maturity model in the Year 2000

EURO Conversion project

Characteristics

- about 100 employees involved in the project
- overall effort: about 300 person months (4 IT / 6 User Departments)
- project with several levels and final dates
- involving about 250 programs (PLI, C, ASS), 550 amount fields, 14 neighbouring information systems concerning the adaptation

Challenge

- strategic importance for the company as market leader
- competition is largely based on quality and functionality of IS
- to conclude life insurance contracts either in DEM or in EURO at the beginning of 1999
- to master the complexity and large extent of the conversion
- to guarantee the essential quality of the final software systems



Improvement approach

Software process improvement program (since 1993)

- to quantify the quality of products & processes (→ measurement programs)
- to better understand the current situation and its causes
- to identify and implement improvements
- to detect, formalise, and structure experience (→ experience base)
- to continuously improve the maturity of processes and products

Results

- about 30% of the development effort is testing effort
- about 50% of detected defects originate in early phases
- too much rework required
- requirements not stable enough
- communication between project members must be improved
- effectiveness and efficiency of inspections are too low



HYPER Project

“High Quality of Software Products by Early Use of Innovative Reading Techniques”

Motivation

- to improve software quality through usage inspections with PBR
- to enhance customer satisfaction through early verification of requirements
- to improve communication between project members
- to improve control of the overall verification and validation process
- to use the results from the HYPER Project for broader use of PBR

Objectives

- to improve productivity by finding defects earlier
- to reduce the number of defects originating in prior phases
- to reduce test effort and overall effort
- to improve customer satisfaction



Fraunhofer IESE in Kaiserslautern



Fraunhofer Society (FhG):

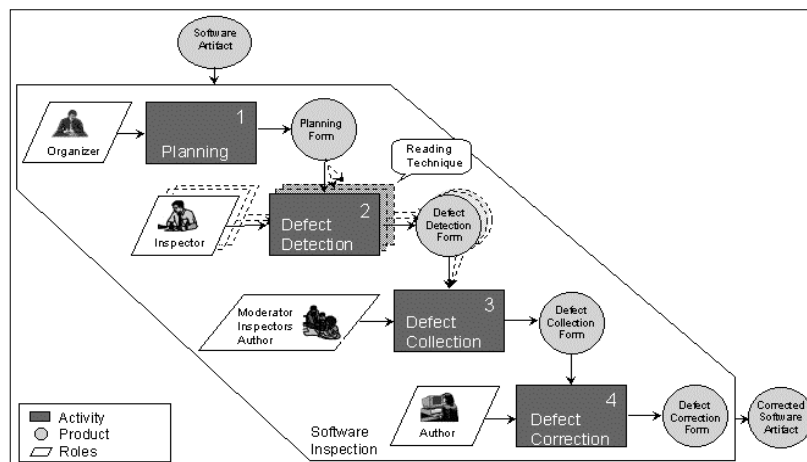
- 48 institutes
- ~ 9,000 employees
- budget of 1.3 billion DEM

FhG IESE:

- 60 full-time employees (45 scientific) from 9 countries on 3 continents
- Head: Prof. Dr. Dieter Rombach
- Sister institute near Washington D.C.
- Head: Prof. Dr. Vic Basili

Further Info → Visit our WWW page:
<http://www.iese.fhg.de>

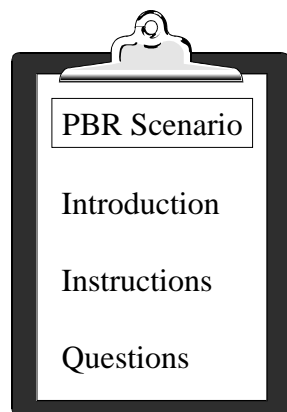
Perspective-based Inspections



Perspective-based Reading

- Problem**
- Software quality is a multi-dimensional concept
 - Quality properties depend on responsibilities in the development process
 - Inspectors get little guidance
 - Inspection result is not repeatable
 - Inspectors replicate each others effort
 - Effort for additional inspectors is not justified
- PBR Solution**
- Read a software artifact from specific perspectives of interested people (customer).
 - Provide procedural descriptions (scenarios) for how to perform the checking.
 - Let inspectors create and document intermediate checking products in order to make results repeatable.
 - Assign perspectives to inspectors (n:n).

Perspective-based Reading (2)



PBR Scenario

Introduction

} What quality factors are interesting?

Instructions

} How to extract information?

Questions

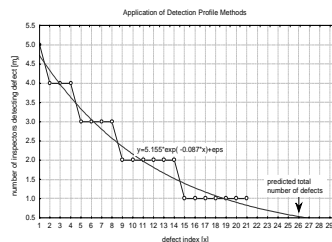
} How to probe extracted information?

How to decide when to re-inspect

- Context** • Inspections are valuable since defects are detected early, before they propagate to subsequent development phases.
- Goal** • Maximize number of defects found by re-inspecting defect-prone documents.
- Problem** • Objective decision when to re-inspect.
- Solution** • Estimate the number of remaining defects with Defect Content Models.

Defect Content Estimation

- Capture-Recapture Models** • Origin in biology: Estimate the size of populations based on incomplete samples
- Based on the overlap of detected defects, the total number of defects can be estimated
- Various models with different assumptions exist.
- Detection Profile Method** • Count number of inspectors detecting a defect. Sort defects descendingly. Fit curve through data points.



Measuring cost-effectiveness

How?

- Proportion of test effort saved due to the introduction of inspections

$$\frac{\#inspection_defects \times testcosts - \#inspection_defects \times inspectioncost}{\#total_defects \times testcosts}$$

inspectioncost = avg. costs to find and fix defect in inspections
testcosts = avg. costs to find an fix defect in test

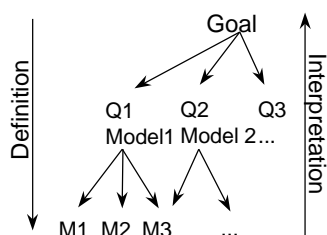
Why?

- Evaluate benefits of technology transfer
- By developing benchmarks, inspections can be compared within an organization or with industry

Issues?

- If necessary, combine objective measurement and subjective measurement.

Measurement: The GQM Approach



• GQM Goal:

- Explicitly specified based on the improvement goals of the organization
- Defined with respect to:
object, purpose, quality focus, from a viewpoint, relative to a particular context

• **Question:** Operationally defines GQM goal

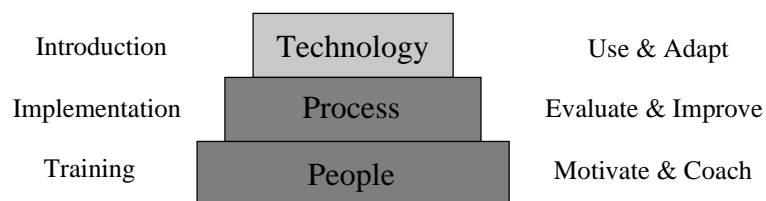
• **Model:** Specifies how to combine & compute measurement data to answer a question

• **Metric:** Associated with question(s) to answer them in a quantitative way

Measurement Goals for Allianz

- Characterize the inspection approach
 - effectiveness (i.e., its capability to find defects)
 - efficiency (i.e., its cost-effectiveness)
 - factors impacting effectiveness and efficiency
 - cost-benefits
- Characterize the defect slippage
 - Do inspections decrease analysis and design defects detected in testing?
- Characterize the project's effort distribution
 - Do inspections reduce the testing and rework effort?
- Evaluate Defect Content Models
 - Can they be applied within Allianz?

Procedure



The people

- teach the technology and motivate all participants
- no boss as a participant in the inspection meeting
- no personnel data evaluations (base of confidence)
- do not detect defects, but gain findings (-> positive atmosphere)
- only interpret data evaluations together with the project members
- “Everybody is a winner, nobody a loser!”



Procedure

The process (what)

➤ The inspection process:

- early planning inspections (project deliverables, perspectives, deadlines)
- developing and concluding the scenario for each perspective
- preparation of each inspection by moderator, author, and inspectors
- performing the inspection meeting and finishing the inspection



➤ The evaluation process:

- developing the GQM Plan
- investigating, collecting, and evaluating data concerning the GQM Plan
- feedback sessions together with the project members
- store new experience in our experience base for future reuse

The technology (how)

- perspective-based inspections; goal-oriented measurement

Lessons Learned

Initial experiences regarding the technology

- understandable and practice-oriented technology transfer
- especially suitable for deliverables with different perspectives

Initial experiences regarding the process

- scenarios especially helpful for inspectors with less experience
- all perspectives at the same time at one table !
- templates for inspection minutes & data investigation very helpful
- considerable additional effort for project & coaching team



Initial experiences regarding the people

- early commitment with the management to get qualified inspectors
- constructive atmosphere during the inspection meeting
- acceptance of later users increases considerably by involving them early
- motivation of all project members through high identification with the results

Initial Results

- **Four inspections of user-interface descriptions**
 - distribution of product related findings according to
 - severity: 78.3% are *very critical* (A) or *critical* (B) findings
 - impacts*: user-friendliness 74.5%, functionality/correctness 19.1%, standards/directions 6.4%
 - causes*: ambiguous 29.8%, alternative 21.3%, new 19.1%, extraneous 17%, others 12.8%
 - effectiveness: 2 findings*/page; efficiency: 2.2 findings*/person day
 - initial effort - benefit (→ rate 1: 1.8) :
 - 28.1 person days (inspection effort)
 - 51.5 person days (primary benefit = estimated effort savings)
 - additional benefits: esp. *higher user satisfaction, relief of the call-center*
 - Defect Content Models: - Detection Profile Method provides plausible estimates



Legend: * (findings of severity A + B)

Outlook

Further actions

- perform further inspections
- continue measurement program concerning the GQM Plan
- perform overall data evaluations at the end of the project regarding to ROI, defect slippage to testing, rework effort, testing effort, overall effort, user satisfaction, etc
- interpret analysis results together with the project team
- gather experience & continue "lessons learned"
- store our experience in our experience base for future reuse
- develop experience package for measurement and PBR inspections
- use the results from the HYPER project for optimizing the inspection process and for broader use of PBR inspections
- publish our results and experiences internally and externally



Quality Assurance Technologies for the EURO Conversion – Industrial Experience at Allianz Life Assurance

Lionel Briand, Bernd Freimut, Oliver Laitenberger, Günther Ruhe
Fraunhofer IESE
Sauerwiesen 6
D-67661 Kaiserslautern
{briand, freimut, laiten, ruhe}@iese.fhg.de

Brigitte Klein
Allianz Lebensversicherungs-AG
Reinsburgstrasse 19
D-70718 Stuttgart
Brigitte.Klein@Allianz.de

Abstract

Early software quality assurance is essential to mastering EURO conversion projects. Allianz Life Assurance sets a specific priority for it by using innovative software inspection technologies, that is, Perspective-based inspections, in the early phases. In addition, quantitative models are used to control the defect content of the inspected artifacts as well as to assess the perspective-based inspection approach. This paper describes these technologies, their transfer to Allianz Life Assurance, and their impact on software quality.

1 Introduction

The unified European currency (EURO) implies substantial technical changes to existing software systems. Each change in a software system may inject new defects, which decreases its quality. A failure resulting from a low quality EURO adaptation would have substantial consequences for a company. Among others, the company's image and reputations would be damaged. Apart from high quality, a further pressing requirement is that the EURO transition must be completed by December 1st, 1999 - the end of the financial year. This deadline causes enormous time pressure, which is considered a prevalent defect driver in software projects.

The Allianz Life Assurance, which is a member of the worldwide Allianz Group and market leader of life-insurance companies in Germany set up a strategy for adapting their software system to the EURO. In 1995, a working group was founded with the task of intensively investigating the effects of the unified European currency on the financial markets. Another project group dealing with the technical and organizational effects was created in 1996. This project group listed a number of measures that are necessary in connection with the introduction of the unified European currency. Finally, Allianz Life founded a EURO Core Team for coordinating the internal adaptation process. The core team consists of staff members of those areas of the company that are affected by the EURO introduction. This includes almost all areas of the company, since insurance products and their rates are just as much affected by the unified currency as payment transactions, billing, and sales.

Within the EURO conversion project at Allianz Life, there is a specific task force in charge of maintaining the high quality of its existing software systems. It investigates innovative quality assurance technologies in the framework of the ESSI Process Improvement Experiment (PIE) "HYPER" (project number 27839). This ESSI PIE is part of an overall software quality improvement program performed at Allianz Life Assurance. The results of the PIE will be included in the entire improvement program so that future projects can benefit from them.

The Fraunhofer Institute for Experimental Software Engineering (FHG IESE) is a subcontractor in the ESSI PIE. FhG IESE is one of the leading institutions in the area of applied software engineering research and technology transfer. It provides expertise in innovative quality assurance technologies as well as in their qualitative and quantitative evaluation.

The transfer of innovative software inspection technologies to the Allianz EURO conversion projects represents the core of the ESSI PIE. Software Inspection is an industry-proven best practice for software quality assurance. They can have a resounding effect on reducing rework cost and delivery time, because a reduced number of defects slip through successive development phases. This is especially the case when software artifacts developed early in the life cycle (e.g., requirements and design artifacts) are inspected. However, to exploit their full potential, software inspection must call for a close and strict examination of the inspected artifact. This requires systematic reading techniques that tell inspection participants what to look for and more important how to scrutinize a software artifact for defects.

However, existing inspection approaches often lack systematic reading techniques. To overcome this limitation, Allianz Life Assurance is performing Perspective-based inspections on their requirements and design artifacts. Perspective-based inspections leverage the basic inspection approach of Fagan and others [Fagan, 1976] [Gilb and Graham, 1993] with the Perspective-based reading

technique (PBR) [Basili et al., 1996]. PBR defines specific inspection viewpoints from which to inspect an artifact. In addition, PBR provides guidance for inspection participants on how to scrutinize a software artifact in a systematic manner.

Since Allianz Life Assurance achieved promising results with Perspective-based inspection in previous projects, they decided to investigate its effect on artifact and process quality in the context of the EURO conversion project. The main objective of this investigation is to optimize their inspection parameters, such as effectiveness (i.e., the capability of detecting defects) and efficiency (i.e., the cost-effectiveness) of Perspective-based inspections.

The vehicle for this investigation is goal-oriented measurement according to the Goal/Question/Metric paradigm (GQM) [Briand et. al.,1997a], [Basili et. al.,1994]. Based on well-defined, company-specific measurement goals, the interesting quality aspects (e.g., inspection effort, testing effort, defects and their cause, characterization of inspected artifacts) as well as the qualitative and quantitative description of context factors that may impact the quality aspects (e.g., size, experience of inspectors) are defined. These definitions are refined into measures to be collected during the measurement program. Analysis and interpretation of the collected inspection data is based on the characterization and understanding of the organizational context and active participation of project team members in feedback sessions, where the measurement results are discussed.

To further exploit the collected inspection data for practical application, quantitative models are built to help manage and optimize Perspective-based inspections. Furthermore, a Return-On-Investment (ROI) model is developed to evaluate this inspection approach. The model helps assess whether costs in terms of the up-front investments in Perspective-based inspection remain significantly lower than benefit in terms of both reduced rework and testing effort. In addition, defect content models are developed to control inspections. This kind of models, such as Capture-Recapture Models [Eick et. al., 1992][Briand et. al., 1997b], are used to determine after an inspection how many defects still remain in the inspected artifact. Based upon the number of remaining defects, one can decide whether the document has to be inspected for a second time. Thus, the decision about re-inspection is objective and based on tangible information.

This paper describes each of these technologies in more detail. Moreover, it presents initial empirical results regarding the application of Perspective-based inspections on requirements documents in the context of a EURO adaptation project at Allianz Life. Experiences and lessons learned regarding the technology transfer of innovative inspection technologies are identified. This may be beneficial to software development projects in general and to other EURO conversion projects in particular.

2 The EURO Conversion - Challenge for Software Development at Allianz Life

2.1 Allianz Life Assurance and its IT Department

Allianz Life Assurance, the market leader of life insurance companies in Germany, is part of the Allianz Group, which has become the largest insurance group in the world. Allianz Life currently has about 5,500 employees and about 200 trainees. The premium income in 1997 was about 12.5 billion DEM, the insured sum 305 billion DEM with a total number of 7,723,000 insurance policies.

The IT Department of Allianz Life consists of more than 500 employees, with 350 of them being application developers. As one of the earliest users and innovators of IT Technology, Allianz Life has a long and very successful tradition of developing commercial software mainly in the area of online-transaction applications with very large databases.

Allianz Life has about 8,500 programs written in PL/I, C, or Assembler. These programs are mainly running on a Host environment. Client-server architectures have only recently been established to achieve higher efficiency and flexibility of development processes. Currently, Allianz Life is building up a new generation of client-server applications. Another strategic task is to increase the portability of our systems, due to the demand for higher mobility for improved customer satisfaction.

The Euro conversion costs are estimated to be about 30 million DEM, of which 70% is going to be EDP adaptation. The total amount of costs for all Allianz companies in Germany will be approximately 110 million DEM. These amounts, however, also contain the investments to be made for the Y2K conversion. Since all software systems must be checked anyway, there will be synergistic effects.

2.2 The EURO Conversion: Characteristics and Challenge

Ever since July 1st, 1994 the European domestic market for insurance companies has been a reality in Germany. The introduction of the EURO will further increase price transparency and competition in Europe. At the same time, there will be greater differences in the quality of products. In the future,

quality, size, and international orientation of a company will also provide even greater competitive advantages on a common European market.

The three-year transition period to the EURO gives companies the chance to keep conversions flexible. Companies will be affected by the changes in payment transactions and in financial investment activities directly on January 1st, 1999. It is true though that, at the beginning, the EURO will only exist as accounting currency.

But there is much more: The entire stock of forms, conditions, stipulations, applications, brochures, and job instructions must be checked for changes that may be required. The three-year transition period is therefore appropriate.

Allianz Life Assurance will not convert to the new currency at once. By law, we are required to convert our contracts by January 1st, 2002. The asset conversion to EURO will be done precisely to the Cent. If it is necessary to round or even any amounts, it will be done to the benefit of our customers.

If a customer desires to have his contract converted at an earlier date, we will, of course, comply with that wish. Starting on January 1st, 1999, our payment systems will be multi-currency, meaning that payments can be made either in DEM or in EURO - independent of the currency of the contract. Thus, it is not necessary to adjust the insurance contract, if the customer's bank account is already converted to EURO. The stipulations of the contract will continue to be valid to the complete extent after the conversion of the insurance policy inventory to EURO.

In the transition phase, insurance policies may still be concluded on a DEM basis. Starting on January 1st, 1999 we will also offer our customers the option of entering into new insurance contracts in EURO. At first, the amounts will only be converted to EURO on the basis of a DEM rate catalog. A EURO rate catalog is expected to be available no later than the year 2000. In the EURO rate catalog, conditions and rates will be reflected in whole EURO amounts, i.e., they will also be re-calculated and re-established. In doing so, we pursue a customer-oriented business policy.

Additionally, Allianz Life Assurance has to adapt applications for the KAG (Allianz Investment Trust), a subsidiary of the Allianz Group, which will also offer their services in EURO or DEM at the beginning of 1999. Then, starting December 1st, 1999 at the latest, all customer accounts will be carried in EURO.

First of all, we derived a strategy regarding how to implement the new European currency in our systems to fulfill all customer needs. This concept is very important and determines the flexibility of our systems in the future. Thus, we need to achieve a very high quality in implementing the customer requirements.

The EURO conversion project represents an effort of about 300 person months and has started at the beginning of 1998. It involves about 100 developers from a number of different departments across the entire organization. To master the complexity and state of the conversion, the project will be performed in several stages.

About 250 current applications in programming languages like PL/I, C and Assembler, 550 amount fields and interfaces of 14 neighboring information systems have to be adapted to the new currency.

Competition on the insurance market is largely based on quality and functionality of information systems. In the past, IT Technology supported mainly the administrative parts. Quality was defined in terms like performance, reliability, effort of development and operation, etc. Today, competition is much stronger and services and products must be adjusted to the requirements of the changing markets. Therefore, criteria like time-to-market, flexibility, ease-of-use, etc. have a much higher significance.

Due to this strategic importance of the EURO Conversion for Allianz Life Assurance, the high quality of software products is an absolute necessity.

2.3 Improvement Approach at Allianz: Software Process Improvement Program and Results

The first initiative to improve quality assurance was launched in 1988. Allianz Life applied walk-throughs and inspections at different stages of the development process. However, a number of issues were raised. Primarily, there was a deficit in the technological expertise including knowledge on how to optimally adapt these techniques to the organization. Other reasons were insufficient preparation for application of the different quality assurance techniques, a lack of training and motivation of the participating persons, and a disappointing cost-benefit ratio.

The next step in the software process improvement program was initiated in 1993. The analysis had shown that technology transfer requires more detailed knowledge about software development processes. Allianz Life understood that process changes must be driven and accompanied by

- specific goals formulated in quantitative form
- an appropriate characterization of the environment (organization, project)
- the definition of essential product, process, and context attributes, and

- the application of an evolutionary learning cycle following the experimental approach.

In strong collaboration with the Software Technology Transfer Initiative Kaiserslautern, Allianz Life initiated a 'Goal-oriented measurement initiative' [Günther et. al. 1996],[Leippert and Ruhe,1998].

The overall framework of all subsequent improvement activities was based on the following principal steps:

- quantify the quality of products and processes by application of goal-oriented measurement,
- better understand the current situation and its causes,
- identify and implement improvements,
- detect, formalize and structure experience,
- continuously improve the maturity of processes and products.

For an initial set of three baseline projects, measurement programs based on the Goal-Question-Metrics paradigm were established to better understand and analyze effort, stability of requirements, and maintenance. What did Allianz Life learn from the measurement programs? Among other things, Allianz Life found that

- the importance of testing for quality assurance is overestimated (too much effort invested),
- communication and common understanding between all departments participating in a project is currently a weakness and has to be improved,
- too many defects are introduced in early stages (requirements analysis, design) are detected during testing,
- effectiveness and efficiency of verification (inspections) is currently low,
- the overall effort for testing (currently 30% of development effort) has to be reduced through early use of verification.

The current situation is characterized by possessing quantitatively based knowledge on the processes and products including success factors. This allows constructive guidance on how to improve existing software development processes. The HYPER project is considered as the initial step in this new area of controlled and experience-based process improvement actions.

2.4 The ESSI PIE Project 'HYPER': Motivation and Objectives

HYPER is an ESSI PIE project that is promoted by the European Commission. ESSI is an initiative for optimal software technologies and a domain of the ESPRIT program that joins together industry research & development projects and the steps for the application of software technologies. PIE stands for process improvement experiment, which aims at improving software development processes by applying software technologies.

HYPER is the abbreviation for "High Quality of Software Products by Early Use of Innovative Reading Techniques". It is an experiment based on the results of the software process improvement program. The project is devoted to the application of innovative and cost-efficient reading techniques to documents of the early phases of the life-cycle (requirements analysis and design). The baseline project for applying and evaluating inspections with innovative reading techniques is the EURO Conversion project. For participating in the HYPER experiment, the entire EURO Conversion project would be much too extensive; therefore one important part of it has been selected.

Software Inspections with innovative reading techniques have shown to be a very effective way of detecting defects early in the development process. Thus, substantial effort can be saved, since defects detected later are known to be significantly more expensive.

The HYPER project started on June 1st, 1998 and has a duration of 18 months. Hence, it will be finished on November 30th, 1999. The planned effort for the Hyper project amounts to 223 person days. The project is coordinated by the IT Department of Allianz Life and supported by the Fraunhofer Institute for Experimental Software Engineering (FhG IESE) with up-to-date knowledge and experience on innovative reading techniques and quality control techniques. Motivation for the HYPER project from the business point of view comes from the urgent need

- to improve quality assurance through the use of innovative software inspection techniques,
- to achieve higher quality and stability of requirements,
- to contribute to better customer satisfaction by early verification of original requirements,
- to better control the overall verification and validation process, resulting in better resource usage,
- to enhance and improve communication and common understanding between the different departments involved in product development,
- to use the results from the baseline project as starting point for broader use of innovative reading techniques within an organization-wide software improvement initiative.

From the technical point of view, the objective of the experiment is to investigate and establish customer-specific scenario-based reading techniques as an essential element of existing software inspection approaches. Quality control techniques and goal-oriented measurement is used to demonstrate quantitatively that proper usage of a systematic reading technique, such as perspective-based reading (PBR),

- improves productivity by finding defects when they are less expensive to correct,
- reduces the number of defects originating in phases prior to the ones where they are detected,
- reduces both test and overall project effort, and
- improves customer satisfaction.

2.5 *The Technology Provider: Fraunhofer Institute for Experimental Software Engineering*

The Fraunhofer-Gesellschaft (FhG) e.V. is the largest funding organization for applied research and technology transfer in Germany. The Fraunhofer-Institute for Experimental Software Engineering in Kaiserslautern, Germany, founded in January 1996, is headed by Prof. Dr. Dieter Rombach, and concentrates on applied research and technology transfer in various areas of software engineering. Additionally, IESE is affiliated with the Fraunhofer-Center Maryland (FC-MD) to work together on projects with multinational as well as US national organizations.

Experimental Software Engineering employs experiments as an instrument for software technology transfer. Based on the recognition that well-understood and quantitatively manageable software development and maintenance processes need to be customized to a company's specific business goals and characteristics, new and innovative software technologies need to be carefully evaluated before being transferred into practice. After transfer, they need to be continuously optimized based on feedback gained from measurements.

Our customers are companies from many different branches, of any size, and from a large number of countries. In order to service such a large variety of customers, we have increased our efforts in building up domain knowledge in key application areas such as telecommunications, automotive systems, and banking/insurance/trade, formed a separate service center for small and medium-size companies, and hired scientists from foreign countries to staff international customer projects.

Transfer of advanced industrial-strength software engineering technologies is the central task of the Fraunhofer IESE. We therefore maintain a transfer-oriented network of collaborations with technology providers, such as universities, with research and development departments of large organizations, with providers of tools that support our technologies, and with strategic partners that otherwise support our work. Competence gained from collaboration with these providers enables the IESE to conduct technology transfer projects with customers, i.e., with the users of our technology. On the technology side, we have to monitor the latest developments, identify promising technologies, and experimentally evaluate and improve them to create industrial-strength technologies. On the customer side, our competencies are to identify strengths and weaknesses of organizations, to define strategic improvement goals with our customers, to implement continuous improvement programs, to set up means to monitor progress of the changes introduced, and to capture and store experiences made.

The cooperation partners of the Fraunhofer IESE range from very large global players to very small companies. They can be roughly grouped into four categories:

- Large national and international companies that seek help in their mid- to long-term endeavor of quality improvement in software development.
- Large national and international companies that can afford their own R & D departments and that search for competent research partners.
- Medium-size companies that want to set up improvement programs but are usually under very tight budget and schedule constraints.
- Small companies that need ready-to-use, evaluated technologies that yield short-term return on investment.

In addition to bilateral cooperation, in 1997, the IESE has started a multi-national consortium of international companies that team up in this joint endeavor to advance their software engineering competence on a global scale, i.e., across different sites and business units and in collaboration with other leading companies in the scene as well as other application domains.

3 Meeting the Euro-Conversion Challenge with Technologies for Early Quality Assurance

3.1 Motivation for Early Quality Improvement

The development of software artifacts involves a series of development activities in which there are many opportunities for the injection of defects. Defects may begin to occur at the very beginning of the development process when the requirements of a software artifact may be erroneously or imperfectly specified, as well as during the later design and development stages when these requirements are implemented.

Many techniques have been proposed to reduce the software defect rate. Amongst them are techniques that help software developers prevent defects, such as formal methods, and others that help them detect and remove existing defects. The most prominent approach of the latter is testing. Testing is confined to a stage immediately prior to operation and maintenance. However, severe consequences can result when defect detection and correction activities are confined to the later stages of development. First, even with the best testing techniques, it is not likely that a software product will become defect free before operation. Second, late defect detection and correction has a negative effect on cost. This is because the later in the life cycle a defect is found, the higher is the cost of its correction. Consequently, if higher quality and lower cost are the goals, defect detection and correction should not be isolated to a stage at the end of the development process but should be incorporated into each phase of the development process. This must be particularly the case for early development phases. Barry Boehm [Boehm, 1981] has stated that one of the most prevalent and costly mistakes made in software projects today is deferring the activity of detecting and correcting software problems until late in the project. Hence, the prevalent reason for early investment in quality is to catch severe defects early before the cost of their correction escalates or threatens the completion of the project.

Software inspection has emerged as one of the best techniques to detect and remove defects early in the development cycle. In its full generality, a software inspection is a structured process in which a peer group rigorously examines the description of a software artifact, looking for possible defects. After Fagan's seminal work in 1976, Fagan, and others, have shown that software inspection can lead to the detection and correction of anywhere between 50 and 90 percent of the defects in a software artifact [Fagan, 1976], [Gilb and Graham, 1993]. Moreover, since inspections can uncover defects shortly after they are introduced, rework costs (i.e., the costs associated with correcting defects) are considerably reduced.

3.2 The Technologies: Perspective-Based Inspections and Statistical Quality Control

3.2.1 Description of Perspective-based Inspections

3.2.1.1 Scope of Perspective-based Inspections

Software inspection in general and perspective-based inspection in particular consists of numerous activities including planning, defect detection, defect collection, and defect correction. As depicted in Figure 1, the planning activity is performed by the organizer who is responsible for setting up an inspection for a particular software artifact. Throughout the defect detection step inspectors individually scrutinize a software artifact for potential defects using a particular reading technique (i.e., the Perspective-based reading technique). Inspectors document all potential defects they find on a

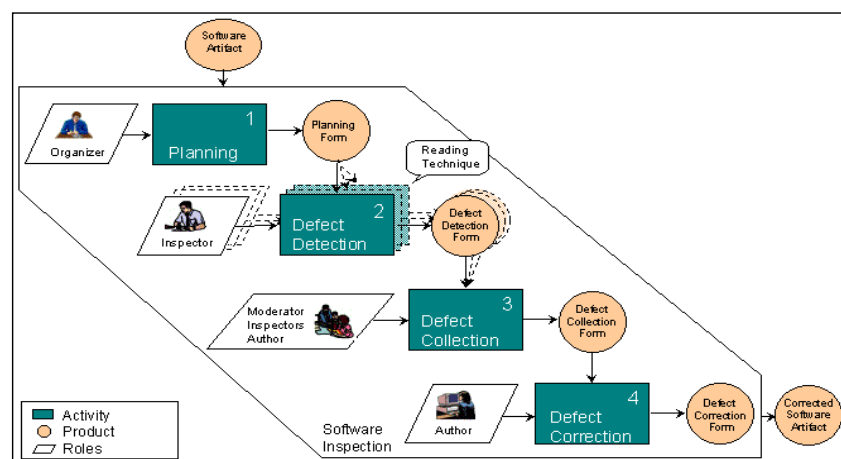


Figure 1 Software Inspection

defect report form. As some of the potential defects documented on the defect report forms might prove not to be real defects, inspectors together with the author and a moderator perform an inspection meeting. The goal of the inspection meeting is to decide upon which of the potential defects are real ones. In addition, new defects might be detected during the inspection meeting. Defect detection, however, is not the primary goal of this meeting. Throughout the meeting, one of its participants documents all real defects on a meeting report form. In the final activity, the author corrects the real defects.

Although each of these activities are vital for an effective inspection, it is the defect detection activity, or "reading" as it is commonly called, that is considered the key part of an inspection [Basili et. al., 1997] and which therefore needs to be supported with adequate reading techniques.

Several kinds of reading techniques have been defined in the literature, the simplest of which is the ad-hoc reading approach. As its name implies, this technique provides no explicit advice as to how to proceed, or what specifically to look for, during the reading activity, so inspectors must resort to their own intuition- and experience-based heuristics to determine how to go about an inspection. A significant improvement over the ad-hoc approach is the so-called checklist approach [Fagan, 1976], in which an inspector is at least given a list of questions to answer. The checklist-based technique thus gives inspectors advice about what to look for in an inspection.

The next level of sophistication is offered by scenario-based reading techniques [Basili et. al., 1997]. The basic idea of a scenario-based reading technique is the use of so called scenarios that describe how to go about finding the required information, as well as what that information should look like. In doing so, scenario-based reading techniques assign clear responsibilities to inspectors and require each of them to take an active role in an inspection. In these two ways, they are similar to active design reviews suggested by Parnas and Weiss [Parnas and Weiss, 1987] for the inspection of design artifacts. However, active design reviews provide little if any guidance to inspectors on how to perform the reading activity.

Of the several forms of scenario-based reading techniques defined to date [Porter et al., 1995], [Cheng and Jeffrey, 1996], [Basili et al., 1996], experiments have shown the Perspective-Based Reading (PBR) technique to be among the most effective. The basic idea behind this approach is to inspect an artifact from the perspectives of its individual stakeholders, with the assumption that collectively these will increase the coverage of the defect space. Such a viewpoint-oriented approach follows the current thinking on quality: everybody, even someone internal to an analysis, design, or coding process, is considered a customer (i.e., stakeholder) and also has customers. Since different stakeholders are interested in different quality factors or see the same quality factor quite differently, a software artifact needs to be inspected from each stakeholders viewpoint.

3.2.1.2 Description of Perspective-based Reading

3.2.1.2.1 Goal of Perspective-based Reading

The basic goal of PBR is to examine the various descriptions of a software artifact from the perspectives of the artifact's various stakeholders for the purpose of identifying defects. Each software artifact is inspected from the perspective of each stakeholder involved in the software lifecycle in such a way as to determine if the descriptions satisfy the stakeholders' particular needs. A stakeholder may be, for example, an external customer who wants to ensure the completeness of the inspected requirements document. If each description of the artifact meets the stakeholders' quality requirements, the end product, that is the final software artifact will meet the specified quality goals. An inspector in a perspective-based inspection reads the inspected descriptions from the perspective of a particular stakeholder. The reading process is driven by perspective-based scenario.

3.2.1.2.2 Perspective-based Reading Scenarios

Throughout the reading process, inspectors follow a perspective-based reading scenario (in short: scenario). A scenario tells an inspector how to go about reading an artifact from one particular perspective and what to look for.

As shown in Figure 2, the scenario consists of an introduction, instructions, and questions framed together in a procedural manner. The introductory part describes the stakeholder's interest in the artifact and explains the quality factors most relevant for this perspective. The instruction part describes what kind of descriptions an inspector is to use, how to read the descriptions, and how to extract the appropriate information from them. While identifying, reading, and extracting information, inspectors may already detect some defects. However, the motivation for providing guidance for inspectors in form of instructions on how to perform the reading activity is three-fold. First, instructions help an inspector gain a focused understanding of the artifact. Understanding involves the assignment of meaning to a particular description and is a necessary prerequisite for detecting more subtle defects,

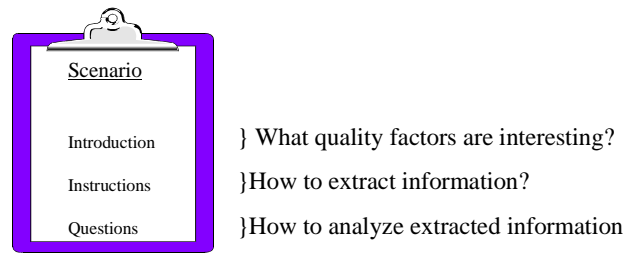


Figure 2 Contents of a scenario

which are often the expensive ones to remove if detected in later development phases. Second, the instructions require an inspector to actively work with the descriptions. Third, the attention of an inspector is focused on the most relevant information, which avoids the swamping of inspectors with unnecessary details.

Once an inspector has achieved an understanding of the artifact, he or she can examine and judge whether the artifact as described fulfills the required quality factors. For making this judgement an inspector is supported by a set of questions which are answered while following the instructions. Following a scenario in this manner is in line with the natural definition of "reading", which is the systematic examination of an artifact's description to gain certain information for a particular purpose.

Figure 3 shows an example for reading from the perspective of a tester.

However, there is not one best way to do Perspective-based reading and, more general, Perspective-based inspection. To be most successful, the Perspective-based inspection approach needs to be tailored to the particular development environment, such as Allianz. This will be described in a later section.

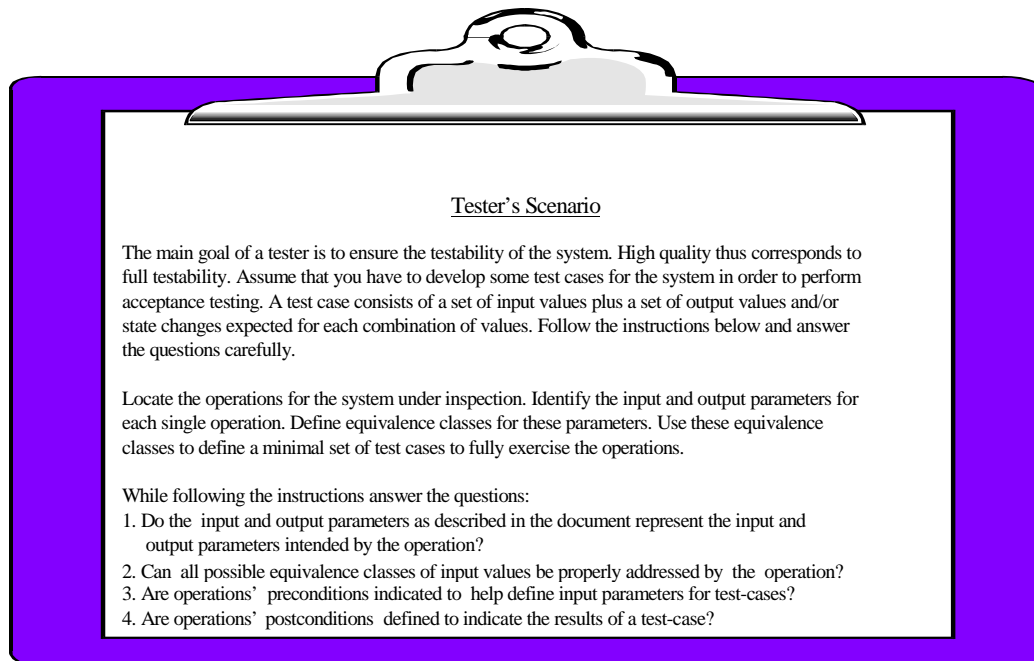


Figure 3 Reading from a tester's perspective

3.2.2 Statistical Quality Control

3.2.2.1 When is an Inspection completed? – Defect Content Estimation

One of the inspections main benefits is that they detect defects before they propagate to subsequent development phases and into the field where they cause high rework expenditures. In practice, however, it has been shown that the effectiveness of inspections can vary widely. To reduce the variability, quality control techniques can be applied. Quality control in this sense means that an inspection is considered complete only if the document has passed a certain quality threshold. Many researchers and companies have defined this threshold in terms of the number of defects *detected*. However, a more appropriate threshold would be the number of defects *remaining*.

Although quite intuitive, this quality definition imposes a practical problem: In order to compute the number of remaining defects we have to know the actual number of defects. Defect Content Models aim at *estimating* this total number of defects.

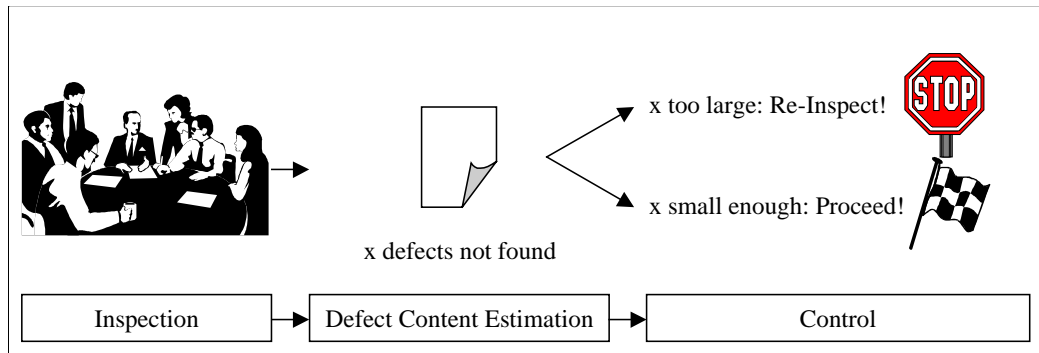


Figure 4 Controlling inspection with Defect Content Estimation

Based upon the data collected during an inspection, the total number of defects is estimated. After subtracting the number of defects found, an estimate for the number of remaining defects can be obtained. If this estimate indicates that most of the defects have been detected by the inspection, the document is considered to be of a sufficient level of quality. On the other hand, if the estimate indicates that many defects remain in the document, a second inspection of the document should be performed.

3.2.2.2 Defect Content Estimation Methods

Basically, existing defect content estimation methods can be classified into two approaches: Capture-Recapture Models [Briand et. al., 1997b] and Graphical Approaches points [Wohlin and Runeson, 1998]. Capture-Recapture Models have their origin in biology and wildlife research. They are used to estimate the size of animal populations. In these disciplines it is often practically difficult or impossible to track and record all animals. Thus, population size has to be estimated based on several incomplete samples of the population.

Transferred to inspections the basic concept can be described as follows. Suppose two inspectors inspect a software artifact with a total of N defects. The first inspector detects n_1 of these defects while the second inspector detects n_2 of these defects. Usually, both inspectors do not detect exactly the same defects, thus let m_2 be the number of defects detected by both inspectors.

If we now assume, that each inspector has a probability p_i ($i=1,2$) of detecting defects, we have $E(n_1)=Np_1$ and $E(m_2)=Np_1p_2$, where $E(x)$ denotes the expected value of x . Thus, we can denote N as

$$N = \frac{E(n_1) \cdot E(n_2)}{E(m_2)}$$

and derive an estimator for the number of defects as

$$\hat{N} = \frac{n_1 \cdot n_2}{m_2}$$

This estimator is known in biology and wildlife research as Lincoln-Peterson Estimator ([Otis et. al., 1978]).

Various models with different assumptions on the kind of defects (do they have the same probability of being detected or do different defects have different probabilities?) and on the capabilities of the inspectors (do all inspectors have the same probability of finding defects or do different inspectors have different probabilities of finding defects) are suitable for inspections [Briand et. al., 1997b].

Alternatives for capture-recapture models are graphical approaches. The rationale behind these approaches is that defects are sorted to some criterion and a curve is fitted to the data points [Wohlin and Runeson, 1998]. The most mature of these approaches is the Extended Detection Profile Method (EDPM)[Briand et. al., 1998b]. For each defect, one calculates how many inspectors detect that defect. The defects are then sorted in descending order according to the number of inspectors detecting them, and are plotted in a graph as shown below.

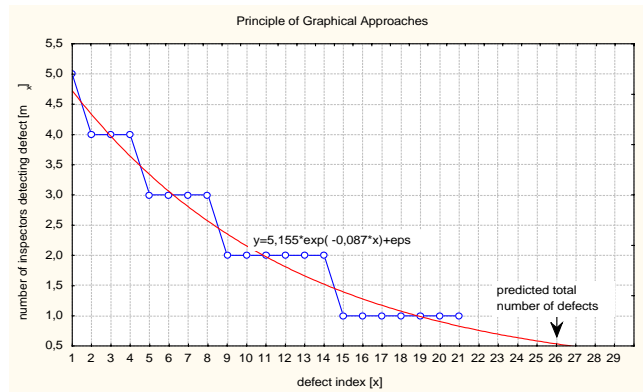


Figure 5 Detecion Profile Method

A curve is fitted through these data points and the total number of defects is determined by the largest defect index for which the curve is above 0.5. Depending on the shape of the data points, an exponential or linear curve is fitted.

The appealing properties of these approaches are that i) few and simple data have to be collected in order to apply these defect content models, ii) a decision on re-inspection can be made as soon as the inspection is finished, and iii) it is applicable to analysis and design inspections, thus allowing decisions on document quality to be made early in the life cycle [Cai, 1998].

3.2.2.3 Does the effort invested in inspections pay off? – Controlling inspections’ ROI

At first, organizations have to invest effort into inspections. Documents have to be distributed to inspectors, inspectors have to carefully analyze the document to be inspected, meetings with all inspectors, a moderator, and the author have to be held. This initial effort should, of course, pay off in later phases (e.g., by means of a reduced testing effort)

In order to determine the actual pay off in a quantitative manner, we have to define formally the cost-effectiveness, also called efficiency, of inspections. A definition of efficiency suitable to evaluate the introduction of inspections in an organization has been proposed by Kusumoto [Kusumoto et. al., 1992]. His definition can be interpreted as the proportion of testing cost saved due to the introduction of inspections.

If we consider inspections and testing as the only two defect detection activities, the efficiency definition can be explained as following:

During an inspection we detect N_{Insp} defects. What did we save by detecting these defects? If we had found these defects later in testing we would have had spent a testing effort of $N_{Insp} \times testcost$ for these defects, where *testcost* denotes the average cost to find and fix a defect during testing. However, this saved testing effort does not denote exactly the savings due to inspections. To compute the cost saved we have to take into account the cost of the inspection itself. This cost can be denoted as $N_{Insp} \times inspcost$, where *inspcost* denotes the average cost to find and fix a defect in an inspection. Thus, the total cost saved is denoted as $N_{Insp} \times testcost - N_{Insp} \times inspcost$. This cost saved is normalized by the effort required to find all defects during testing. If N_{total} denotes the total number of defects in the software system, the cost to find all defects during testing becomes $N_{total} \times testcost$.

Thus, the definition runs

$$\text{efficiency} = \frac{N_{Insp} \times testcost - N_{Insp} \times inspcost}{N_{total} \times testcost}$$

This definition can be generalized to any stages of defect detection activities such as unit test, integration test, and acceptance test

3.2.2.4 Adapting Efficiency Definitions to Real Projects

In many cases the application of models, such as the efficiency model presented above, to actual software projects is limited since not all required data can be possibly collected. For example, the efficiency model above requires the average cost to find and fix a defect during testing. However, the cost to fix a defect may not always be collected, which is the case for Allianz Life Assurance. Thus we have two possibilities: to choose a different efficiency definition usable with the collected data or to obtain the missing data by other means.

Our approach is to obtain missing actual data via subjective measurement based experts’ opinion. To perform this subjective measurement in a accurate manner, a number of well-known, thoroughly

studied heuristics are described in the literature regarding expert judgement [Vose, 1996]. One of them is referred to as *availability* and is defined as the ability of the expert to remember past occurrences of the events related to the quantity to estimate.

If we want to ensure that we will obtain an accurate efficiency estimates, we need to consider these heuristics to re-express the efficiency model above. In other words, this must be a function of parameters that can be assessed either based on existing, objective measurement or through expert assessment complying with the availability heuristic, that is relating to events being a regular part of the expert's life.

For example, the efficiency model above can be re-formulated as:

$$\text{efficiency} = \frac{N_{\text{Insp}}}{N_{\text{total}}} \times \left(1 - \frac{\text{inspcost}}{\text{testcost}}\right)$$

where *efficiency* is expressed in terms of data that can be objectively measured (i.e., the number of defects found during inspections N_{Insp} and the total number of defects detected N_{total}) and a ratio that is determined by subjective measurement (i.e., the effort/cost ratio between finding and fixing a defect during inspections and test).

Such a ratio complies with the availability heuristic since it is strongly related to the everyday developer's experience of isolating and fixing defects throughout the project life cycle. We therefore expect experts to provide us with reasonably accurate estimates for this ratio. However, we also have to take into account the inherent uncertainty associated with such estimates, for example by asking the expert to provide minimum, most likely, and maximum values. These values are then converted to a probability distribution (e.g., a triangular distribution) over the *inspcost/testcost* range. Then, using objective measurement for N_{Insp} and N_{total} , and the probability distribution of the *inspcost/testcost* ratio, we can use Monte-Carlo simulation [Vose, 1996] to obtain an efficiency probability distribution where the larger the variance, the larger the uncertainty of our estimate regarding efficiency.

The whole efficiency estimation process is illustrated in Figure 6 where X1 and X2 denotes the input parameters of the efficiency model: $X1 = N_{\text{Insp}}/N_{\text{total}}$, $X2 = \text{inspcost}/\text{testcost}$.

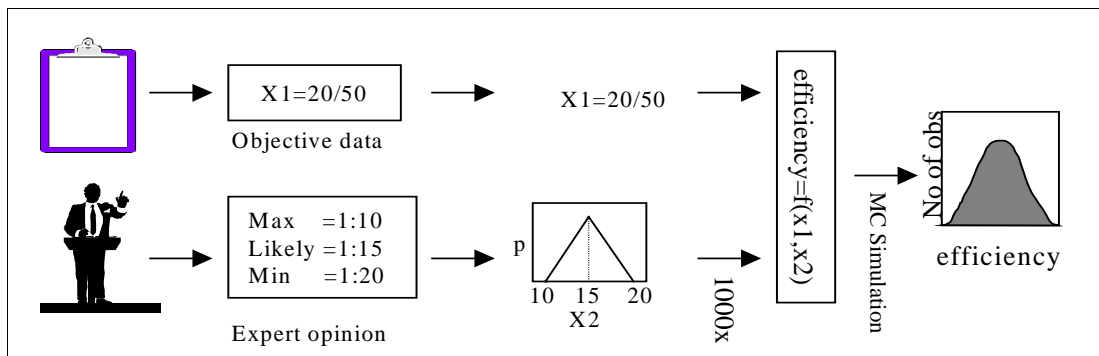


Figure 6 Evaluating cost-effectiveness using experts' opinion

3.2.3 How to collect and analyze data: The GQM-Approach

Beside the introduction of the inspection technology, we want to characterize in a quantified manner the inspection process itself and its impact on the development process. As basis for this quantification we introduce GQM-based measurement programs. The GQM approach provides guidelines for defining measurement goals, refining them into measurable entities, and providing a context for data analysis and interpretation. Two major processes characterize this approach. First, the explicit measurement goals are refined in a top-down manner into measures via questions and models tailored to the environment where measurement takes place. Second, the collected data are interpreted in a bottom-up manner in the context of the defined models and measurement goals. During both processes, the expected stakeholders are actively involved in the definition of measurement goals, the derivation of measures, and in the interpretation of measurement results.

The GQM-approach consists of six steps and specifies how to plan, implement, and analyze GQM-based measurement programs:

1. *Pre-study*: In this step the organizational context is characterized, a suitable pilot project is selected, and training and motivation for the measurement program is performed.
2. *Identification of GQM goals and definition of GQM plans*: The GQM goal describes for which purpose the measurement program is performed. With this explicit goal, the measurement program is given in a clear context. A GQM goal is defined by considering five aspects: what

- object is investigated, why the object is investigated, which property of the object (quality focus) is under study, who is going to use the measurement results, and in which environment the measurement takes place. A set of questions and models is derived that refine the measurement goal. Finally, a set of measures is derived to address the questions and models.
3. *Development of measurement plan:* The measurement plan defines for the measures identified in the GQM plan by whom, how, and when the measures should be collected.
 4. *Data Collection, analysis, and interpretation:* Data validation and analysis are performed by measurement experts. The interpretation of the analysis' results is performed in feedback-sessions and in close collaboration with the project team. This interpretation is used to interpret trends, take corrective actions concerning the project, process, or measurement program.
 5. *Post-mortem analysis of data:* The analysis of the completed project is performed and feedback provided to the organization.
 6. *Packaging:* The experience and findings gained from the measurement program are stored in an experience base to provide support for future projects. Examples for such experience packages might be baselines for defect distributions or lessons learned.

3.2.4 Synergy Effect of the Technologies

Combining Perspective-based inspection with methods for statistical quality control offers unique synergy effects for software process improvement. Since Perspective-based inspections follow a disciplined and structured process, they provide the vehicle for collecting high quality defect data and data on the inspections themselves. The Goal-Question-Metric paradigm can be applied to structure the data collection effort. GQM offers a systematic approach to translate measurement needs into models and metrics. The collected data can then be exploited by using statistical quality control techniques, such as Defect Content Models. The combination of techniques, therefore, offers some strong benefits. First, when some of these other statistical quality control and measurement techniques are implemented, the data collected by perspective-based inspections can be used to quantify quality improvement. Second, such techniques help predict latent problems, that is, those that have not been identified in the inspected artifact yet. Hence, a decision can be made, on an objective basis, regarding whether to re-inspect a certain document or not. Finally, inspection data has value beyond correcting the defects in the work product inspected. They may be analyzed for the purpose of process improvement, a usage which has been exploited for defect prevention, for example, at IBM [Jones, 1985]. There, statistics of the types of defects that are found can suggest reasons why certain defects occur, and indicate potential areas of improvement like better training for programmers or clearer documentation.

3.3 Technology Transfer to Allianz

3.3.1 Inspection Training

Introducing a software engineering best practice, such as perspective-based inspection, within an established development environment is a challenging endeavor. The challenge is to reach the critical mass of skill and motivation that is necessary to sustain a consistent and effective use of the new process, as a standard part of the development activity. To provide the motivation for perspective-based inspection, FhG IESE performed two training sessions at Allianz in which the perspective-based inspection approach has been introduced and explained to the training participants. Each training session consisted of three main parts: Motivation, inspection principles, and perspective-based reading.

The first part, that is, motivation, explained the reasons for performing perspective-based inspections at Allianz. It consisted of a more company specific part performed by Allianz and a more general motivation part performed by FhG IESE. In the company specific part, some initial results from the application of perspective-based inspection in a different Allianz project were already available and were used for this purpose. Among others, the more general motivation part explained the benefits of early defect detection to participants.

The second part, that is, inspection principles, explained the essentials of software inspection in terms of the activities performed (process), the roles that participants have in the process, the documents filled out throughout the process, and reading techniques. Finally, the perspective-based reading technique was explained to the training participants. After the training, each participant had acquired the knowledge to take an active role in a perspective-based inspection.

3.3.2 GQM Measurement Program

Besides the introduction of inspections, their impact on the software development process and the resulting software product has to be assessed in a quantitative manner. This assessment helps the

gauging whether the introduction of inspections was successful. It can also be used as a baseline for future projects, and it enables the identification of improvement opportunities.

For these reasons, a GQM based measurement program is conducted in the project implementing inspections. Explicit measurement goals were motivated by the expected benefits from inspections: Inspections are supposed to reduce the number of early defects detected late during testing decreasing the testing and rework effort. Additionally, the inspection process and its cost-benefit relationship are analyzed.

In close co-operation with FhG IESE, the process engineering group of Allianz, and the project leader of the involved project, a GQM plan was derived which

- (1) characterizes the inspection process in terms of its effectiveness (i.e., its ability to detect defects) and efficiency (i.e., its cost-effectiveness) and analyzes factors impacting these properties,
- (2) characterizes the proportion and kinds of defects that are detected by testing but are caused in the early development phases,
- (3) characterizes the effort distribution of the development project,
- (4) evaluates the cost-benefit relationship of the Perspective-based inspection approach,
- (5) evaluates the accuracy of defect content models.

The development of the GQM plan involved both selecting appropriate models such as the efficiency model presented in 3.2.2.3. and knowledge acquisition to capture the implicit quality models of the Allianz project participants.

In addition to the development of the GQM plan, the members of the development team are trained in the basic principles of the GQM approach. This is to ensure that the project members understand the rationale for their interpretation. This understanding is an essential prerequisite for motivating the development team members to provide the required data.

4 Application of the technology to the Euro conversion project

4.1 Current Status of *HYPER*

The initial phase of the project was devoted to describing the situation before the PIE, to train involved project team members in basic technologies, and to establish the measurement program for qualitative and quantitative analysis. During the kick-off meeting, objectives and basic activities of the project were discussed with the project team. Their motivation and understanding are an essential prerequisite for the success of the project.

Next, the inspection process had to be adapted and tailored to the environment of Allianz. This adaptation also involved the definition of user-scenarios, which take into account the specific context factors of Allianz and the development projects.

Based upon this inspection process, the first inspections were conducted. This enables us to present the experience gained during the first steps and to present initial analysis results.

4.2 Procedure concerning the People, the Process, and the Technology

The success of the technology transfer is not only influenced by the technologies themselves. It depends on several factors: the people, the process, and the technology. Each of these components has to be addressed.

The people

The key to a successful technology transfer is to have motivated people. The people have to be motivated to change to new procedures and processes and to apply them continuously.

Efficient and tailored transfer of the Perspective-based inspection approach and the goal-oriented measurement approach was crucial. The project members realized the benefits of inspections and the accompanying measurement program.

Beside the motivation aspect, the training primarily enables the project members to apply the technology. This is especially true for the inspection process. However, the training courses also are a first step to establish the related know-how for subsequent reuse in future projects.

Once the people are convinced and enabled to perform the inspection process, they must be continuously motivated to apply it further. Therefore, a very important issue is to establish a positive and constructive working atmosphere.

A cornerstone to do so is that people can express themselves without fear of possible negative consequences. Therefore, the inspection team should not include personnel at different hierarchical levels. Otherwise, the authors might be intimidated by the defects detected in their document. Inspectors could also be inhibited when their superior is present. They might be afraid to blame themselves or to contradict the statements of the superior.

To promote the positive atmosphere further, Allianz uses a special word for issues raised during an inspections: “finding” (German: Erkenntnis). This word is not a synonym for “defect”, because it comprises, on the one hand, defects in a narrower sense and on the other hand, questions, improvement proposals, and comments. Besides, the meaning of the word “finding” is entirely positive, so that negative associations do not arise at all. This has a favorable effect on the inspection meeting: Inspectors are not detecting defects, but collecting findings.

Another crucial issue for maintaining a positive atmosphere concerns the handling and usage of collected data. Once people fear that the data can be used against them, for example to evaluate their individual performance, they will stop providing accurate, complete data. Therefore, a trusting relationship has to be built up by assuring all project members that no personnel evaluation will take place. All analysis that will be made are anonymous and there should be no reason of worrying about negative personal consequences.

In this context it is also very important to interpret the data analysis results together with the project members. This is important since they are the only ones to know exactly the context in which to interpret the results. The project team thus has the guarantee that there will be no misguided interpretations with possibly negative impacts.

A way to involve the project members is the organization of feedback sessions, which are part of the GQM approach. With these feedback sessions, every project member is therefore integrated in the process improvement initiative from the beginning to the end. Additionally, the feedback sessions serve to collect valuable lessons learned regarding the performance of inspections.

As a conclusion, one can say that the success of Perspective-based inspections and its associated measurement program strongly depends on a win-win situation of all participants. Hence, we follow the principle “Everybody is a winner, nobody a loser” or as Tom DeMarco puts it “Blame the process, not the people!”.

The process

Besides motivating the people to adopt inspections, tailoring the inspection process to the specific Allianz environment is another essential issue. The Allianz inspection process is performed as follows:

At first, the inspections are planned early. Based on the project deliverables it is decided which of those documents are extremely important for the project and, therefore, selected to be inspected. For each document, the required perspectives and the meeting date are defined. For each perspective, the scenarios are developed for each document in cooperation with the representatives of the perspectives.

Based upon the devised planning, the moderator – normally the project leader –coordinates the inspection meeting. This coordination involves setting up the actual inspection date, selecting the inspectors, and selecting a room. One week before the meeting date, at the latest, the inspection papers (document to be inspected, target, scenario, and cover letter) have to be sent to the inspection participants. The inspectors have to prepare the inspection by reading the document beforehand; the authors prepare themselves and the moderator prepares the agenda.

The performance of an inspection meeting follows this agenda. Each inspection participant has a well-defined role: inspector, moderator, author, recorder, consultant, or guest. The inspectors speak about their findings and decide if the finding is correct. If so, the finding is recorded in the minutes. The moderator ensures that the atmosphere is constructive, that the schedule is followed, and that nobody digresses from the subject. Discussions are allowed only on a small scale. Further discussions are beyond the scope of an inspection meeting, because solutions should be developed by the project team and not by the inspection team. Solutions are only relevant to an inspection meeting if they can be quickly devised. The consultant is responsible for the adherence to the prescribed inspection process and also acts as a co-moderator.

After the inspection meeting, the recorder writes the minutes. These minutes are the basis for the correction of the document by the author. Once, the author adapted the document, it is sent to the inspection participants who may give their feedback on the corrections.

The data collection and subsequent analysis is not seen as an add-on to the inspection process but as an integral part of the entire approach. Based on the developed GQM plan, the required data are collected during all development activities in general and for inspection activities in particular. Examples of such collected data are various effort values (e.g., effort spent for the inspection meeting or reading the document) or classifications of defects (e.g., the severity or impact of a defect).

As soon as data analysis results are available, feedback sessions involving the project members are performed. The results of these sessions are used to change and optimize the inspection process or to package experiences to be stored in Allianz’ experience base for future reuse.

The technology

An essential parameter in the inspection process is the reading technique. PBR was adapted for Allianz. Usually, a separate scenario is meant to be developed for each perspective. However, when developing the single scenarios for the Euro conversion project, we realized that the quality aspects for one perspective might also be interesting for the inspectors of other perspectives. Therefore, we adapted the development of scenarios in such a way that we combined the scenarios for all perspectives into a single scenario. For each document to be inspected, we have thus one single scenario with questions and activities. For each of these questions and activities, there are annotations that denote to which perspective they originally belong to.

4.3 Initial Results: Initial Evaluations and Assessments

The HYPER project is currently in progress. Therefore, we can only present here initial results from the first four inspections for which measurement was taken. Due to the small number of inspections, only qualitative results can be provided that are to be further investigated through additional interviews with the project members and further data collection on subsequent inspections. Such a qualitative analysis should be considered as a way to help focus our future investigations and identify hypotheses to be verified in the remainder of the project.

In the first subsection, we investigate the kind of findings detected during these inspections. Second, we compare the inspections with respect to their effectiveness and efficiency. Third, we attempt an early evaluation of the costs and benefits of inspections. Finally, we investigate the application of defect content models.

4.3.1 Data Source

The results are based on four inspections performed for the EURO Conversion project. The inspected documents were user-interface descriptions capturing the system requirements to be implemented. They consisted of screens' descriptions, that will help answer client calls in the service center, and settlement letters of accounts, which will be sent to the client.

Since the inspected documents come from the same project and are of the same type, we can assume that they have a similar defect density and the number of pages can be assumed to be a simple but reasonable size measure. These assumptions have been deemed reasonable by the inspection participants and help simplify the data collection analysis.

4.3.2 Classification of findings

As alluded previously, inspections aim at collecting findings. A finding can be characterized by the following pair: Finding = (Class, Reference). *Class* denotes whether an *issue* was raised, a *question* was raised, an *improvement proposal* was made, or a *comment* has been written down.

Questions are used to identify problems that need to be further investigated or discussed outside the inspection meeting. Improvement proposals aim at collecting suggestions for future versions of the product or for future coordination strategies between different departments. Comments record additional, relevant information related to the project.

Reference may denote the object related to a finding (the inspected product, another product in the project, the development process). It may also refer to another project when additional or new requirements to interfacing systems are identified. It refers to the development process when it is decided to change the project's process. When findings concern other documents that were developed or have to be developed, they are classified as 'other product in the project'. Issues (a specific class of findings) that concern the inspected product are usually referred to as *defects* in the literature but it was decided to use Allianz' terminology in this paper.

Each finding is characterized by three attributes: *Severity* denotes the importance of the finding. There are three levels: *very critical* (A), *critical* (B), and *interesting* (C). *Cause* denotes the kind of error was committed that led to the finding. *Impact* denotes the impact the finding would have had if it had not been detected.

Severity

Figure 7 shows the severity of the detected findings. A proportion of 78,3% of all findings had either a *very critical* (A) or *critical* (B) severity and only 21,7% were of severity *interesting* (C). The large proportion of *very critical* and *critical* findings suggests that inspections are useful, as they lead to essential improvements in the product. Yet, their net benefit still needs to be investigated and compared to their cost.

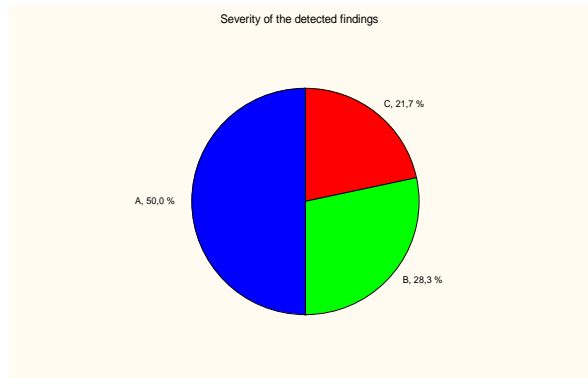


Figure 7 Severity Distribution

Impact

Looking at Figure 8 reveals that mostly findings of type *User-friendliness* and *Function* are detected. The high prevalence of user-friendliness findings can be attributed to the type of the inspected documents. User-friendliness and understandability are the main quality aspects of user-interface descriptions.

In light of Allianz’ business goals, i.e., “better customer satisfaction”, these inspections were thus successful as most findings would have had a direct consequence on user’s satisfaction.

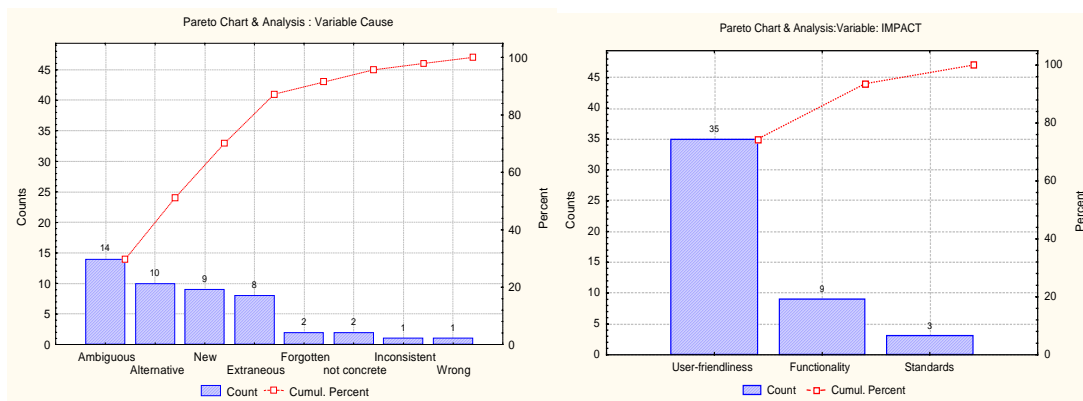


Figure 8 Finding Distributions

Cause

The most prevalent finding causes were *ambiguity*, *alternative*, *new*, and *extraneous*. The high proportions of categories *alternative* and *new* suggest that inspections are an effective means to elicit new ideas through discussions. This might also explain in part the high proportion of effort invested in meetings, as discussed below in Section 4.3.3.

A high proportion of findings of type *ambiguity* and *extraneous* help generate more understandable letters for customers and provide better information systems to support customer service. This statement was confirmed when interviewing the inspections’ participants.

4.3.3 Comparing Effectiveness and Efficiency

In order to analyze the effectiveness of inspections (i.e., their capability of detecting findings) as well as their efficiency (i.e., their cost-effectiveness), these values were computed and displayed in Figure 9. The effectiveness is defined as the number of findings of severity A and B per unit of size and the efficiency is defined as the number of findings of severity A and B per unit of effort.

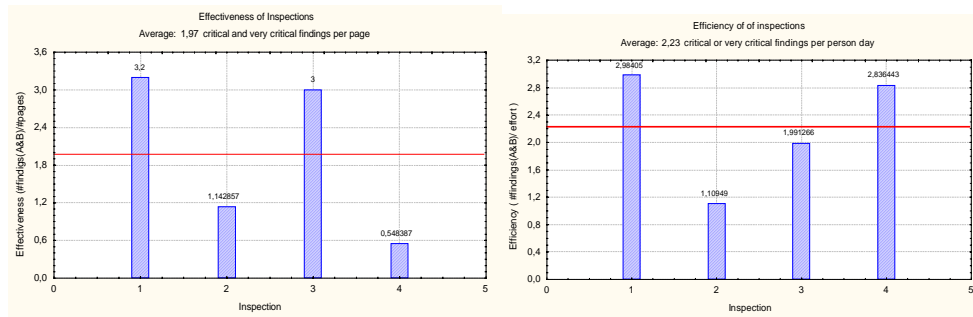


Figure 9 Effectiveness and efficiency for all inspections

Inspection 2 appears to have significantly lower effectiveness and efficiency. The inspection team consisted of inspectors who had less experience in the application domain. Therefore, the moderator had to provide the inspectors with additional explanations and background information. This resulted in a higher meeting effort and, therefore, a lower efficiency value.

On the other hand, the inspection team of inspection 4 was nearly identical to the inspection team of inspection 2. Furthermore, the same kind of document was inspected. Therefore, the inspection team could benefit from the experience gained during inspection 2. Most likely, this resulted into a smaller inspection effort and, therefore, a higher efficiency value.

Similarly, inspection 4 shows a low effectiveness value. This inspection was special in the sense that the inspected document was significantly larger than the other documents, but the effort invested was not much different than for the other inspections. The effort per unit of document size is thereafter referred to as *effort density*.

In order to investigate further the effect of effort density, we computed it for each defect detection steps (preparation by reading, meeting). This effort density denotes the effort that is spent per unit of size. The result is shown in Table 1.

For inspections 2 and 4, which showed the smallest effectiveness values, these inspections also showed less preparation effort per unit of size. Thus, it might be a plausible interpretation is that effort density has an impact on inspection effectiveness. In addition, we looked at the relation between preparation effort density and effectiveness (see Figure 10). This scatterplot seems to confirm the impact, as there is a visible correlation between preparation effort density and effectiveness.

However, more data points are required to make final statements and provide sound statistics. A sufficient number of data points can then be used to determine a functional relationship between effort density and effectiveness. Such a relationship can then be practically applied to plan the preparation effort that is required to achieve a certain level of quality, that is, effectiveness. [Briand et. al, 1997c]

For example, the simple regression line in Figure 10 indicates that, for each additional hour of preparation, roughly 1.5 critical and very critical, additional findings will be found in a page on average. Yet, it must be taken into account that this line is based only on a few inspections. When more

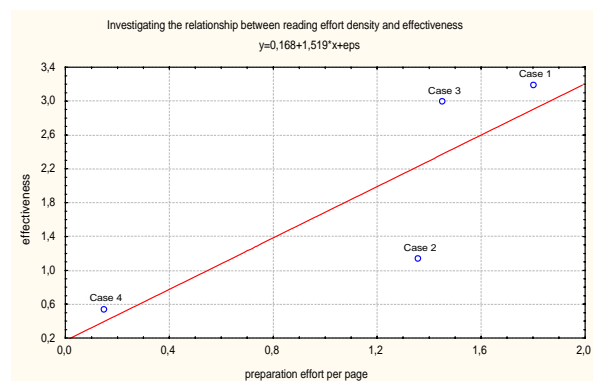


Figure 10 Effectiveness impacted by effort density

data will be available, it would also be interesting to see whether the relationship between effectiveness and effort density starts to plateau above a certain level of effort density. Such a plateau would indicate a threshold above which additional effort does not pay off in terms of effectiveness

	#findings (A&B) per page	Preparation effort [person-hours]	Preparation effort per page	Meeting effort [person-hours]	Meeting effort per page
1	3.2	9.00	1.80	28.00	5.60
2	1.14	9.50	1.36	40.30	5.76
3	3	2.90	1.45	15.00	7.50
4	0.55	4.55	.15	30.00	.97

Table1: Effort and effort densities

The scatterplot shows also that effort density is not the only factor having an impact on effectiveness. The effectiveness value for inspection 2 is significantly lower than the value that is predicted by the regression line between effectiveness and effort density (Figure 10). A plausible explanation is that the inspection team consisted of inspectors with less experience in the application domain. Therefore, the impact of the training and experience of inspectors on effectiveness should be carefully investigated in the future.

In conclusion, we can say that a lower preparation effort per size (i.e., preparation effort density) is a plausible strong driver of effectiveness. Yet, more data have to be collected to make final statements about a relationship. It is important to note, however, that effort density does not explain all the effectiveness variation. As discussed earlier, inspection 2 has a lower effectiveness than expected which could be attributed to the lack of application domain experience of the inspectors.

4.3.4 Cost-Benefit Analysis

In order to assess whether the introduction of inspections was successful, the costs of inspections have to be compared with their benefits. The costs of inspections are determined by the effort spent on inspections (e.g., training, creating scenarios, planning, preparation, meeting, etc.) The benefits are twofold. First, effort is saved due to the early detection of defects, as otherwise the defects have to be removed later on, thus incurring higher costs. Second, an inspection can have additional, indirect benefits such as a better communication between the participants.

As a first attempt to evaluate the costs and benefits of inspections, the project leader assesses subjectively the benefits of the inspection after the inspection meeting has taken place. First, the effort saved in later phases due to the early detection of defects is subjectively estimated (referred to as effort savings or “primary benefit”) by inspection participants. In Figure 11, for each inspection, the total inspection effort spent is compared with the effort savings. It can be observed that, for all inspections, the effort savings are larger than the invested effort. Therefore, from this point of view, the investment in inspections seems to be beneficial. This result should, however, be interpreted carefully since the uncertainty regarding the estimated effort savings is currently unknown and should be the object of further investigation. A more accurate way to quantitatively assess the cost-effectiveness of inspections is the efficiency model presented in Section 3.2.2.3. During the course of the HYPER project, the data required for this model will be collected and used to estimate cost savings.

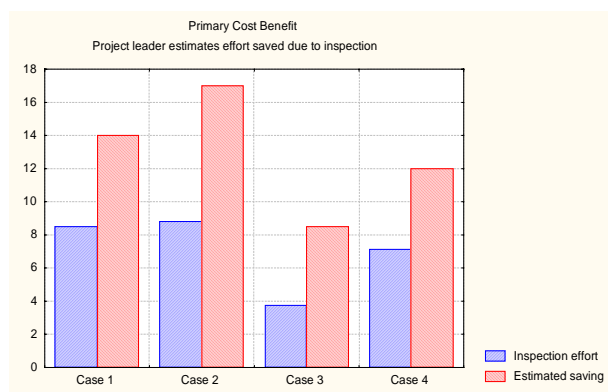


Figure 11 Primary benefits

Yet, inspections do not only provide benefits in terms of saved correction effort. There are also indirect benefits. After each inspection, the project leader assesses these benefits on an ordinal scale: (none, low, medium, high). Figure 12 shows these benefits. If we look at the median scores for each indirect benefit considered, we can see that *user satisfaction*, *call-center relief*, and *follow-up projects* show the highest scores.

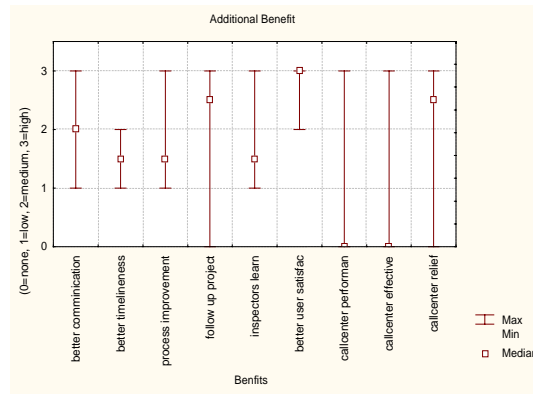


Figure 12 Additional benefits

The high benefit of *user satisfaction* indicates that the involvement of the future (internal) clients in early phases and through inspection led to the definition of a more appropriate system. This contributes to Allianz’ business objective (“Better customer satisfaction of delivered products”) that motivated the introduction of inspections in the first place.

The benefit *follow-up projects* can be explained as follows: Next year, there will be a new project dealing with the topic of creating more customer-friendly letters. The inspected documents contained several standard letters. Thus, the findings of the inspections will be valuable for the follow-up project.

The benefit of *call-center relief* is explained as follows: As a result of the inspections, Allianz’ clients receive letters that are easier to read and understand. This results in fewer phone calls to the call-center, whose workload is therefore reduced.

The high maximum values of all benefits indicate that, in some circumstances that remain to be determined, most of the indirect benefits mentioned above can be obtained through inspections. Since some significant scoring variation can be observed for most indirect benefits, further investigation is required to identify the factors than can help obtain such benefits.

Effort Distribution

Figure 13 shows the effort distribution for all inspection steps. As it can be seen, *meeting* and *preparation* show by far the highest proportions. The proportions across inspection steps are surprisingly similar across inspections, except for meeting preparation, which shows slightly more variation.

The large amount of meeting effort reflects the fact that the inspection meeting is not only used to collect findings found during preparation. It also serves as an additional defect detection step and as a tool for coordination and communication among the various project stakeholders. The latter point could be a point of further investigation since shorter meetings might help reduce a substantial inspection time span and thus facilitate their implementation under tight schedule pressure.

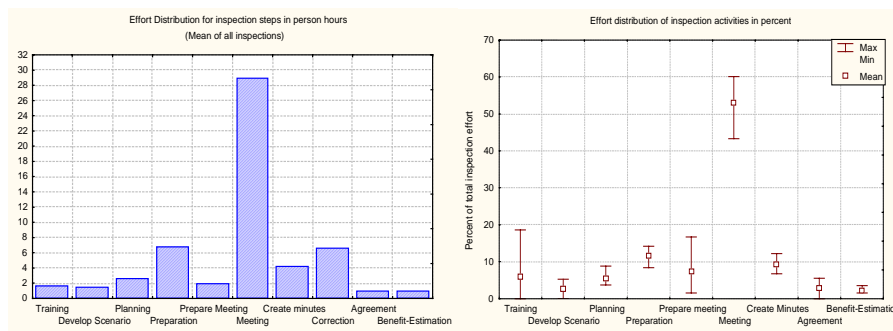


Figure 13 Effort Distribution for inspections

4.3.5 Defect Content Estimation

First, an appropriate defect content estimation method had to be selected to satisfy the constraints under which Allianz is performing inspections. Capture-Recapture Models have the most stringent requirements on the inspection process. They require the inspectors to note all the defects they detect and agree on common defects during preparation and meeting, respectively. Defects detected during

meetings have to be distinguished from those found during preparation. So far, such detailed data is not collected in the inspections under study.

In addition, Capture-Recapture Models are inaccurate for inspections where a large proportion of defects is detected by only one inspector [Briand et. al. 1998]. This is the case for the data at hand and can be explained by the fact that user-interface descriptions do not mainly contain defects but also generates many new ideas and alternative solutions. Therefore, the Extended Detection Profile Method (EDPM), as presented above, seems to be the most reasonable approach within the Allianz environment.

During this early stage of the HYPER project, where no testing data are available, only initial, qualitative evaluations regarding the accuracy of the defect estimates can be made. In the remaining course of the project, we are going to determine the total number of defects in the inspected documents by tracing back defects to the document where they were introduced. Thus, we will be able to compare the actual and estimated numbers of defects in the inspected documents.

For the initial evaluation, we compare the detected numbers of findings (severity A and B) in the inspected product, which is Allianz' equivalent of a defect, with the corresponding EDPM estimates.

	#findings (AB) detected: D	EDPM	Remaining findings: EDPM-D	Estimated proportion of remaining findings (EDPM-D)/EDPM
Insp1-AB	16	25	9	36%
Insp2-AB	7	11	4	36%
Insp3-AB	6	6	0	0%
Insp4-AB	17	32	15	47%

Table2: Defect Content Estimates

Interestingly, inspection 3 shows a predicted number of remaining findings equal to 0. Since we can expect that the document has reached a relatively stable and mature state after a second inspection, this suggests that the EDPM estimate is reasonable. In addition, Inspection 4 which has the largest document and the lowest effort density also shows the largest number of remaining defects, which is also consistent with intuition. Overall, these are qualitative indications that the EDPM estimates are plausible ones.

Yet, these findings have to be confirmed by analyzing defects detected during testing. For example, based upon this estimate we would expect the document in inspection 4 to be more defect-prone than the other documents during the testing phase.

4.4 Lessons Learned about the people, the process, and the technology

There are several important lessons that we have learnt. In the following, we will group these lessons into three categories: people, process, and technology.

People

First, it is necessary to motivate and inform all people concerned about the new technologies to introduce. This is true not only for the developers but also for the management. Thus, the commitment of the management is an essential prerequisite for technology transfer. If managers are not convinced of the benefits of inspections, they might be tempted to assign to inspections those employees who are readily available and not those who are most qualified. Therefore, managers have to understand the entire scope and general conditions under which inspections and data collection take place.

For the motivation and training of the future inspection participants, the training courses provided by FhG IESE showed to be of valuable help. These training courses were developed taking into account the specific context of Allianz. With these training courses, our developers were motivated and capable of performing Perspective-based inspections and motivated for goal-oriented measurement.

Another crucial point to the transfer of inspections was the creation of an open and constructive atmosphere during inspection meetings. It is the main task of the moderator to create and maintain this atmosphere. For example, s/he has to be able to cope with possibly antagonistic participants. Special moderator training is therefore helpful for the people who are going to this responsibility. However, the inspection participants should also be reminded that they play an important role in ensuring a constructive atmosphere in inspection meetings.

A positive side effect of the inspection meeting is also that all relevant stakeholders get the opportunity to discuss their findings. This speeds up the quality assurance process since the decisions made during the inspection meeting, with the consent of all inspectors, would have taken weeks in other circumstances.

Also, the Allianz' term for defect, i.e., 'finding', proved to be of psychological importance. In particular, it conveys a positive meaning, which ease the introduction of inspections. An additional psychological benefit of inspections is that the inspectors identify themselves with the inspection's results since these were obtained through joint discussions.

A final important aspect of Perspective-based inspection was the early involvement of Allianz employees who later on deal with the inspected documents. Since these employees often have direct access to external customers, they can bring in their experience how the inspected product should look like in order to achieve high customer satisfaction. This was perceived very beneficial.

The technology

Perspective-based inspections were found to be easy to use and practical. The approach was very suitable for the Allianz' environment, where various stakeholders are interested in the project deliverables to be inspected. Additionally, the scenario for the perspectives was very helpful, especially for inspectors with little experience in the domain and perspective-based inspections.

The process

In order to support the inspection process, a template for the inspection minutes was a simple and effective tool. The template was on one hand useful for tracking the inspection process. For example, it contains information on who will inspect the document, it guides the task of the recorder, it helps determine and schedule the steps after the inspection meeting, and it serves as an input to the corrections performed by authors. On the other hand, the template also guides the data collection during the inspection. For example, it contains information regarding finding classification schemes. Overall, the inspection participants judged this template to be 'very helpful'.

A template for data analysis was also developed and assisted the coaching team to validate the collected data with respect to their completeness. Additionally, computer-based tools, such as common spreadsheet applications, are extremely valuable to perform and visualize analysis results.

A very crucial aspect during the measurement program is to continuously support and motivate the project members to collect data, since measurement requires a constant, additional effort. In our case, the project team members have to specify their project effort and classify defects. Usually, these data collection tasks are regarded as additional burden. Thus, especially in the initial phase of a measurement program, a coaching team has to constantly demonstrate the benefits of measurement to project members.

Since the development team has to focus on its development tasks, the coaching team analyzes the inspection data. Data collection, validation, and analysis as well as the final experience packaging have to be performed by a separate unit, in this case the coaching team.

4.5 Outlook for project HYPHER: further actions

The results and lessons learned presented so far were made on an initial, small set of inspections within the EURO conversion project. Further inspections will be performed and the measurement program is continued in order to investigate the benefits of perspective-based inspections and effective ways to plan, control, and improve them, from both the standpoints of effort and quality.

The next steps of the project will first include the analysis of the collected data according to the defined GQM measurement plan. We will then further assess the direct and indirect benefits of Perspective-Based Reading (PBR) inspections. In particular, we will focus on reduction in defect slippage to testing and reduced test and rework effort. The data will also be used to improve the benefits of PBR inspections by better planning and controlling them and through a better understanding of the efficiency factors that drives them, e.g., inspectors' experience.

The analysis results will be then presented to inspectors and project participants in order to interpret them in context, get feedback, and identify relevant directions for further investigation. Following the GQM approach, this will be performed through structured, feedback sessions [Briand et. al. 1997a].

The experience and lessons learned gathered throughout the project will be gathered and stored in the Allianz Life Assurance experience base. This involves the development of an operational experience package including a user-oriented manual providing guidelines and heuristics for the application of both goal-oriented measurement and PBR inspections. The overall project results will also be reported and will be used to support further dissemination and of the newly introduced technologies.

The results will be published both internally within Allianz (e.g., at established meetings and workshops for best practice experience exchange) and externally (e.g., conferences).

5 Summary and Conclusions

The EURO conversion implies tremendous changes to existing software systems across Allianz divisions. Due to this strategic importance of the EURO conversion for Allianz Life Assurance, high quality of the final software system is a necessity. For this reason, Allianz Life Assurance investigates innovative quality assurance technologies in the framework of the ESSI Process Improvement Experiment (PIE) "HYPER" (Project no. 27839). This ESSI PIE is part of a wider-scope software quality improvement program performed at Allianz Life Assurance. The innovative quality assurance technologies include Perspective-based inspections and quantitative models to control the defect content of the inspected artifacts as well as to assess the Perspective-based inspection approach.

Software Inspection is an industry-proven best practice for software quality assurance. Yet, inspections have shown a wide variation in terms of benefits across the industry [Briand et. al. 1998a]. In order to exploit their full potential, we believe that software inspections require:

- Systematic reading techniques that tell inspection participants what to look for and, more importantly, how to scrutinize a software artifact for defects.
- Ways to control the resulting quality of an artifact after inspection, e.g., remaining number of defects, and decide about further inspection.
- Ways to plan inspections to ensure enough effort is assigned to meet quality requirements.
- A good understanding of the driving factors that may optimize their benefits.

Perspective-based inspections provide effective, systematic reading techniques that leverage existing Fagan-like inspection approaches. PBR inspections define specific inspection viewpoints that focus upon specific sets of quality properties. In addition, PBR provides guidance for inspection participants on how to scrutinize a software artifact in a systematic manner. Capture-recapture models and other related graphical approaches help assess defect content after inspections. Based on the data collected through our GQM measurement plan, models can also be built to understand the relationships between effort and other factors (e.g., inspector experience) with inspection effectiveness.

In this paper, we presented initial empirical results regarding the application of PBR inspections on requirements documents. In particular, we investigated the effectiveness and efficiency of requirement inspections. Although the results are very preliminary and qualitative in nature, a plausible driver of effectiveness we identified was the effort spent on gaining findings and the inspector experience with respect to the application domain. An analysis of the inspection findings also shows the tangible benefits of PBR inspections, both in terms of effort saved and more indirect benefits. For the documents inspected so far, these indirect benefits mainly translate into an improvement of the user-friendliness of the end software product and, thus, a higher user satisfaction. This is plausible when considering that, in this study, user-interface descriptions were inspected.

Finally, a convenient way to estimate the remaining finding potential after inspection was identified in the Allianz Life Assurance context. Although the evaluation is very preliminary, the Extended Detection Profile Method (EDPM) estimates seemed plausible. Further validation of the constructed model will be undertaken once testing defect data will be collected.

6 Acknowledgment

We would like to thank the project and inspection members of HYPER for their cooperation and willingness to provide data.

7 References

- Ackermann et. al. 1989 Ackerman, A. F., Buchwald, L. S., and Lewsky, F. H. (1989). Software Inspections: An Effective Verification Process. *IEEE Software*, 6(3):31-36.
- Basili et. al. 1996 Basili, V., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S., and Zelkowitz, M. (1996). The Empirical Investigation of Perspective-based Reading. *Journal of Empirical Software Engineering*, 2(1):133-164.
- Basili 1997 Basili, V. R. (1997). Evolving and Packaging Reading Technologies. *Journal of Systems and Software*, 38(1).
- Boehm 1981 Boehm, B. W. (1981). *Software Engineering Economics*. Advances in Computing Science and Technology. Prentice Hall.
- Briand et. al. 1997a Lionel C. Briand, Christiane Differding, and H. Dieter Rombach, Practical Guidelines for Measurement-Based Process Improvement, *Software Process Improvement and Practice Journal*, vol. 2, no. 3, 1997

- Briand et. al. 1997b Lionel Briand, Khaled El Emam, Bernd Freimut, and Oliver Laitenberger, Quantitative Evaluation of Capture Recapture Models to Control Software Inspections, in Proceedings of the 8th International Symposium on Software Reliability Engineering, pp. 234-244, 1997. Also available as International Software Engineering Network Technical Report ISERN-97-22¹
- Briand et. al. 1998a L. Briand, K. El-Emam, T. Fußbroich, and O. Laitenberger. (1998). Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects. In Proceedings of the 20th International Conference on Software Engineering, pages 340-349. IEEE Computer Society Press, 1998.
- Briand et. al. 1998b Lionel C. Briand, Khaled El Emam, and Bernd G. Freimut, A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method, , in Proceedings of the 9th International Symposium on Software Reliability Engineering. Also available as International Software Engineering Network Technical Report ISERN-98-11, Fraunhofer Institute for Experimental Software Engineering, 1998.
- Cai, 1998 Kai-Yen Kai. (1998). On Estimating the Number of Defects Remaining in Software, Journal of Systems and Software, vol. 40, pp. 93-114.
- Cheng and Jeffrey 1996 Cheng, B. and Jeffrey, R. (1996). Comparing Inspection Strategies for Software Requirements Specifications. In Proceedings of the 1996 Australian Software Engineering Conference, pages 203-211.
- Fagan 1976 Fagan, M. E. (1976). Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, 15(3): 182-211.
- Gilb 1993 Gilb, T. and Graham, D. (1993). Software Inspection. Addison-Wesley Publishing Company.
- Günther et. al. 1996 Holger Günther, H. Dieter Rombach, and Günther Ruhe, Kontinuierliche Qualitätsverbesserung in der Software-Entwicklung, Wirtschaftsinformatik, vol. 38, pp. 160--171, Apr. 1996.
- Jones 1985 Jones, C. L.. (1985). A Process-Integrated Approach to Defect Prevention, IBM Systems Journal, vol. 24, No. 2. pp. 150-167.
- Leippert and Ruhe 1998 Friedrich Leippert and Günther Ruhe, "Software Best Practice Activities -- An Interesting Experience at Allianz Life (Germany)." ESSI g-r-a-m: Newsletter of the Software Best Practice Community, Feb. 1998.
- Wohlin and Runeson, 1998 Claes Wohlin and Per Runeson. (1998). Defect Content Estimation from Review Data. Proceedings of the 20th International Conference on Software Engineering. pp. 400-409
- Parnas 1987 Parnas, D. L. (1987). Active Design Reviews: Principles and Practice. Journal of Systems and Software, 7:259-265.
- Porter et.al. 1995 Porter, A. A., Votta, L. G., and Basili, V. R. (1995b). Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. IEEE Transactions on Software Engineering, 21(6): 563-575.
- Vose, 1996 Vose, D. (1996). Quantitative Risk Analysis: A Guide to Monte Carlo Simulation Modelling. John Wiley Sons.

¹ All ISERN reports can be accessed via http://www.iese.fhg.de/ISERN/pub/isern_biblio_tech.html

An Experience in Automatic Verification of Railway Interlocking Systems

Jakob Lyng Petersen

ScanRail Consult

jlp@rdg.bane.dk

Banestyrelsen rådgivning

ScanRail Consult

Introduction

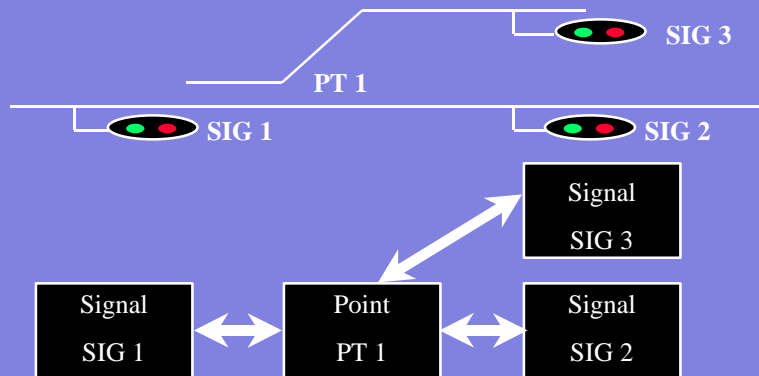
- Motivation of ScanRail consult:
 - Interlocking systems using new technology
 - Long verification process
- Automatic black-box verification
(medium sized station
w/ 22 points, 23 signals)

Banestyrelsen rådgivning

ScanRail Consult

STERNOL programs

Composing instantiated generic objects



Banestyrelsen rådgivning

ScanRail Consult

STERNOL evaluation

- Guarded variable assignments:

```
Instr := {SwitchLeft if  
          ConditionLeft  
          SwitchRight if  
          ConditionRight  
          ...}
```
- Iterate until fixed point is found

Banestyrelsen rådgivning

ScanRail Consult

STERNOL semantics

- Semantics: Set of fixed point states;
A conjunction of formulas of the form:

$$var = expr \Leftarrow bexpr$$

$$\begin{aligned} Program &\equiv \\ var_1 = expr_1 \Leftarrow bexpr_1 \wedge \dots \\ var_n = expr_n \Leftarrow bexpr_n \end{aligned}$$

Program-formula is true in all possible fixed point states

Banestyrelsen rådgivning

ScanRail Consult

The Stålmarch method

If a propositional formula

$$\varphi \equiv \neg (Program \Rightarrow Req)$$

is self-contradictory then there exists an n for which this is discovered by n -saturating φ

The *Stålmarch algorithm* computes

$$n\text{-sat } \varphi \text{ in } O(|\varphi|^{2n+1})$$

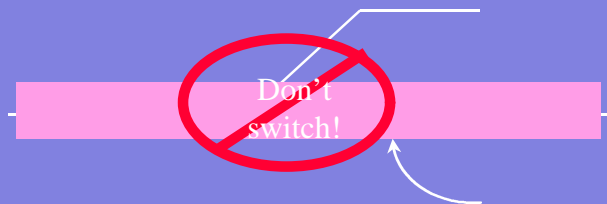
Emperically: often $n = 0$ or $n = 1$ will suffice!!

Banestyrelsen rådgivning

ScanRail Consult

A Local Requirement

LockAllocReq: A point must not switch position if it is part of an allocated train route



Banestyrelsen rådgivning

ScanRail Consult

Proving local requirements

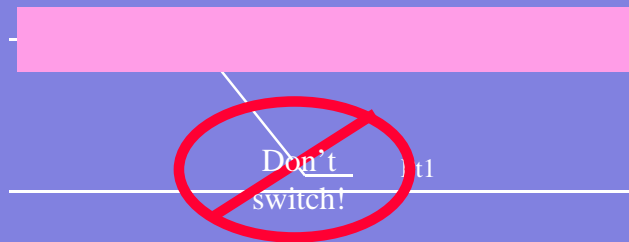
- We need only consider a *generic* Point-object
- Size: 5131 subformulas, 276 integer variables
- Easily proved in 4 sec.

Banestyrelsen rådgivning

ScanRail Consult

A non-local requirement

LockFlankReq: Pt1 must not switch position if implementing flank protection for an allocated train route



Banestyrelsen rådgivning

ScanRail Consult

Problems in proving non-local requirements

- A non-generic requirement
- Size of entire program:
 - 388000 subformulas, 17000 int. vars.
 - Potential state space $> 10^{100000}$
- 0-sat: 20 sec, 1-sat: 20 hrs. (no result)
2-sat? Forget it!
Counter model: not found within reasonable

Banestyrelsen rådgivning

ScanRail Consult

“Projecting” programs

Throw away formulas which don't influence
“interesting” variables

Projecting onto {z}:

Projecting onto {z}:

$$y = 3 \leftarrow w = 3 \wedge u < 2$$

$$x = 0 \leftarrow a \neq 6 \wedge b = 2$$

$$x = 2 \leftarrow a = 6 \wedge b \neq 2$$

$$z = 7 \leftarrow a = 5 \wedge d = 8$$

$$z = 8 \leftarrow b = 2 \vee a \neq 5$$

Projection properties

- If projected programs fulfill req., so does non-projected program
- Counter models for project are not trustworthy
- Minor reductions using **project**: 10%
- Using project typically between 50% and 90%

Is LockFlankReq fulfilled?

- Applying project:
 - 2-sat possible (< 3 hrs), but no result!
After 2-sat: counter model easily found
Counter model can be extended
might be... is not fulfilled.

Banestyrelsen rådgivning

ScanRail Consult

Conclusions

- Local requirements tend to be easily proved
Non-local requirements are difficult to handle; apparently *not* implemented
- Defensive programming might help
Programs should be designed with the

Banestyrelsen rådgivning

ScanRail Consult

An Experience in Automatic Verification of Railway Interlocking Systems

Jakob Lyng Petersen
ScanRail Consult
e-mail: jlp@rdg.bane.dk

Abstract

This paper presents experiences in applying formal verification to a large industrial piece of software. The area of application is railway interlocking systems which has earlier been addressed in for instance [10], [9], [8], and [7]. We try to prove requirements of the program controlling the Swedish railway station Alingsås by using the decision procedure which is based on the patented Stålmärck algorithm. While some requirements are easily proved, others are virtually impossible to manage due to a very large potential state space, which is in excess of 10^{100000} . We present what has been done in order to get, at least, an idea of whether or not such difficult requirements are fulfilled or not, and we express thoughts on what is needed in order to be able to successfully verify large real-life systems.

1 Introduction

In recent years the use of computer-controlled industrial systems has increased. When these are safety critical, the need to verify the safety requirements of such a system is evident. As the size and complexity grows, the need for well-designed programs based on suitable abstractions as well as for computerized verification tools becomes more urgent. European Norms such as [20, 21] are recommending use of formal methods (and hence, computerized verification tools). Obviously, many industrial manufacturers and customers desire that as much as possible of the verification is done automatically, quickly and, hence, with low financial costs.

In this paper, which is a survey of some of the results of the PhD thesis [16], we present experiences with a case study: automatic verification of requirements of an in-use interlocking system which controls the Swedish railway station Alingsås. Alingsås is a medium-sized railway station containing 22 points and 33 signals.

Our aim is to avoid the need for detailed knowledge about the actual program implementation. Since the program requirements are derived from requirements to the physical devices controlled and observed by the program (i.e., they are expressed solely in terms of input

and output variables) we only have to know how the program interacts with its surroundings. An automatic theorem prover, on the other hand, should be able to sort out the complicated structure of the actual implementation.

The verification is based on applying the decision procedure *NP-Tools*, developed by the Swedish company Logikkonsult NP AB. It is based on the patented Stålmärck algorithm [17, 18, 19].

Others have tried to verify properties of railway interlocking systems, e.g. [13, 6, 2]. Groote, Koorn and van Vlijmen have managed to verify several requirements for a small Dutch railway station [8], and their results indicated that applying the Stålmärck method might be more efficient than applying other techniques, including ROBDDs [3].

The satisfiability checker SATO [24] has proved very efficient on some problems, but, unlike NP-Tools, does not incorporate arithmetic.

2 Railway terminology

A railway station (see Figure 1) consists of a number of tracks divided into *track segments*.

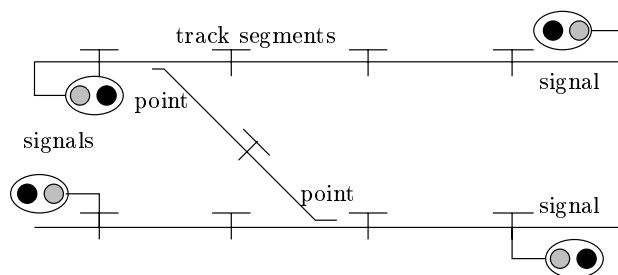


Figure 1: A simple station with two points, four signals and ten track segments. Track segments are separated by the Ts.

A *point* is used to control how a train can pass through a branching track segment. A point is in “left-hand position” if it allows a train to drive through the left branch, in “right-hand position” if it allows a train

to drive through the right branch. A point is said to be *locked* if the control program prevents the point from changing position.

A *train route* is a list of track segments which can be allocated or deallocated. If a train route is allocated, trains are allowed to drive through its track segments, otherwise not.

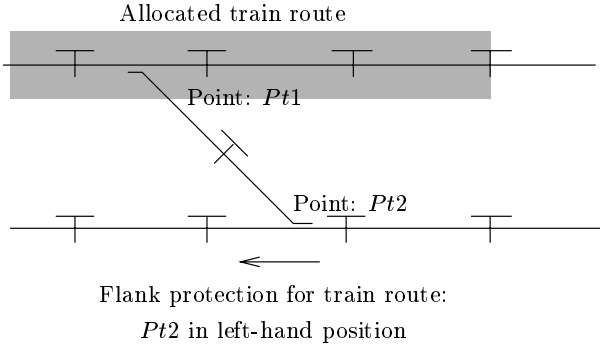


Figure 2: When $Pt1$ is part of an allocated train route, $Pt2$ must be in left-hand position in order to implement flank protection, hence, preventing trains from driving towards the allocated train route from the lower track segments.

A point is implementing *flank protection* for an allocated train route, if the point (which is not itself part of the train route) is positioned such that a train/wagon driving through the point is not directed into a track segment of the train route, and if this would happen in case the point was positioned differently. Figure 2 visualizes the concept.

3 The interlocking program

The program controlling the station is written in the language STERNOL [1, 14], which has been developed by ADTranz (formerly ABB Signal) in Sweden. STERNOL is used to construct interlocking programs for railway applications.

The source code of a STERNOL program consists of implementations of a few *generic* objects. To form a program for an actual railway station, the generic objects are instantiated and connected according to a site-data file. That two objects, O_1 and O_2 , are connected means that O_1 is able to read some variables which are being assigned by O_2 and vice versa. Each instantiated object forms a set of base formulas. The program is the union of all the objects. A number of variables are associated each object. The values of the variables indicate the state of an object.

The generic objects are typically designed to control each kind of physical device of the station such as a point, a signal or a level crossing. The object connections reflect the topology of the physical objects of the station.

The program works by repeatedly performing an *evaluation cycle*, i.e., the values of output and local variables are updated according to the values of the input variables when the cycle begins.

The program source code can be translated directly into a set of logical formulas, *base formulas*, which are all true for any variable assignment which is possible when an evaluation cycle terminates. In this paper we are only concerned with the resulting set of base formulas. A base formula has the form:

$$v = e \Leftarrow be$$

where v is a variable, e is an integer expression and be is a boolean expression.

An example of base formulas controlling an output variable, **Instruction**, which gives an instruction to a point is (simplified):

$$\begin{aligned} \text{Instruction} = \text{SwitchLeft} &\Leftarrow \text{Command} = \text{TryLeft} \wedge \\ &\text{Condition_OK}_L \\ \text{Instruction} = \text{SwitchRight} &\Leftarrow \text{Command} = \text{TryRight} \wedge \\ &\text{Condition_OK}_R \end{aligned}$$

where **Instruction** and **Command** are program variables, **SwitchLeft** and **SwitchRight** are constants, and *Condition-OK_L* and *Condition-OK_R* are propositions (expressing statements about other program variables) which are true when the point is allowed to switch into left-hand position and right-hand position (respectively) and false otherwise.

A value assignment making all the base formulas true is a possible evaluation-cycle result for the program only if no variable is assigned a value which is not in its (declared) value domain and if for each variable, there is exactly one base formula of that variable such that the right-hand side of the \Leftarrow is true. In the following we shall only take such *well-formed* variable assignments into consideration.

4 Formalizing requirements

The safety requirements for railway stations are basically that trains must not collide and must not derail. However, these requirements are implemented by a number of requirements for the physical devices of a station and by a set of assumptions about the accepted behaviors of trains (based on the regulations of the railway authority in question). A number of requirements

exists for the Swedish railway stations [7]. We shall examine two of these:

LockAllocReq: A point should be locked if it is part of an allocated train route

LockFlankReq: A point should be locked if it implements flank protection for an allocated train route

We note that both requirements are generic in the sense that they do not mention any *specific* points or train routes. By studying the program documentation, it is possible to find the corresponding program requirements. LockAllocReq looks the same for all the point objects of the station:

$$\begin{aligned} & Point.Status \neq DeallocFree \Rightarrow \\ & Point.Instruction \neq SwitchLeft \wedge \\ & Point.Instruction \neq SwitchRight \end{aligned}$$

where *Point* refers to some point object, *Status* and *Instruction* are variables of that object and *DeallocFree* is a symbolic constant indicating that the point is not part of an allocated train route and is not occupied by a train, and *SwitchLeft* and *SwitchRight* are symbolic constants standing for point instructions “switch to left-hand position” and “switch to right-hand position”, respectively.

The other requirement will look differently depending on what objects are affected by the given train route and which flank protecting point we are examining. We chose one instance of the requirement. This instance is reduced by simplifications which can be justified by making observations about the concrete topology of the station. See [15] for a more detailed discussion of this. The situation is similar to the one depicted in Figure 2. We end up with the requirement

$$\left(\begin{array}{l} Pt1.Position = Left \quad \wedge \\ Pt1.Status = Alloc \quad \wedge \\ Pt2.Position = Left \end{array} \right) \Rightarrow Pt2.Instruction \neq SwitchRight$$

where *Pt1* and *Pt2* are specific point objects of the station, *Position* is a variable of the objects and *Left* is a symbolic constant indicating that a point is in left-hand position. *Alloc* is a symbolic constant indicating that the point is part of an allocated train route.

5 The NP-Tools theorem prover

The theorem prover *NP-Tools* is an interface to the patented Stålmarek proof procedure for propositional logic [17, 18].

Briefly, the method is based on constructing sub-formula natural deduction proofs. The basic concept is that of *n*-saturation [23]. The *n*-saturation of a set of formulas is the set of all (sub-formula) conclusions that can be drawn using a proof with at most *n* simultaneous hypotheses. E.g., the 0-saturation of a set of formulas is the set of conclusions that can be drawn using a proof without any hypothetical reasoning at all.

In order to get as many conclusions as possible without the need for hypothetical reasoning, the proof procedure includes all valid simple inference rules involving a single logical connective. This includes rules that would be redundant under standard presentations of natural deduction, e.g. rules such as

$$\frac{, \vdash A \vee B \quad , \vdash \neg A}{, \vdash B}$$

and

$$\frac{, \vdash \neg A}{, \vdash \neg(A \wedge B)}$$

The proof procedure uses a single hypothetical inference rule expressing the principle of the excluded middle:

$$\frac{, , A \vdash B \quad , , \neg A \vdash B}{, \vdash B}$$

Proofs in NP-Tools are done as refutation proofs: to prove that a formula *B* follows from some assumption set *A*, the negation of *B* is added to *A* and the resulting set is *n*-saturated (for some *n*). If absurdity (contradiction) is among the conclusions of the saturation, *B* has been shown to follow from *A*. If a proof can be found using *n*-saturation, the problem is said to be *n*-easy.

If NP-Tools fails to find a proof, it will attempt to find a counter model, e.g. an assignment of truth values to propositional variables, where all formulas in *A* are true, but *B* is false.

Clearly, the lower the *n*, the faster the saturation. 0-saturation can be done in linear time in the size of the problem. General *n*-saturation can be done in time $O(s^{2n+1})$ where *s* is the size of the problem. The important property of the method is that “practical” problems can often be solved with *n* equal to 0 or 1 [11]. This means that in practise the procedure can handle very large problems.

In addition to the propositional reasoning, NP-Tools includes a (very) incomplete handling of integer arithmetic¹.

The size of a problem in NP-Tools is usually measured in the number of *triples*—a data structure of the theorem prover. Roughly, the number of triples corresponds to the number of operators (logical connectives,

¹Integer arithmetic is of course always incomplete since it is undecidable, but the arithmetic in NP-Tools is particularly weak, as the theorem prover is intended for propositional logic.

relational or arithmetical operators) in the the problem formulas.

For a fuller explanation of the Stålmarck method, see [16] and [23].

6 Verifying requirements using NP-Tools

By using NP-Tools to saturate a set consisting of the base formulas of a program and a negated requirement, we can use it to attempt to prove whether or not given program requirements are fulfilled.

LockAllocReq can be proved in two fundamentally different ways. We can either consider a generic point object, or consider each of the actual points in the railway station in question.

Considering a generic point object is the most elegant approach, since a successful proof implies that the requirement will be fulfilled for any point in any railway station whatsoever. On the other hand, since a generic object has no “real” neighbours, the proof must take into account every possible combination of input data from the potential neighbours—even combinations that could never be generated by an actual neighbour in an instantiated STERNOL program. This makes it possible that a generic object might fail to fulfill the requirement even if it will always be fulfilled by an object in an instantiated program.

Attempting to prove LockAllocReq for a generic point object turns out to be easy. The point object generates 5131 triples and 276 arithmetic variables. The requirement can be proved using 1-saturation in 65 seconds² or by using 0-saturation to prove two sub-requirements, each using about 2 seconds of runtime.

LockFlankReq is not so easily managed. A generic proof is not feasible since the requirement deals with the relation of the points to other objects. A direct attempt to prove the requirement for the points *Pt1* and *Pt2* fails. The problem turns out not to be 1-easy, and 2-saturating the problem (as well as finding a possible counter model) takes unreasonably long time. Another approach must be taken.

7 Reducing the state space of a program

The main reason why LockFlankReq causes troubles is that the state space of the program is very large.

²All timing figures refer to NP-Tools 2.1 running on a HP 715/80 workstation.

The size of the program, represented in NP-Tools, consists of about 388000 triples and 17000 integer variables. This gives us a potential state space of roughly $2^{388000} \approx 10^{116800}$. In order to overcome the state space problems, we try to reduce the state space of the program *with respect to* the requirement we are considering. The first approach is to only regard the program objects which are constrained by the requirement: if we can prove that such a cluster of objects cannot enter a state which would violate the requirement no matter how other parts of the program behaves, then we have proved that the program fulfills the requirement. LockFlankReq constrains variables of the objects *Pt1* and *Pt2* only, so we consider only the program fragment consisting of those two objects. This drastically reduces the state space of the problem. However, the reduction is still not enough, so further reductions are needed.

7.1 Program projections

Another idea is to eliminate all base formulas which do not affect the value of the “interesting” variables. Here the interesting variables are the ones mentioned by the requirement. We do so by “projecting” a program onto a set of variables.

An integer expression is constructed by integer constants, variables and the usual arithmetic operators. Define the *variables of an integer expression* by:

$$\begin{aligned} V_i(c) &= \emptyset & V_i(v) &= \{v\} & V_i(\Leftarrow e) &= V_i(e) \\ V_i(e_1 \text{ op } e_2) &= V_i(e_1) \cup V_i(e_2) \end{aligned}$$

Similarly, a boolean expression is constructed by relations between integer expressions and the usual boolean operators. Define the *variables of a boolean expression* by:

$$\begin{aligned} V_b(\neg be) &= V_b(be) \\ V_b(e_1 \text{ relop } e_2) &= V_i(e_1) \cup V_i(e_2) \\ V_b(be_1 \text{ bop } be_2) &= V_b(be_1) \cup V_b(be_2) \end{aligned}$$

Define the *immediate dependences* of a variable x in the base formula

$$\varphi \hat{=} v = e \Leftarrow be$$

as

$$ID(x, \varphi) \hat{=} \begin{cases} \{v\} \cup V_i(e) \cup V_b(be) & \text{iff } x \in \{v\} \cup \\ & V_i(e) \cup \\ & V_b(be) \\ \emptyset & \text{otherwise} \end{cases}$$

For a set of base formulas, Π , define

$$ID(x, \Pi) \hat{=} \bigcup_{\varphi \in \Pi} ID(x, \varphi)$$

Define the *dependences* of a variable, x , in the program, Π , as

$$D(x, \Pi) \hat{=} \{x\} \cup \bigcup_{y \in ID(x, \Pi)} D(y, \Pi)$$

Define the dependences of a set of variables, X , as:

$$D(X, \Pi) \hat{=} \bigcup_{x \in X} D(x, \Pi)$$

Finally, by *the variable of a base formula*, we write:

$$Var(var = expr \Leftarrow bexpr)$$

and by this mean *var*, i.e., the identifier on the left-hand side of the “=” sign.

We can now define the *projection of Π onto a set of variables, X* by

$$\varphi \in \text{project}(\Pi, X) \Leftrightarrow \varphi \in \Pi \wedge Var(\varphi) \in D(X, \Pi)$$

The idea of this projection is similar to that of program slicing [22, 8].

7.2 Modified projection

Unfortunately, a projection of the program onto the variables of LockFlinkReq does not lead to very large reductions. To obtain a more significant reduction, we can eliminate all base formulas which do not directly have an effect on the value of one of the variable which we project onto. Consider the base formula: $v = e \Leftarrow be$. This base formula influences the value of the variable v . So if v is one of the variable projected onto, this base formula should be included in the result. However, we do not necessarily want to include it if v is not one of the variables projected onto, even if one of these variables occur in the e or be . Thus, the base formulas of the projection-result are the ones determining the value of the interesting variables (and the variables they are controlled by etc.) and *not necessarily* the base formulas of variables which are *controlled by* the interesting variables. A new modified projection can be defined by changing the definition of immediate dependency:

$$ID(x, v = e \Leftarrow be) \hat{=} \begin{cases} V_i(e) \cup V_b(be) & \text{iff } v = x \\ \emptyset & \text{otherwise} \end{cases}$$

We shall refer to the modified projection as project_M , and the “complete” projection defined in Sect. 7.1 as project_C .

Theorem 1 *For all sets of base formulas, Π , all sets of variables, X , and all requirement formulas, ϕ : if $\text{project}_M(X, \Pi) \models \phi$ then $\text{project}_C(X, \Pi) \models \phi$.*

Proof. The complete projection contains everything that the modified projection does (a proof of this can be found in [16]). The extension theorem for propositional logic says that

$$\text{If } \Gamma \models \phi \text{ then } \Gamma, \Delta \models \phi$$

Since (by the property stated above)

$$\text{project}_C(X, \Pi) = \text{project}_M(X, \Pi) \cup \Delta,$$

for some set of base formulas Δ , Theorem 1 follows directly from the extension theorem. \square

Methods for reducing the size of problems has previously been addressed in the literature. The interlocking program for Alingsås could probably be reformulated such that the compositional model checking suggested by Clarke, Long and McMillan[5] could be applied. It seems, however impractical for a system as large as the interlocking system, and a lot of work would have to be done in parting the program into small processes etc. Clarke, Grumberg and Long has suggested to apply a number of abstractions depending on the nature of the problem [4]. It is not obvious, however, that these abstractions could be applied well to the interlocking program. At least, a lot of work would have to be spent in understanding in detail how the program works—the idea of this paper is to examine how far we can get without such insight. Also, the work of Kurshan[12] is probably related.

8 Verifying with projection

Because of Theorem 1 we can prove LockFlinkReq if we can prove that the requirement is fulfilled by the base formulas resulting from applying the modified projection onto the variables occurring in LockFlinkReq.

By using project_M after having made trivial reductions of the program, the size of the program objects *Pt1* and *Pt2* was reduced by 84%. This made it possible to try a 2-saturation on the problem in about three hours. Unfortunately, this was not enough to prove LockFlinkReq. This means that either we have to go to even higher saturation levels or that the requirement is not fulfilled. Trying a 3-saturation is infeasible, since this would take an enormous amount of time, so instead NP-Tools was asked to try to find a counter model for the requirement. After a few hours, a counter model for the reduced program was found. That is, LockFlinkReq is not fulfilled by the projected program.

8.1 False counter models

This does not necessarily mean that the requirement is not fulfilled, since it may not be possible to extend the counter model to be a counter model for the non-reduced program. Consider the following set of base formulas:

$$\begin{aligned}x = 0 &\Leftarrow z = 8 \wedge a \neq 6 \\x = 2 &\Leftarrow z = 7 \wedge a = 6 \\z = 7 &\Leftarrow a = 5 \wedge d = 8 \\z = 8 &\Leftarrow b = 2 \vee a \neq 5\end{aligned}$$

Suppose we project the formulas (using project_M) onto $\{z\}$. The result does not contain the base formulas for x . Suppose that there are no other base formulas for x and z . It is now possible to find a model (for the projection) where $z = 7$. We notice that this must mean that $a = 5$ (since exactly one of the two base formulas of z must have the expression on the right-hand side of \Leftarrow true). Considering the un-projected set, we see that this means that x must have the value 2 if $z = 7$, but it cannot have that value if $a = 5$. Hence, there is no well-formed variable assignment where $z = 7$. That is: it may be possible to find a model for a projection which cannot be extended to a model for the non-projected program.

The found counter models for LockFlankReq may not be true counter models after all!

8.2 Extending the counter model

Once the counter model for the problem reduced by a modified projection was found, it could be extended rather easily to a counter model for the same problem reduced by a complete projection (and this could again be extended to a model for $Pt1$ and $Pt2$). So, requirement LockFlankReq is *not* fulfilled by the part of the program consisting of the objects $Pt1$ and $Pt2$.

It is, however, still not certain that the counter model can be extended to a counter model for the entire program. Since there are a large number of counter models for the system consisting of $Pt1$ and $Pt2$ only, it may be quite difficult to extend the counter model to be a counter model for the entire program. Of course, we could add more and more objects, each time extending the model. However, if, at a certain point, the model cannot be extended, we would have to backtrack and try out all remaining possibilities. Doing so corresponds to letting the program find a counter model for the entire program—and we already know that takes too long time for any practical use.

This leaves us with an unanswered question: is LockFlankReq fulfilled by the program or not? All we can

say is that it is not fulfilled when we only consider the objects mentioned by the requirement. It may be that other objects of the program can never enter a state that forces $Pt1$ and $Pt2$ to get into a state where the requirement is not fulfilled, but we cannot tell.

This means, that either the requirement is not fulfilled, or the program has been made in a non-defensive manner: “impossible³” inputs from other program objects is not handled safely. Since such assumptions are not expressed anywhere, it makes the task of verifying the requirements very difficult.

9 Conclusion

We have presented experiences of formal verification of an existing railway interlocking program, based directly on the program source code rather than on an abstraction of the program behaviour. The idea was to find out how far we can come in verifying programs without a deep insight of the program structure. An insight which on one hand *may* help us decomposing the problems into several easier problems, but on the other hand is hard⁴ to obtain, especially if programs are poorly documented. Immediate conclusions are:

- By using the Stålmarck method (NP-Tools) we are able to prove properties of very large systems, provided that the proofs are 0- or 1-easy.
- As it turns out, “local” requirements (i.e., requirements dealing with one program object/physical device) tend to be 0- or 1-easy. LockAllocReq is an example of this.
- Requirements constraining multiple objects (such as LockFlankReq) often cannot be proved by 1-saturation. This means that either the requirement is more difficult to prove, or that it is not fulfilled.
- Reducing a problem (e.g., by projecting the problem onto a set of interesting variables) can help us prove 2-easy problems (but not harder problems in realistic examples).
- LockFlankReq was actually not fulfilled (when we only considered the objects mentioned by the requirement). This *could* be taken as an indication that the experiences of Logikkonsult that if a “real-life” problem is harder than 1-easy, then it most likely is not fulfilled. However, we do not have enough experience to jump to such conclusions.

³By tacit assumptions

⁴And therefore costly.

What can we do to be able to determine whether or not the requirements are fulfilled by programs on the industrial scale? One could claim that theorem provers are not yet mature to handle such programs, or one could claim that the programs are not yet mature to be handled by theorem provers.

Naturally, work should be done to improve the techniques of theorem provers, but it may very well take a long time before theorem provers will be able to prove properties of programs that have not been designed with formal verification in mind (such as the Alingsås interlocking program). Therefore, we might have more immediate success by demanding that programs should be designed such that they are manageable by today's theorem provers. For instance, one could require that the Alingsås interlocking program *must* be such that it is possible to prove requirements just by considering the parts of the program which are directly affected by the requirement. This may very well lead to better abstractions and better program design, and help making assumptions explicit.

Acknowledgements

I would like to thank Lars-Henrik Eriksson who has been supervising this work during my visit at Logikkonsult NP AB in January–July 1996 while I did my PhD at the Department of Information Technology, Technical University of Denmark.

References

- [1] ABB Signal AB. *Sternol Programming Language*, 1994. Doc. No. 3NSS100003A0004.
- [2] A. Anselmi, C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, and F. Torielli. An experience in formal verification of safety properties of a railway signalling control system. In G. Rabe, editor, *Proceedings of the 14th International Conference on Computer Safety, Reliability and Security (SAFECOMP'95)*, pages 474–488, Belgirate, Italy, October 11–13 1995. Springer Verlag.
- [3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(12):1035–1044, 1986.
- [4] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proceedings of the 19th Annual Symposium on Principles of Programming Languages, Santa Fe, New Mexico*, January 1992.
- [5] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 353–362, Asilomar Conference Center, Pacific Grove, California, June 5–8 1989. IEEE Computer Society Press.
- [6] Babak Dehbonei and Fernando Mejia. Formal development of safety-critical software systems in railway signalling. In Michael G. Hinchey and Jonathan P. Bowen, editors, *Applications of Formal Methods*, chapter 10. Prentice Hall, 1995.
- [7] Lars-Henrik Eriksson. Formalisering av krav på ställverk (slutrapport). Technical Report NP-K-LHE-003, Logikkonsult NP AB, December 1995. Preliminary version. In Swedish.
- [8] J.F. Groote, J.W.C. Koorn, and S.F.M. van Vlijmen. The safety guaranteeing system at station Hoorn-Kersenboogerd. Technical Report 121, Logic Group, Preprint Series, Department of Philosophy, Utrecht University, 1994.
- [9] Kirsten Mark Hansen. *Linking Safety Analysis to Safety Requirements—Exemplified by Railway Interlocking Systems*. PhD thesis, Department of Information Technology, Technical University of Denmark, 1996.
- [10] Kirsten Mark Hansen. Modelling railway interlocking systems. In *Computer Applications In Transportation Systems*, June 1996. Basel, Switzerland.
- [11] John Harrison. Stålmarch's algorithm as a HOL derived rule. In *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'96), Finland*, August 1996.
- [12] Robert P. Kurshan. *Computer Aided Verification of Coordinating Processes*, chapter 8: Reduction of Verification. Princeton University Press, 1994.
- [13] Matthew J. Morley. *Safety Assurance in Interlocking Design*. PhD thesis, Department of Computer Science, University of Edinburgh, 1996.
- [14] Jakob Lyng Petersen. En uformel beskrivelse af sternol. In Danish. Danish State Railways (DSB), 1995.
- [15] Jakob Lyng Petersen. Formal requirement verification of a swedish railway interlocking system. Technical Report IT-TR: 1997-005, Department of Information Technology, Technical University of Denmark, 1997.

- [16] Jakob Lyng Petersen. *Mathematical Methods for Validating Railway Interlocking Systems*. PhD thesis, Department of Information Technolog, Technical University of Denmark, 1998.
- [17] Gunnar Stålmärck. En metod och anordning för tautologicheckning. Swedish patent 467 076, 1989. Approved 1991.
- [18] Gunnar Stålmärck. A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula. U.S. patent 5 276 897, 1989. Approved 1994.
- [19] Gunnar Stålmärck and Mårten Säflund. Modelling and verifying systems and software in propositional logic. In B. K. Daniels, editor, *Proceedings of Safety of Computer Control Systems 1990 (SAFE-COMP'90)*, pages 31–36, Gatwick, UK, 1990. Pergamon Press.
- [20] Technical Committee CENELEC TC 9X. *Cenelec prEN 50126: Railway Applications—The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*, June 1997. Final Draft.
- [21] Technical Committee CENELEC TC 9X. *Cenelec prEN 50128: Railway Applications—Software For Railway Control And Protection Systems*, June 1997. Final Draft.
- [22] F. Tip. A survey of program slicing techniques. Technical Report CS-R9438, CWI (Centrum voor Wiskunde Informatica), Amsterdam, 1994.
- [23] Filip Widebäck. Stålmärck's notion of n-saturation. version 1. Logikkonsult NP AB, NP-K-FW-200, January 1996.
- [24] Hantao Zhang. SATO: A decision procedure for propositional logic. *Association for Automated Reasoning Newsletter*, 22(1–3), March 1993.

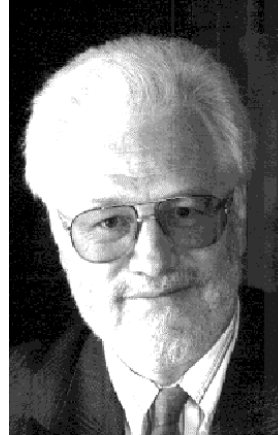
„Risk Management Technology:

A rich practical toolkit for identifying, documenting, analyzing

By Tom Gilb,
Senior Partner,
Result Planning Limited

TomGilb@Result-Planing.com
URL www.Result-Planning.com

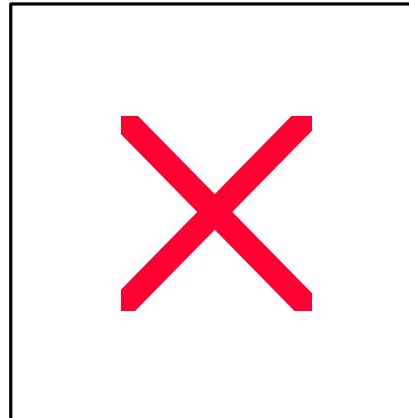
For Quality Week Brussels, Nov 11-
13 1998, Talk 4PM Thursday 12th
November 1998



"Risk Management Technology" ©Gilb@acm.org

A broad definition of 'Risk'

- Risk: is an abstract concept covering the area of control of
- A 'risk' is anything which itself can lead to negative unpredictable results.
- Risk Management is any activity which identifies risks, and takes action to remove, reduce or control 'negative results'.
- 'Negative results' are any measurable or testable outcomes due to activity or inactivity, which are worse than the results planned or expected.



"Risk Management Technology" ©Gilb@acm.org

A method for specification of risk level objectives

Slide
3

- We must go through the following
 - Identify *all critical* quality and cost *aspects* of the system where we would like to control risk.
 - Define exactly how to *understand variation* in each critical aspect by specifying a scale of measure
 - Define one or more critical points on the defined scale of measure which are needed and expected for the system to function properly
 - Define Must and Plan levels for any important to you aspects of time, place and event. We call this using 'qualifier'.
- We must go through the following steps:
 - **Availability**
 - **Scale: % System up 24H 7Day**
 - **Must [Acceptance Test] 98%**
 - **Plan [First Year Oper.] 99.90%**
 -

"Risk Management Technology" ©Gilb@acm.org

Specification Examples

Slide
4

- **Qualifier: When, Where, IF**
 - Avoids risk of change or overgeneralization
 - **Direct specification of uncertainty using \pm**
 - **Generic parameter "[defined tasks]"**
 - **Specific Parameter**
 - "Modifying Files"
- **Plan [1999, Europe, IF no war in**
 - **Must [2001, UK, IF Euro is used in Norway & UK] 60% \pm 20%**
 - **Usability:**
 - **Scale: Mean Time to learn [defined tasks] to Minimum proficiency**
 - **Must [Release 2.0, English Version, Task: Modifying Files] 10 minutes.**
 - **Plan [Release 3.0, French & Dutch Versions, Task: Finding a File by Content] 5 minutes.**

"Risk Management Technology" ©Gilb@acm.org

Risk analysis of using particular solutions

Slide
5

- The most critical (failure of system) risk is that the first set of conditions has a time to learn exceeding 10 minutes.
 - The secondary risk is the second set of conditions and a failure there to learn in 5 minutes or less.
 - It should be obvious that the degree of risk can be expressed in terms of the deviation from the target levels.
 - Means that method A poses a real risk and method B does not, for meeting the specified targets.
- Usability:
 - Scale: Mean Time to learn [defined tasks] to Minimum proficiency
 - Must [Release 2.0, English Version, Task: Modifying Files] 10 minutes.
 - Plan [Release 3.0, French & Dutch Versions, Task: Finding a File by Content] 5 minutes.
 - For example
 - Method A can sometimes result in a learning time of 10 minutes, while method B can never result in a learning time exceeding 4 minutes.

"Risk Management Technology" ©Gilb@acm.org

Specifying Uncertainty (a Risk)

Slide
6

- Plan 60-80
- Plan 60±30
- Plan 60 → 90
- Plan 60?
- Plan 60??
- Plan 60 ← Dubious Source
- Plan <60> <Fuzzy Brackets>
indicate 'data needing improvement'

"Risk Management Technology" ©Gilb@acm.org

Avoiding risk of requirements not needed

Slide
7

- Plan [IF NATO includes Russia as Full Member] 99%
- Risk is controlled by making the specification totally dependent on the IF condition.
- There is no risk that anyone will plan to achieve 99% IF the condition is false.
- But they are warned to plan to achieve 99% should the condition turn true.

"Risk Management Technology" ©Gilb@acm.org

Avoiding wrong strategy risk

Slide
8

- Strategy99
- [IF Hunger Famine in a country, IF Road and Rail Transport Unavailable]
- Aerial Supply of Food OR Aerial Removal of Refugees to Food Supply
- This shows how a strategy (a means for achieving a goal) can be specified, and made totally dependent on one or more conditions in the Qualifier.
- This reduces the risk that an expensive strategy is applied under inappropriate conditions,
- and that, for example, 'cost budgets' risk being threatened.

"Risk Management Technology" ©Gilb@acm.org

Risk Policy Summary

Slide
9

- **.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
- **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 'Meters' (test or measuring instruments for determining where we are
- **CONDITIONS SPECIFIED**
- The requirements levels shall be qualified with regard to when where and under which conditions the targets apply, so there is no
- **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
- **.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 '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
 - .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.

10 Principles of Risk Management

Slide
10

1. Frequent Feedback
 2. Rigorous Requirements
 3. Requirement Impact Estimation
 4. Upstream Pollution Control
 5. Personal Risk Responsibility
 6. Design Out Risk
 7. Maximum Risk Policy
 8. Maximize profit, not minimize Risk itself
 9. Backups are part of the Price
 10. Contract Out Risk
- Early frequent and measurable feedback from reality must be planned into your development process, to identify
 - All critical success-and-failure quality/performance/cost requirements must be identified, made measurable and tracked through design and evolutionary deployment.
 - A design phase must address all critical few requirements and systematically estimate the impact of all design
 - 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.
 - People must be give personal responsibility in their sector for identification and mitigation of risks.
 - 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.
 - 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.
 - Focus not on elimination of all risk, but on maximization of benefit to cost result delivery, even considering risks.
 - Conscious planning and development of backup for risks is a necessary minimum cost of planning and projects.
 - Make vendors contractually responsible for risks, they will give you better advice and services as a result. ■

TWELVE TOUGH QUESTIONS

Slide 11

- 1. Why isn't the improvement quantified?
- 2. What is degree of the risk or uncertainty and why?
- 3. Are you sure? If not, why not?
- 4. Where did you get that from? How can I check it out?
- 5. How does your idea affect my goals, measurably?
- 6. Did we forget anything critical to survival?
- 7. How do you know it works that way? Did it before?
- 8. Have we got a complete solution? Are all objectives satisfied?
- 9. Are we planning to do the 'profitable things' first?
- 10. Who is responsible for failure or success?
- 11. How can we be sure the plan is working, during the project,
- 12. Is it 'no cure, no pay' in a contract? Why not?

There is a detailed paper on these questions at www.result-planning.com

Impact Estimation

Slide 12

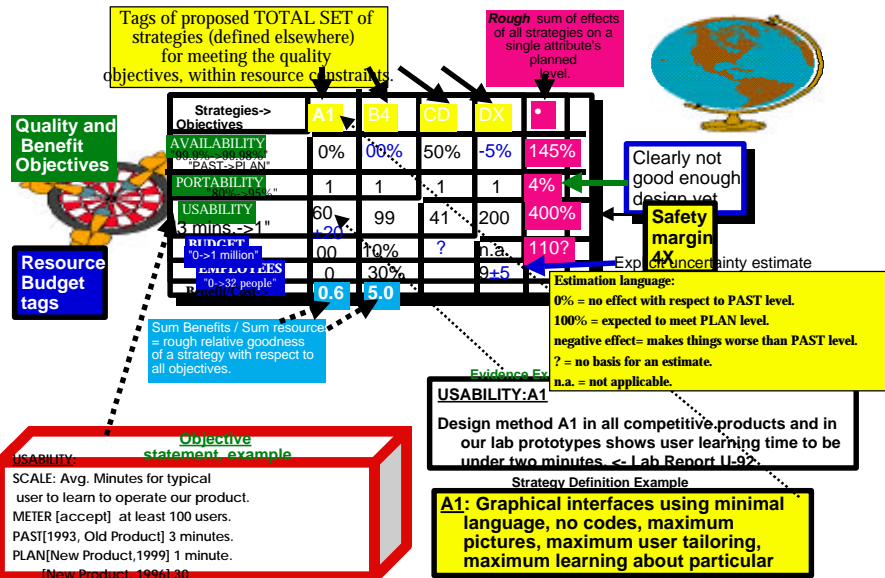
(Tags of defined Strategies-> Objectives	A1	B4	CD	DX	
"99.9%->99.98%" "PAST->PLAN"	0%	100%	50%	-5%	145%
"80%->95%"	1%	1%	1%	1%	4%
"3 mins.->1"	60% +20	99%	41%	200%	400%
"0->1 million"	100%	10%	?	n.a.	110?
"0->32 people"	0%	30%		9%±5	
Benefit/Cost->	0.6	5.0			

Lieuw Zieglerman of Dutch Rail suggested PAST->PLAN notation

"Risk Management Technology" ©Gih@acm.org

Advanced "Impact Estimation" concepts

Slide 13



"Risk Management Technology" ©Gilb@acm.org

Credibility Rating (for Impact Table)

Slide 14

- | Cred. Score | Meaning |
|-------------|---|
| 0.0 | wild guess, no credibility |
| 0.1 | we know it has been done somewhere |
| 0.2 | we have one measurement somewhere |
| 0.3 | there are several measurements in the estimated range |
| 0.4 | the measurements are relevant to our case |
| 0.5 | the method of measurement is considered reliable |
| 0.6 | we have used the method in-house |
| 0.7 | we have reliable measurements in-house |
| 0.8 | reliable in-house measurements correlate to independent external measurements |
| 0.9 | we have used the idea on this project and measured it |
| 1.0 | perfect credibility, we have rock solid, contract-guaranteed, long-term, credible experience with this idea on this project and, the results are unlikely to disappear |

"Risk Management Technology" ©Gilb@acm.org

Impact Tables and Risk

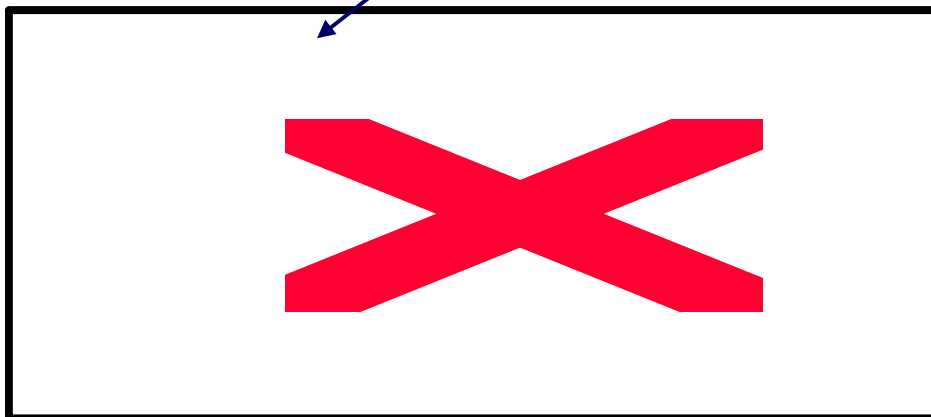
Slide
15

- **Forces thorough Analysis**
 - Of all cost/quality impacts
 - Based on facts, not opinion
- **Analysis is documented**
- **Analysis can be quality controlled**
- **Risk is explicit**
 - Credibility rating
 - Safety factors
- **Forces better definition specification**
 - Requirements
 - Designs
 - evidence
- **Acceptable Risk levels can be controlled**
 - **By Setting safety factor limits in Rules for specification**
 - “defect’ if not met
 - **By setting exit/entry levels for Credibility averages**
 - Unacceptable/not completed if we fail to meet these levels

“Risk Management Technology” ©Gilb@acm.org

Evo Plan: Deviation = Risk
Identified during project : ‘for real’

Slide
16

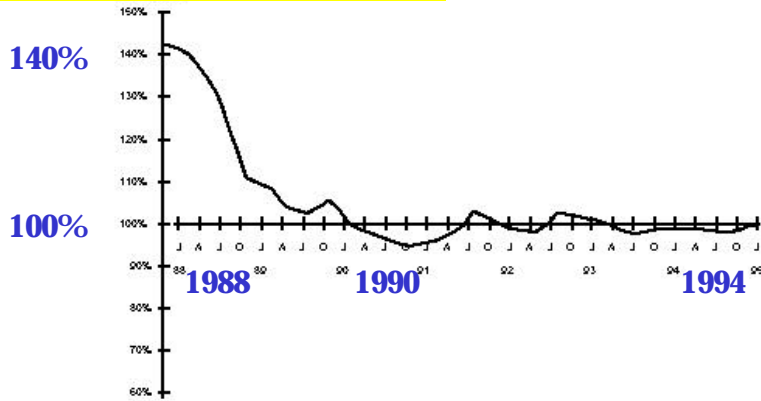


“Risk Management Technology” ©Gilb@acm.org

Achieving Project Predictability: Raytheon

Slide 17

Cost At Completion / Budget %



Tuesday, October 13, 1998

Copyright Gilh@acm.org

Slide 17

Cost of Quality over Time: Raytheon 95

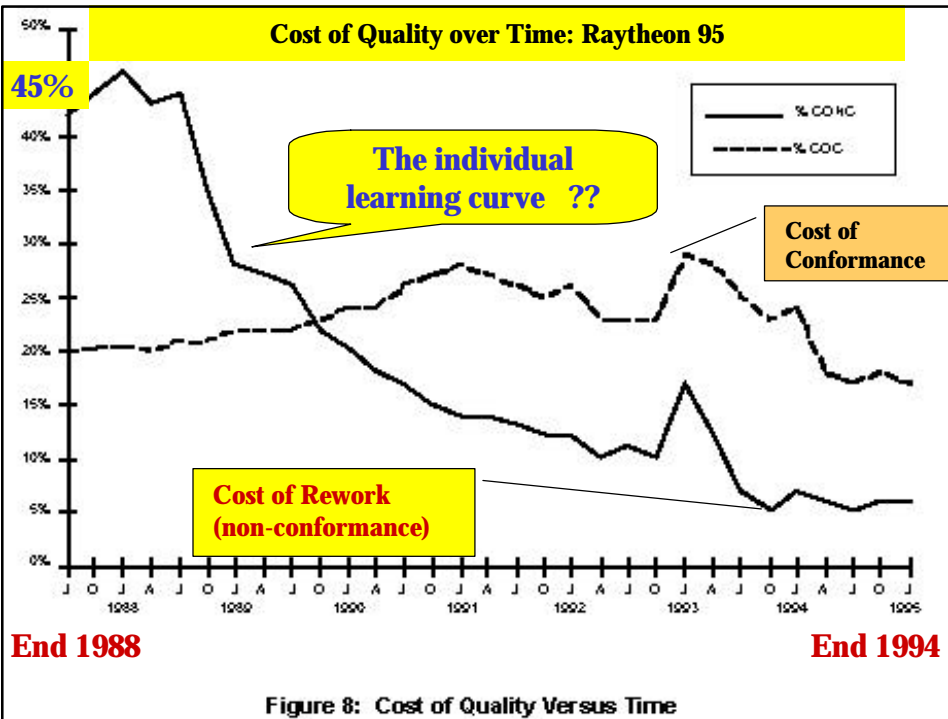
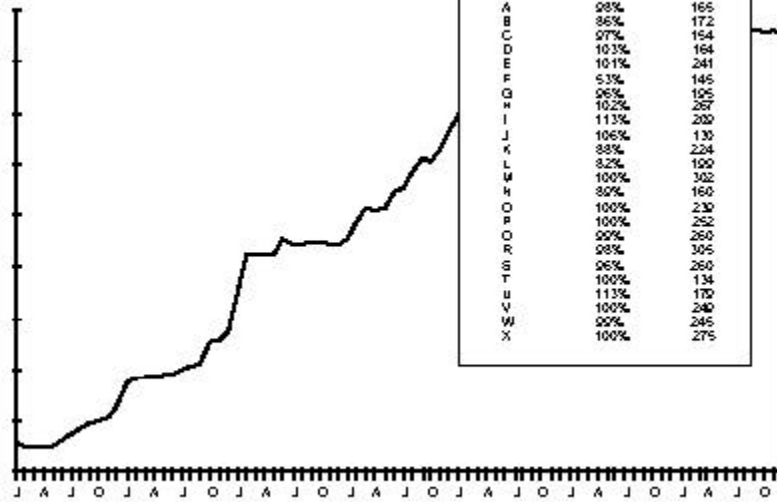


Figure 8: Cost of Quality Versus Time

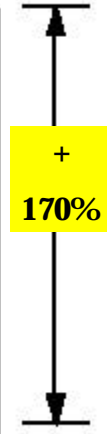
Raytheon 95 Software Productivity 2.7X better

Slide 19

Productivity



Project	CAC/BU D	Productivity
A	98%	165
B	96%	172
C	97%	154
D	103%	164
E	101%	241
F	53%	145
G	96%	195
H	102%	267
I	113%	230
J	106%	130
K	98%	224
L	82%	190
M	100%	302
N	80%	160
O	100%	230
P	100%	252
Q	90%	260
R	98%	305
S	96%	260
T	100%	134
U	113%	170
V	100%	240
W	90%	245
X	100%	275

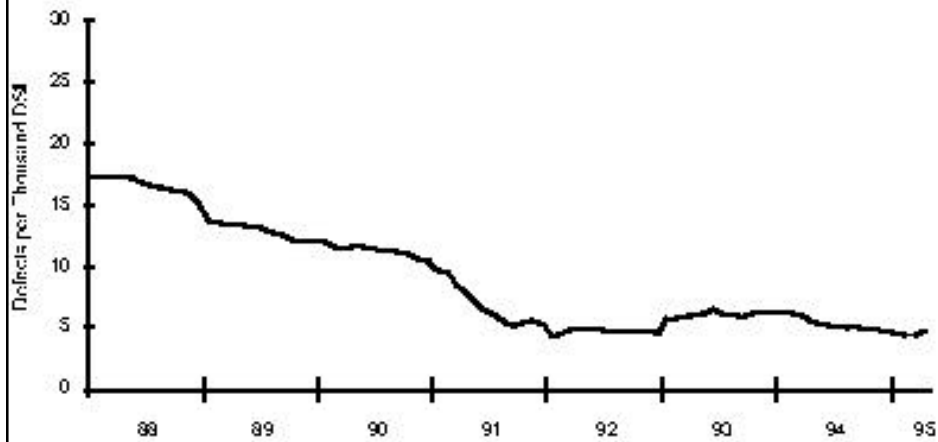


1988

1994

Overall Product Quality: Raytheon 95
Defect Density Versus Time

Slide 20



Version 3.0 Sept. 27 1998

For Quality Week in Brussels, 11th-13th November 1998.
Talk at 4pm on Thursday, 12th November.

"Risk Management : A practical toolkit for identifying, analyzing and coping with project risks"

**By Tom Gilb,
Senior Partner, Result Planning Limited**

Paper Summary

Risk management must be fully integrated into all the development and maintenance processes for systems. It involves more than applying risk assessment methods to identify and evaluate system risks.

To explain this broad approach to risk management, this paper discusses the way in which Requirements Driven Management (RDM) methods contribute to handling risks.

Definition of ‘Risk’

Risk is an abstract concept expressing the possibility of unwanted outcomes.
A ‘risk’ is anything which can lead to results that deviate from the requirements.

It is in the nature of risk that the probability of risks actually occurring, and their actual impact when they do so, can only be predicted to varying degrees of accuracy. Not all risks can be identified in advance.

Risk Management is any activity which identifies risks, and takes action to remove, reduce or control ‘negative results’ (deviations from the requirements).

Principles of Risk Management

In my view, the fundamental principles of risk management include:

1. Quantify requirements

All critical quality and resource requirements must be identified and quantified numerically.

2. Maximize profit, not minimize risk

Focus on achieving the maximum benefits within budget and time-scales rather than on attempting to eliminate all risk.

3. Design out unacceptable risk

Unacceptable risk needs to be ‘designed out’ of the system consciously at all stages, at all levels in all areas, e.g. architecture, purchasing, contracting, development, maintenance and human factors.

4. Design in redundancy

When planning and implementing projects, conscious backup redundancy for outmaneuvering risks is a necessary cost.

5. Monitor reality

Early, frequent and measurable feedback from reality must be planned into your development and maintenance processes, to identify and assess risks before they become dangerous.

6. Reduce risk exposure

The total level of risk exposure at any one time should be consciously reduced to between 2% and 5% of total budget.

7. Communicate about risk

There must be no unexpected surprises. If people have followed guidelines and are open about what work they have done, then others have the opportunity to comment constructively. Where there are risks, then share the information.

8. Reuse what you learn about risk

Standards, rules and guidance must capture and assist good practice. Continuous process improvement is also needed.

9. Delegate personal responsibility for risk

People must be given personal responsibility in their sector for identification and mitigation of risks.

10. Contract out risk

Make vendors contractually responsible for risks, they will give you better advice and services as a result.

Let’s now consider, each of these principles in turn and describe some (not all!) of the roles that the RDM methods play in risk management. However, first here is an outline sketch of the RDM methods:

- **Planguage**; a requirements specification language insisting on quantified values.
- **Impact Estimation (IE)**; an analysis tool (a table) allowing evaluation of the likelihood of achieving requirements and, the evaluation and comparison of different designs (strategies). A strength of IE is that it also helps identify new designs and uncover previously unstated requirements.
- **Evolutionary Delivery (Evo)**; based on the work by the quality gurus Deming and Juran, a way of working that focuses on evolutionary delivery of early, measurable, system benefits to the customers. A system is developed, by small risk steps, in a series of plan, develop, deliver and evaluate cycles.
- **Inspection**; a technique for measuring and improving technical document quality. Technical documents are evaluated against their source documents and any prevailing standards by Inspection teams consisting of individuals with specially assigned roles. The overall aims are to identify defects, to identify patterns in the introduction of defects (leading to process improvement), to help train individuals to avoid creating defects and, to assist team-building.

Readers wanting a more detailed explanation of these methods should look in the References.

Principle 1. Quantify requirements

All critical quality and resource requirements must be identified and quantified numerically.

Risk is negative deviation from requirements. So, if we are going to understand risk, we must have some way of specifying exactly what we want. If we use vague ways like “State of the Art, World Class, Competitor-Beating Levels of Quality”, we cannot understand and assess risk.

Planguage helps because it demands numerically quantified requirements. Using Planguage, we must go through the following steps:

- Identify *all critical* quality and resource *attributes* of the system. In practice, this could be ten or more critical qualities (e.g. availability) and, five or more critical resources (e.g. operational costs).

- Define exactly how to *understand variation* in each attribute by specifying a scale of measure, e.g. ‘Scale: Probability of being fully operational during the office day’ and ‘Scale: Total of all monetary operational expenses including long term
- For each attribute, define one or more critical points on the defined scale of measure which are needed for the system to function properly and profitably. There are two important categories: ‘Must’ and ‘Plan’. A ‘Must’ level defines the system survival level. A ‘Plan’ level defines the planned point for success. For risk management, ‘Must’ is the first level and ‘Plan’ is the second level for risk determination. A value for any attribute less than its required Must level means total system failure. Only when all Plan levels for all the attributes have been met can a system be declared a success.
- For all the Must and Plan levels, define additional qualifying information. We call this using ‘qualifiers’. You are basically defining time, place and event, i.e. when it is critical for you to achieve a certain level of an attribute, where it is critical and under what conditions. For example,

Plan [1999, Europe, IF European Monetary Union implemented anywhere] 99.98%

We can even give direct expression to the amount of risk we are prepared to take by a statement such as :

Must [2001, UK, IF Euro is used in Norway & UK] 60% \pm 20%

In other words the range of results 40% to 80% is an acceptable upper and lower limit, but below 40% is unacceptable.

Here is a more complete example:

Usability:

Scale: Mean time to learn [defined tasks] to minimum proficiency.

Must [Release 2.0, English Version, Task: Modifying Files] 10 minutes.

Plan [Release 2.0, English Version, Task: Modifying Files] 7 minutes.

Plan [Release 3.0, English Version, Task: Modifying Files] 5 minutes.

Plan [Release 3.0, French & Dutch Versions, Task: Finding a File by Content] 5 minutes.

In the example, the most critical (failure of system) risk is the Must level. The other statements are only of secondary risk; they indicate the levels required to declare success.

It should be obvious that the degree of risk can be expressed in terms of the deviation from the target levels. For example,

Method A can sometimes result in a learning time of 10 minutes, while method B can never result in a learning time exceeding 4 minutes.

This means that for the specified requirements, method A poses a real risk, but method B does not.

A template specification of risk levels

In addition to the basic statements described above, it should be noted that there are a wide variety of ways within Planguage to indicate that the information contains some element of risk. Here are some examples:

Plan 60-80	Specification of a range
Plan 60±30	Specification of an upper and lower limit
Plan 60 → 90	
Plan 60?	Expressing that the value is in doubt
Plan 60??	Expressing that the value is in serious doubt
Plan 60 ← A wild guess	Using the source of the information to show the doubt
Plan 60 ← A.N. Other	Depends on A.N. Other’s credibility in setting this value
Plan <60>	Fuzzy brackets indicate data needing improvement

All of the above signals can be used to warn of potential risk. Of course, the culture must encourage such specification rather than intimidate people from using it.

Plan [IF Euro is used in UK] 99%

The above is an example where the risk is controlled by making the specification totally dependent on the IF condition. There is no risk that anyone will plan to achieve 99% if the condition is false. However, they are warned to plan to achieve 99% should the condition turn true.

Note, you can also use IF qualifiers to constrain the use of a strategy (a means for achieving a goal). This reduces the risk that an expensive strategy is applied under inappropriate conditions.

Strategy99 [IF hunger famine in a country, IF road and rail transport unavailable] Aerial Supply of Food.

Principle 2. Maximize profit, not minimize risk

Focus on achieving the maximum benefits within budget and time-scales rather than on attempting to eliminate all risk.

Elimination of all risk is not practical, not necessary and, not even desirable. All risk has to be controlled and balanced against the potential benefits. In some cases, it is appropriate to decide to use (and manage) a strategy with higher benefits and higher risks. I use Impact Estimation (IE) to help me assess the set of strategies I need to ensure I meet the required objectives. My focus is always on achieving the objectives *in spite of* the risks.

Outline Description of Impact Estimation (IE)

The basic IE idea is simple: estimate quantitatively how much your design ideas impact all critical requirements. This is achieved by completing an IE table. The left-hand column of the table should contain the objectives and, across the top of the table should be the proposed strategies. For the objectives, assuming you have expressed them using Planguage, it is a question of listing down all the quality and resource attributes you wish to consider. You need next to decide on a future date you want to use. This should be a system ‘milestone’; a date for which you have specified Must and Plan levels. Then, against each attribute, you state the current level and the Plan level for your chosen date. (If you are especially risk averse you would use the Must level!) For the strategies, you simply list them across the top of the IE table.

You then fill in the table, for each cell you answer the question, ‘How does this strategy move the attribute from its current level towards the Plan level?’ First you state the actual value you would expect and then you convert this into a percentage of the amount of required change.

For example, Training Time for Task A is currently 15 minutes and you require it to be 10 minutes within six months. You estimate Strategy B will reduce Training Time for Task A to 12 minutes. In other words, Strategy B will get you 60% of the way to meeting your objective. See Table 1.

TABLE 1

	Strategy B	
	Real Impact	% Impact
Training Time Past = 15 minutes in June 1998 Plan = 10 minutes by end of Dec. 1998	12 minutes	60%
Resource = Development Budget Plan = \$2000 up to end Dec. 1998	\$1,000	50%

Further improvements to specifying the impacts

There are a number of improvements to this basic idea, which make it more communicative and credible. Here is a brief summary of them :

Uncertainty of Impact: you can specify a range of values rather than a single value.

Evidence for Impact Assertion: you can state the basis for making your estimate. For example: "Strategy B was used for 5 projects last year in our company, and the percentage improvement for Training Times was always 60% to 80%".

Source of Evidence for Impact Assertion: Of course, some skeptic might like to check your assertion and evidence out, so you should give them a source reference, e.g. "Company Research Report ABR-017, pages 23-24."

Credibility Rating of the Impact Assertion: We have found it very useful to establish a numeric 'credibility' for an estimate, based on the credibility of the evidence and the source. We use a scale of 0.0 to 1.0 (because it can then be used later to modify estimates in a conservative direction). See Table 2.

TABLE 2

Credibility Rating	Meaning
0.0	wild guess, no credibility
0.1	we know it has been done somewhere
0.2	we have one measurement somewhere
0.3	there are several measurements in the estimated range
0.4	the measurements are relevant to our case
0.5	the method of measurement is considered reliable
0.6	we have used the method in-house
0.7	we have reliable measurements in-house
0.8	reliable in-house measurements correlate to independent external measurements
0.9	we have used the idea on this project and measured it
1.0	perfect credibility, we have rock solid, contract-guaranteed, long-term, credible experience with this idea on this project and, the results are unlikely to disappear

Further Analysis of the IE data

Once you have completed filling in all the impacts, there are a number of calculations, using the percentage impact estimates (%Impact), that help you understand the risks involved with your proposed solution.

Let me stress that these are only rough, practical calculations. Adding impacts of different independent estimates for different strategies, which are part of the same overall architecture, is dubious in terms of accuracy. But, as long as this is understood, you will find them very powerful when considering such matters as whether a specific quality goal is likely to be met or which is the most effective strategy. The insights gained are frequently of use in generating new strategies.

Impact on a Quality: For each individual quality or resource attribute, sum all the percentage impacts for the different strategies. This gives us an understanding of whether we are likely to make the planned level for each quality or cost. Very small quality impact sums like '4%' indicate high risk that the architecture is probably not capable of meeting the goals. Large numbers like 400% indicate that we might have enough design, or even a 'safety margin'.

TABLE 3

Example: Adding the percentage impacts for a set of strategies on a single quality or cost can give some impression of how the strategies are contributing overall to the objectives. Note Strategies A, B and C are independent and complementary.

	Strategy A	Strategy B	Strategy C	Sum of Strategy Impacts	Sum Uncertainty
Reliability 900->1000 hours MTBF	0+/-10%	10+/-20%	50+/-40%	60%	+/-70%

Impact of a Strategy: For each individual strategy, sum all the percentage impacts it achieves across all the qualities to get an estimate of its overall effectiveness in delivering the qualities. The resulting estimates can be used to help select amongst the strategies. It is a case of selecting the strategy with the highest estimate value and the best fit across all the critical quality requirements. If the design ideas are complementary then the aim is to choose which strategies to implement first. If the strategies are alternatives, then you are simply looking to determine which one to pick.

TABLE 4

A measure of the effectiveness of strategy ‘Big Idea’ can be found by adding together its percentage impacts across all the qualities

QUALITY	PAST-PLAN	Big Idea	
Reliability	900->1,000 hours MTBF	50%+/-10%	
Maintainability	10 min. fix to 5 min. to fix.	100%+/-50%	
		150%+/-60%	Estimate of total effect of Big Idea on all goals

In addition to looking at the effectiveness of the individual strategies in impacting the qualities, the cost of the individual strategies also needs to be considered, see next section.

Quality to Cost Ratio: For each individual strategy, calculate the quality-to-cost ratio (also known as the benefit-to-cost ratio). For quality, use the estimate calculated in the previous section. For cost, use the percentage drain on the overall budget of the strategy or use the actual cost.

The overall cost figure used should take into account both the cost of developing or acquiring the strategy and, the cost of operationally running the strategy over the chosen time scale. Sometimes, specific aspects of resource utilization also need to be taken into account. For example, maybe staff utilization is a critical factor and therefore a strategy that doesn't utilize scarce programming skills becomes much more attractive.

My experience is that comparison of the 'bang for the buck' of strategies often wakes people up dramatically to ideas they have previously under- or over-valued.

Average Credibility / Risk Analysis: Once we have all the credibility data (i.e. the credibility's for all the estimates of the impacts of all the strategies on all the qualities), we can calculate the average credibility of each strategy and, the average credibility of achieving each quality. This information is very powerful, because it helps us understand the risk involved. For example, "the average credibility, quality controlled, for this alternative strategy is 0.8". Sounds good! This approach also saves executive meeting time for those who hold the purse strings.

Principle 3. Design out unacceptable risk

Unacceptable risk needs to be 'designed out' of the system consciously at all stages, at all levels in all areas, e.g. architecture, purchasing, contracting, development, maintenance and human factors.

Once you have the completed initial IE table, you are in a position to identify the unacceptable risks and design them out of the system. Unacceptable risks include:

- Any quality or resource attribute where the sum of the %Impacts of all the proposed strategies does not total 200%. (A 100% safety factor has been assumed to reduce the risk of failure.)
- Any strategy providing i) a low total for the sum of its %Impacts, ii) very low credibility or iii) low benefit-to-cost ratio.

New strategies will have to be found that reduce these risks. In some cases, it may be decided that the levels set for the objectives are unrealistic and they may be modified instead.

Within software engineering, the art of designing a system to meet multiple quality and cost targets, is almost unknown [GILB88]. However, I have no doubt that there is great potential in conscious design to reduce risks. For example, it is a hallowed engineering principle to be conservative and use known technology. However, this concept has not quite caught on in software engineering technology, where ‘new is good’, even if we do not know much about its risks. At least, with the use of an IE table there is a chance of expressing and comparing the risk involved in following the differing strategies.

Principle 4. Design in redundancy

When planning and implementing projects, conscious backup redundancy for outmaneuvering risks is a necessary cost.

Under Principle 3, we have discussed finding new strategies. Principle 4, takes this idea a step further. Actively look for strategies that provide backup. An extreme example of this practice is NASA’s use of backup computer systems for manned space missions.

Principle 5. Monitor reality

Early, frequent and measurable feedback from reality must be planned into your development and maintenance processes to identify and assess risks before they become dangerous.

I expect the IE information only be used as an initial, rough indicator to help designers spot potential problems or select strategies. Any real estimation of the impact of many strategies needs to be made by real tests (Ideally, by measuring the results of early evolutionary steps in the field). Evolutionary Delivery (Evo) is the method to use to achieve this (See next Principal).

Principle 6. Reduce risk exposure

The total level of risk exposure at any one time should be consciously reduced to between 2% and 5% of total budget.

IE can also be used to support Evolutionary Delivery (Evo) as a budgeting and feedback mechanism during project building and installation of partial deliveries [GILB98, MAY96].

The Evolutionary Delivery (Evo) method typically means that live systems are delivered step by step to user communities for trial often (e.g. weekly) and early (e.g. 2nd week of project).

One of the major objectives of Evo is to reduce and control risk of deviation from plans. This is achieved by:

- getting realistic feedback after small investments
- allowing for change in requirements and designs as we learn during the project
- investing minimum amounts at any one time (2% to 5% of project time or money) so that total loss is limited if a delivery step totally fails.

IE is of use in helping to plan the sequencing of Evo steps. IE tables also provide a suitable format for presenting the results of Evo steps. See Table 5.

TABLE 5

Step-> Attribute	STEP1 plan %	actual %	Devia- tion %	STEP2 to STEP20 plan	plan cumul- ated to here	STEP21 [CA,NV,WA] plan	plan cumul- ated to here	STEP22 [all others] plan	plan cumul- ated to here
QUAL-1	5	3	-2	40	43	40	83	-20	63
QUAL-2	10	12	+2	50	62	30	92	60	152
QUAL-3	20	13	-7	20	33	20	53	30	83
COST-A	1	3	+2	25	28	10	38	20	58
COST-B	4	6	+2	38	44	0	44	5	49

Table 5 is a hypothetical example of how an evolutionary project can be planned and controlled and risks understood. The ‘deviation’ between what you planned and what you actually measured in practice is a good indicator of risk. The larger the deviation, the less you were able to correctly predict about even a small step. Consequently there is a direct measure of areas of risk in the ‘deviation’ numbers.

The beauty of this, compared to conventional risk estimation methods [HALL98] is that it:

- is based on *real* systems and *real* users (not estimates and speculation *before* practical experience)
- is *early* in the *investment* process
- is based on the results of *small* system increments, and the *cause* of the risk is easier to spot, and perhaps to eliminate, or to modify, so as to avoid the risk.

Evolutionary Project management does not ask what the risks *might* be. It asks what risks have shown up in practice. But it does so at such an *early* stage, that we have a fair chance to do something about the problems.

Principle 7. Communicate about risk

There must be no unexpected surprises. If people have followed guidelines and are open about what work they have done, then others have the opportunity to comment constructively. Where there are risks, then share the information.

Hopefully, readers will by now have begun to understand that PLanguage and IE are good means of communicating risk. Let me now introduce Inspection as a third useful method.

Inspection is a direct weapon for risk reduction. [GILB93]. Early Inspections on all written specifications is a powerful way to measure, identify and reduce risk of bad plans becoming bad investments. The key idea is that Major defects are measured, removed, and that people learn to avoid them, by getting detailed feedback from colleagues. A *defect* is a violation of a ‘best practice’ rule. A *Major* defect is defined as a defect which can have substantial economic effect ‘downstream’ (in practice, in ‘test’ phases and in the field). By this definition, a Major defect is a *risk* ! So Inspection measures risks!

Many people think that the main benefit from Inspection is in identifying and removing Major defects early (e.g. before source code reaches test phases). This is not the case. (My experience is that Inspection is as bad as testing in % defect-removal effectiveness. In very rough terms half of every defect present is not identified or removed.) The really important economic effect of Inspection is not what happens at the level of a single document, but in teaching the people and the organization. The real effect of Inspection is in:

- • teaching individual engineers exactly how often they violate best practice rules
- • motivating the engineers to take rules seriously (really avoid injecting Major defects)
- • regulating flow of documentation, so that high Major defect documents can neither exit nor enter adjacent work processes.

Staff involved in Inspections learn very quickly how to stop injecting defects. Typically, the defects introduced by an author reduce at the rate of about 50% every time a new document is written and Inspected. For example, using Inspection, Raytheon reduced ‘rework’ costs, as a % of development costs, from 43% to 5% in an eight year period [DION95].

Sampling

One other little-appreciated aspect of Inspection is that you can use it by *sampling* a small section of a large document, rather than trying to ‘clean up’ the entire document. If the sample shows a high Major defect density (say more than one Major/Page) then the document is probably ‘polluted’ and action can be taken to analyze the defect sources. A complete rewrite may be necessary using appropriate specification rules or new/improved source documents. This is generally cheaper than trying to clean up the entire document using defect removal Inspection or testing.

Principle 8. Reuse what you learn about risk

Standards, rules and guidance must capture and assist good practice. Continuous process improvement is also needed.

In the previous section, the importance of Inspection was discussed and rules were highlighted as one of the essentials required to support it. It is worth emphasizing the aspect of reuse that is occurring here. The more effort that is put into making rules more effective and efficient by incorporating feedback from Inspections, the more productive the Inspections and the greater the reduction in risk.

Even more benefit can be achieved if what is learnt from Inspection is used to modify the processes that are causing the defects; Continuous Process Improvement has been shown to have a major influence on risk. For example, Raytheon has achieved zero deviation from plans and budgets over several years. They used a \$1million/year (for 1,000 software engineers) for 8 years to do continuous software process improvement. They report that the return on this investment was \$7.70 per \$1 invested on improving processes such as requirements, testing and Inspection itself. Their software defect rate went down by a factor of three [DION95].

Using Inspection, analysis of the identified defects to find process improvements is carried out in the Defect Prevention Process (DPP). DPP was developed from 1983 at IBM by Robert Mays and Carole Jones and, is today recognized as the basis for SEI CMM Level Five. The breakthrough in getting DPP to work, compared to earlier failed efforts within IBM, was probably in the decentralization of analysis activity to many smaller groups, rather than one ‘Lab Wide’ effort by a Quality Manager. This follows what the quality Guru Dr. W Edwards Deming taught the Japanese; factory workers must analyze their own statistics and be empowered to improve their own work processes.

Analysis of ‘root causes’ of defects is very much a risk analysis effort [HALL98] and a handful of my clients are reporting success at doing so. But, most are still working on other disciplines like Inspection and others elsewhere in this paper.

Principle 9. Delegate personal responsibility for risk

People must be give personal responsibility in their sector for identification and mitigation of risks.

To back up communicating about risk, people must be given ownership of the risks in their sector (e.g. allocating ownership/sign off of IE tables and giving people specific roles within Inspections).

Principle 10. Contract out risk

Make vendors contractually responsible for risks, they will give you better advice and services as a result.

I would like to point out that contracting for products and services gives great opportunity to legally and financially control risks by squarely putting them on someone else's shoulders.

The effect of contracting a risk to someone else is that:

- you have gotten rid of the risk in some senses, but if they fail, you will still be affected!
- the supplier (assuming they get the risk) will be more motivated to take steps to eliminate the risks,
- be motivated to tell you exactly what you have to do to avoid being hit by risks,
- might come up with a more realistic bid and time plan to cope with the risks.

Summary

Risks can be handled in many ways and at many levels. I have tried to point out some risk management methods which are not so well known or well treated in existing literature. Hopefully, the need to fully integrate risk management into all the development and maintenance processes is clear.

Table 6 and Table 7 recap the ideas presented in this paper. Table 6 is a set of policies for risk management [See GILB98 for more detail]. Table 7 contains ‘Twelve Tough Questions’ to ask when assessing risk.

TABLE 6: Policy Ideas for Risk Management

- **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 [GILB98, 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.

TABLE 7: Twelve Tough Questions

1.	Why isn't the improvement quantified?
2.	What is degree of the risk or uncertainty and why?
3.	Are you sure? If not, why not?
4.	Where did you get that from? How can I check it out?
5.	How does your idea affect my goals, measurably?
6.	Did we forget anything critical to survival?
7.	How do you know it works that way? Did it before?
8.	Have we got a complete solution? Are all objectives satisfied?
9.	Are we planning to do the 'profitable things' first?

10. Who is responsible for failure or success?
11. How can we be sure the plan is working, during the project, early?
12. Is it ‘no cure, no pay’ in a contract? Why not?

References

DION93: Raymond Dion, "Process Improvement and the Corporate Balance Sheet", IEEE Software, July 1993, Pages 28-35.

DION95: Raymond Dion, Tom Haley, Blake Ireland and Ed Wojtaszek of Raytheon Electronic Systems, “The Raytheon Report: Raytheon Electronic Systems Experience in Software Process Improvement”, November 1995, SEI web-site, <http://www.sei.cmu.edu/products/publications/95.reports/95.tr.017.html/>. This is an important update of earlier reports.

GILB88: Tom Gilb, “Principles of Software Engineering Management”, Addison-Wesley, 1988, 442 pages. ISBN 0-201-19246-2. See particularly Chapter 6, Estimating the Risk (reproduced in Boehm, Software Risk Management, IEEE CS Press, 1989 page 53).

GILB93: Tom Gilb and Dorothy Graham, “Software Inspection”, Addison-Wesley, 1993, ISBN 0-201-63181-4, 5TH printing 1998, 471 pages. This book covers the Defect Detection Process and the Defect Prevention Process, as well as giving sample Rules to check by, defined processes and a well defined set of Glossary terms to aid quantification and comparison. It is a next-generation Inspection, with hundreds of larger and smaller improvements over initial Inspection practices.

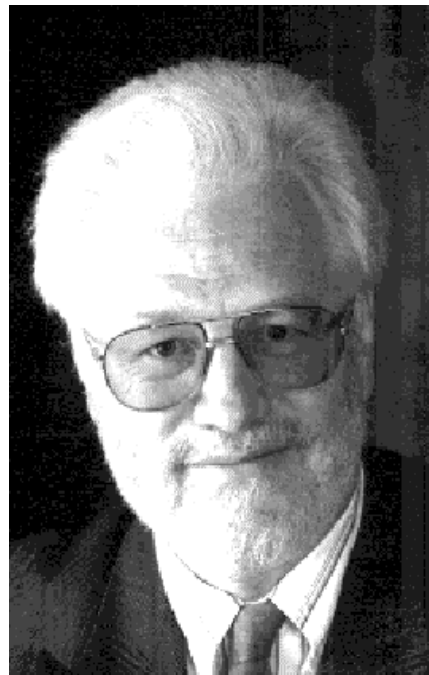
GILB98: Tom Gilb, Various papers and manuscripts on <http://www.Result-Planning.com/>. The manuscripts include:

- . ‘Requirements-Driven Management using Planguage’ (1995-6)
- . ‘Evolutionary Project Management’ (1997)
- . ‘Requirements Engineering Language’ (1998).

HALL98: Elaine M. Hall, “Managing Risk: Methods for Software Systems Development. SEI Series in Software Engineering”, Addison Wesley Longman, USA. (enq.orders@awl.co.uk) , £31.95, 1998, ISBN 0-201-25592-8, 374 pages. This book is impressive and contains a lot of useful detail and original thought. Anyone interested in risk will enjoy and learn from the book as I did. It does not however deal with most of the subjects in this paper {specification languages, impact estimation, inspections, evolutionary delivery}. This in no way detracts from the book’s favorable recommendation. It does tackle ‘quantified objectives’ much better than other texts.

MAY96: Elaine L. May and Barbara A. Zimmer, “The Evolutionary Development Model for Software”, Hewlett-Packard Journal, August 1996, Vol. 47, No. 4, pages 39-45.

*The author was at HP in 1989 on a project team who were taught early versions of the Planguage method. The article is full of practical advice and case studies gleaned from ten major projects in eight HP divisions. It must be strongly recommended to anyone interested in the practical implementation of Evo in a project and especially in an organization for many projects. See also article by Todd Cotton in same issue on Evo. HP Journal subscription free to qualified individuals: Write Distribution Manager, HP Journal, M/S 20BH, 3000 Hanover Street, Palo Alto, CA, USA-94304, or Email: **hp_journal@hp_paloalto-gen13.om.hp.com**. HP Journal is available on World Wide Web at “<http://www.hp.com/hpj/Journal.html>.” (Warning HP may have moved this site, but you can get to new site from here)*



Author Biography

Tom Gilb is the author of “Principles of Software Engineering Management” (1988) and “Software Inspection” (1993). His book “Software Metrics” (1976) coined the term and, was used as the basis for the Software Engineering Institute Capability Maturity Model Level Four [SEI CMM Level 4]. His most recent interests are development of true software engineering and systems engineering methods. His sons, Kai and Tor, now work with him.

Tom Gilb was born in Pasadena CA in 1940. He moved to England in 1956, then two years later he joined IBM in Norway. Since 1963, he has been an independent consultant and author.

This paper was edited by Lindsey Brodie, lindsey@brodie.source.co.uk

-----*End of Paper*-----

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data: An Industrial Case Study

Dr. Peter Liggesmeyer, Dr. Michael Rettelbach
Siemens AG, Munich, Germany

- Motivation
- The goals
- The approach
- Description of the case study
- Measures
- Applied statistical methods
- Results
- Conclusions

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 1

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

Motivation

- Increasing application of software and system measures requires
 - techniques to interpret the measurements in a correct way and
 - to reduce the measurement effort to a minimum
- Measuring only makes sense, if correct conclusions can be drawn
- Measuring costs more time and money than necessary, if too many
 - Manual interpretation of measures based on experience
 - is complicated (multi-parameter problems),
 - requires experts and
 - provides on information on the dependability of the

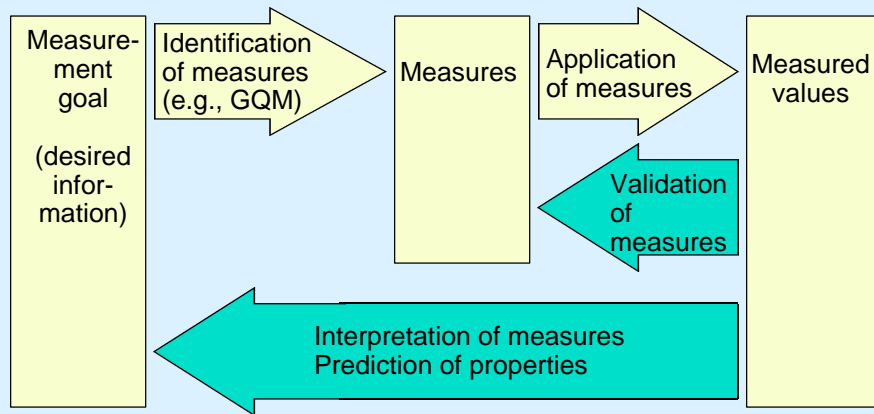
***Statistical interpretation of measures provides
a solution for these problems***

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 2

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

The goals

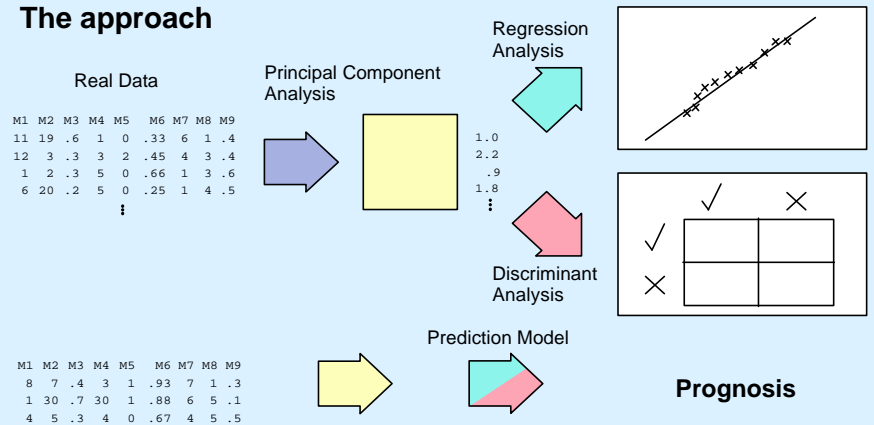


© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 3

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

The approach



© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 4

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

The approach

- Steps:
 - Definition of measurement tasks and selection of measures
 - Model construction and calibration by measurement data, evaluation of prediction quality
 - Analysis of further measurement data based on the previously selected model, estimation of the induced error
- Benefits:
 - Determination of the set of essential, i.e., most suitable,

 - Detection of errors in the initial selection step
 - Prediction of system characteristics by interpreting measurements in a systematic way

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 5

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

Description of the case study

- Determination of models for the prediction of different properties (delay, budget overdraw, ...) using measures based on routine
- Evaluation of the models, e.g., determination of the dependability
- Selection of optimal combinations of single measures, i.e., identification of unnecessary measures
- Study based on 329 projects, that
 - started and ended within a period of two years,
 - have been planned to take at least 6 months,
 - and have not been cancelled during runtime.

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 6

S

Prediction of Project Quality by applying Stochastic ...

Measures

At project start:

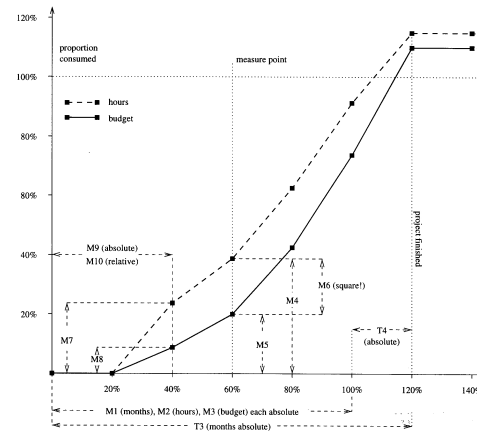
- M1: Planned length (months)
- M2: Estimated total working hours
- M3: Estimated total budget

At measure point:

- M4: Proportion of consumed working hours
- M5: Proportion of spent budget
- M6: Square of the difference of both

Additionally:

- M7: Relative height of the first account movement (working hours)
- M8: Relative height of the first account movement (budget)
- M9: Month of the first account movement
- M10: Month of the first account movement



Target values:

- T1, T2, T3: Relative consumption of working hours, budget, time
- T4: Absolute difference between planned and actual project duration

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 7

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

Applied statistical methods

- Application of *Discriminant Analysis* to predict projects that are likely to consume more than 110% of the planned budget.
- Measurement points: 50% and 60% of the planned total project
- Application of Fisher's¹ method
 - determines a vector $I = (I_1, I_2, \dots, I_{10})$ and a scalar c
 - so that $I_1 * M1 + I_2 * M2 + \dots + I_{10} * M10$ optimally distinguishes between the two populations (here: bad . good projects), if the result it compared to c .
- Mathematically founded optimal criterion for interpreting measured
- Calculation of prediction quality

1 Johnson, R.A., Wichern, D.W.: Applied Multivariate Statistical Analysis, Prentice Hall, 1982

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 8

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

Results

- Prediction quality:
 - Apparent error rate (AER: Number of misclassifications compared to the number of classifications): 24.6% (50% measure point), 23.7% (60% measure point)
=> ~75% of the predictions are correct
 - It is possible to take into account different costs of misclassifications (predicting a bad project as good predicting a good project as bad)
- Identification of the most significant measures
- Half the measures are not necessary for the prediction => reduction of measurement costs without losing quality

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 9

S

Prediction of Project Quality by applying Techniques to Metrics based on Accounting Data

Conclusions

- We are fully satisfied with the hit proportion of 75%
- We are convinced it can be improved by additional measures
- Application of stochastical techniques to measures ...
 - ... permits early and dependable problem identification
 - ... provides a decision scheme that generates results with a known quality and can easily be applied by non-experts
 - ... saves measurement effort by identifying unnecessary

 - ... identifies the most important measures as a basis for

 - ... provides the basis for optimizing the measures
 - ... saves money by avoiding wrong measurement interpretations

© Dr. Liggesmeyer, ZT PP 2, Siemens AG, 10

Prediction of Project Quality by applying Stochastical Techniques to Metrics based on Accounting Data: An Industrial Case Study

Michael Rettelbach, Peter Liggesmeyer

Siemens AG, Corporate Technology ZT PP 2S, Munich, Germany

E-mail: {michael.rettelbach,peter.liggesmeyer}@mchp.siemens.de

August 27, 1998

Abstract

We demonstrate that routine observations of project accounts (working hours, budget, planned project length, ...) contain enough statistically usable information for the prediction of the outcome of projects (delay, budget overdraw, ...). Our case study includes more than 300 projects that started and ended within a fixed period of two years, have been planned to take at least 6 months and have not been cancelled during runtime.

Keywords

statistical metrics; discriminant analysis; Fisher's classification; industrial case study.

1 Introduction

The primary aim of this study was to statistically analyze accounting data recorded during the runtime of industrial projects. In Section 2 we will state which data sets are suitable for investigation and which metrics can be deduced. The statistical analysis based on these metrics will be described in Section 3. The analysis includes the construction of prediction models for different conclusions, the evaluation of such models, and the selection of optimal (linear) combinations of single metrics. In Section 4 we present computational results and conclude this paper with a detailed interpretation of these results (Section 5).

2 Data and metrics

2.1 Description of the data set

Our investigations include 329 projects that started and ended within a fixed period of two years, have been planned to take at least 6 months, and have not been cancelled during runtime. Usual financial data was recorded monthly, only including planned project length, and two accounts for working hours and budget, respectively.

2.2 Metrics

For each single project we recorded the following metrics: planned length in months (M1), estimated total number of working hours (M2), and estimated total budget (M3). These three are known at the beginning of the projects. After a certain proportion of the total project time called measure point, we recorded the following: proportion of consumed working hours (M4), proportion of spent budget (M5), and the square of the difference of both to consider if both are drifting apart (M6).

We additionally investigated the suspicion that a late start of a project in combination with a disproportionately high start of accounting will influence the project's outcome. Therefore we measured the relative height of the first account movement of both, working hours and budget (M7 and M8), the month of the first account movement (M9), and the proportion of the planned total project time until that moment (M10).

Figure 1 shows an example of a way a project might develop, and the recorded metrics.

2.3 Target values

We investigated a bundle of target values at the end of the projects, including the relative consumption of working hours, budget, and project time (T1, T2, T3), as well as the (absolute) difference between planned and actual project time (T4).

3 Applied statistical methods

Our aim was to establish a model by applying *discriminant analysis* methods to the metrics. The focus of our investigation was predicting projects that are likely to overdraw their budget.

3.1 Fisher's discriminant function

We decided to use Fisher's method [JW82], because of the following advantages:

- Although initially designed for multivariate normal measures, it is also applicable if this precondition is not fulfilled, performing generally well (see [FHT96]).

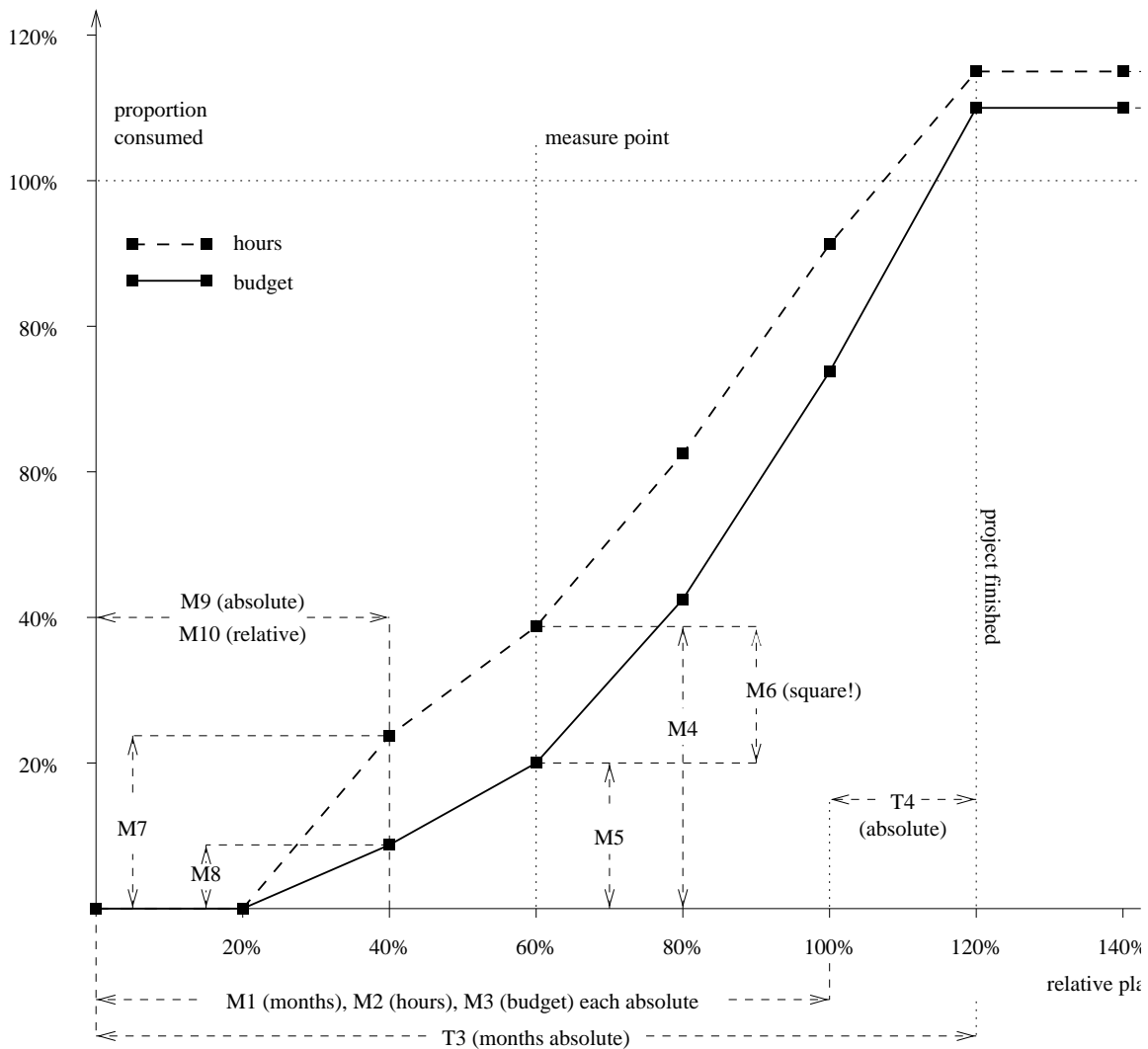


Figure 1: Example of a project.

An investigation of our random variables reveals that none is normally distributed, not even approximately. Additionally the transformation functions that are commonly used to produce normal distributions fail for almost each of the variables (see e.g. [JW82])

- Fisher's model is based on a linear combination of the variables. Thus, it is simple to implement and performs fast making a prediction. Additionally the whole database can be omitted once the coefficients of the linear combination have been computed. This linear combination can be implemented in generally available spreadsheet applications like MS EXCEL.

The fast decision is a clear advantage compared with other procedures, because we will have to apply it very often for evaluating the quality of the prediction model. We

will discuss this aspect in Section 3.2.

- Other methods like *loglinear models* or *smooth density estimators* failed to produce useful discrimination models [FHT96].

The result of Fisher’s procedure is a vector $l = (l_1, \dots, l_{10})$ and a scalar c such that the linear combination $l_1 \cdot \mathbf{M1} + \dots + l_{10} \cdot \mathbf{M10}$ optimally distinguishes between the two populations. c is the value that is used to classify an observation as belonging to the first or to the second population.

Hereby we obtain a mathematically founded optimal decision criterion that substitutes the usual manual procedure based on pure intuition. It is a rule how to combine metrics with respect to the decision criterion for classifying objects (projects, ...) as ’bad’ or ’good’.

3.2 Evaluation of the classification

To evaluate the discrimination quality of the classification we investigated two distinct aspects: the *apparent error rate* (AER), which denotes the total number of misclassifications compared to the total number of classifications, and the *expected cost of misclassification* (ECM), which additionally takes into account the cost of the different kinds of errors [JW82]. We assume that our n observations are divided into two populations of size n_1 and n_2 , respectively. Furthermore, we use the following scheme to evaluate the quality of predictions.

		prediction		
		bad	good	
observation	bad	$\mathbf{n_{11}}$	n_{12}	$n_{11} + n_{12} = n_1$
	good	n_{21}	$\mathbf{n_{22}}$	$n_{21} + n_{22} = n_2$

This means that n_{11} and n_{22} observations have been correctly recognized to belong to population 1 and 2, respectively. n_{12} observations have been classified as members of population 2 although they belong to population 1. The meaning n_{21} is corresponding.

The AER-value is the proportion of misclassifications. It is given by

$$\mathbf{AER} = \frac{(n_{12} + n_{21})}{n}; \quad n = n_1 + n_2.$$

To compute ECM we additionally need to know the costs of misclassification for both kinds of errors: classifying a type 1 observation as being type 2 (c_{12}) and vice versa (c_{21}). This is important, because the costs may be significantly different. Thus we get

$$\mathbf{ECM} = \frac{n_{12}c_{12} + n_{21}c_{21}}{n}.$$

To compute both values AER and ECM we could compute a classification model from all observations and classify all observations afterwards, counting correct and wrong ones. This approach usually underestimates both values, because the data set for fitting the model (fit

data set) and the data set for testing (test data set) are the same. We will refer to this approach as *simple evaluation* in the rest of the paper.

Another way would be to randomly separate all observations into fit and test data set. The usual ratio using this approach is about 2:1. The classification model is computed only using the fit data set, and evaluated with the test data set. The drawback of this approach is that the final classification model does not include all information provided by the observations, especially if there is only a small number of observations.

Although we have a large number of observations (> 300), we decided to select a third approach, the *holdout procedure*. The classification model is computed n times, for each of these exactly one observation is not used (holdout). With each model exactly one observation is classified, namely the holdout, counting correct and incorrect ones.

3.3 Selection of variables

Usually everyone will agree that it is foolish to measure anything that is measurable. This is especially true for Fisher's classification. We will be able to observe that selectively leaving out metrics will significantly increase the quality of the prediction. Thus, the question is which metrics should be omitted.

There are direct decision algorithms that compute an optimal subset of the vector of metrics. Unfortunately, these algorithms usually include the assumption that the metrics are normally distributed [FHT96]. As already stated, this is not true in our context. Therefore we decided to perform an exhaustive search for the optimum, which includes $2^{10} - 1 = 1023$ combinations. Computing the holdout AER and ECM for all these would have taken about some 20 hours on a workstation for each prediction model. This is caused by the fact that for each variable combination 329 prediction models are to be computed. In total this requires 336,000 calculations of prediction models.

To reduce the effort, we applied the following approach: At first we use the simple evaluation method to obtain a lower bound of the actual values for each subset of metrics. This procedure also includes 336,000 classifications, but only demands 1023 computations of the classification model. After that step we use the holdout classification starting with the simple evaluation winners. We stop this procedure when we obtain a holdout evaluation better than all remaining simple evaluations. By this way we definitely find the global maximum (or maxima) and usually reduce the computational effort to a few minutes.

We will observe that the optimization of AER and ECM will result in different optimal subsets of the metrics.

4 Numerical results

We defined two different measure points to investigate whether later measurement causes better prediction quality. We used 50% and 60% of the planned total project time as measurement points. One might argue that these points are too late to positively influence the outcome of a project. We agree with this objection, but hold the fact against it that

the data lacks of quality because it is not explicitly measured for statistical purposes. Later measurement points promise to bear measures containing more information about a project's outcome. Our initial aim to show that prediction is possible in general with this kind of data could be achieved this way. *Thus, we obtained good arguments to establish better recording of metrics, also in earlier project phases.*

Optimizing ECM we only need to know the ratio of the misclassification costs, c_{12}/c_{21} . We use the values 2 and 3 as examples for this ratio, i.e., a not recognized 'bad' project causes twice (three times) the costs as an erroneously as 'bad' classified 'good' one. We denote this as ECM^2 and ECM^3 .

Example: Assume that we want to predict that a project will use more than 110% of its planned budget. We split up our whole population along this borderline and found 78 projects above and 251 below it. We computed the following prediction model for all variables based on the 50% measurement points:

	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
	-0.19463	0.00040	-0.00266	0.01390	0.02651	-0.00061	-0.04871	0.02031	0.09850	-0.00465
c_{AER}	=	-0.96212		AER	=	28.88%				
c_{ECM}^2	=	-0.48653		ECM^2	=	0.41337				
c_{ECM}^3	=	-0.89199		ECM^3	=	0.56535				

This table is to be read as follows. Provided you want to optimize AER, assume for a new observation that the expenses exceed 110% of the planned budget, iff

$$-0.19463 \cdot M1 + 0.00040 \cdot M2 + \dots - 0.00465 \cdot M10 \leq -0.96212.$$

Based on the 60% measures we computed the following prediction model:

	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
	-0.19548	0.00040	-0.00238	-0.00737	0.03932	0.00019	-0.01247	-0.00805	0.15032	-0.00845
c_{AER}	=	-0.53710		AER	=	25.23%				
c_{ECM}^2	=	-0.06150		ECM^2	=	0.39210				
c_{ECM}^3	=	-0.46697		ECM^3	=	0.51064				

Afterwards we applied the procedure for the optimization of the metrics vector as described in Section 3.3. At first we wanted to optimize the AER value. For the 50% measures we found one optimal variable subset:

$$AER_{optimized} = 24.62\% \qquad 50\%$$

	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	c_{AER}
	-0.19414				0.04014		-0.02791		0.08592		-0.79943

For the 60% measures there have been eight of them.

AER *optimized* = 23.71%

60%

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	c_{AER}
-0.20360				0.03211			-0.01946	0.15873	-0.00882	-0.56303
-0.19969			-0.01141	0.04348			-0.01970	0.15472	-0.00866	-0.54047
-0.15714			-0.01339	0.04171	0.00013	-0.01579	-0.00230			-0.42866
-0.19997		-0.00048		0.03196		-0.02017		0.14927	-0.00880	-0.64202
-0.20098		-0.00049		0.03213		-0.01054	-0.00989	0.15236	-0.00909	-0.64694
-0.20316	-0.00002			0.03206			-0.01951	0.15794	-0.00895	-0.58133
-0.19906	-0.00003		-0.01156	0.04356			-0.01976	0.15364	-0.00882	-0.56425
-0.19612		-0.00047	-0.01120	0.04312		-0.02039		0.14529	-0.00863	-0.61916

Assuming $c_{12}/c_{21} = 2$, we found

ECM² *optimized* = 0.37690

50%

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	c_{ECM}^2
		-0.00097	0.03185		-0.00084	-0.05166	0.03251		0.00864	0.79530
			0.03190		-0.00081	-0.05171	0.03354		0.01107	0.99771
			0.03289		-0.00081	-0.05260	0.03355	-0.10521	0.02590	1.08058
	-0.00009		0.03274		-0.00084	-0.05318	0.03376	-0.10608	0.02469	0.97279

ECM² *optimized* = 0.35866

60%

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	c_{ECM}^2
	0.00044	-0.00291	-0.01582	0.04123		-0.01380			0.00234	1.03508
	0.00044	-0.00291	-0.01561	0.04125		-0.01381			0.02581	1.04862
	0.00044	-0.00291	-0.01560	0.04128		-0.01383		0.02340	0.00047	1.05808
	0.00044	-0.00299	-0.01478	0.03941	0.00017	-0.01357				0.94160
	-0.00008		-0.01922	0.04471		-0.01351		0.02598		1.05964
	-0.00007		-0.01921	0.04474		-0.01353		0.02314	0.00055	1.07077
			-0.01934	0.04398		-0.01287				1.01616
			-0.01852	0.04298	0.00012	-0.01286				1.01884

And for $c_{12}/c_{21} = 3$

ECM³ *optimized* = 0.50456

50%

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	c_{ECM}^3
-0.16714				0.03560		-0.02528				-0.86324
-0.16629				0.03547		-0.05242	0.02695			-0.86330

ECM³ *optimized* = 0.47720

60%

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	c_{ECM}^3
-0.15921			0.02563			-0.01423				-0.51516

In our initial study we additionally investigated budget overdraws (more than 100%), overdraws of the accounted hours, and late project completions (≥ 2 months late and 17.6% of total project time late) in the same detail. The achieved values for AER were located in the range 1/3 to 1/4. The quality of ECM was comparable.

5 Interpretation

General observations concerning the prediction quality

Although our data set is only fit for limited service for statistical investigations we could achieve to correctly classify 2/3 to 3/4 observations. We are very content with these values. On the other side we could not reach AER values significantly better than 25%. We suggest two reasons for this fact. The first is that there are some outliers (single observations that lie far away from most of the observations) in the data set [Wad89]. We could selectively omit those values to improve the prediction quality. Preselection of observations in general will give a good chance to improve the prediction quality. Excessive use of this approach, however, will generate a set of observations that only includes few aspects of our initial one, resulting in a bad prediction quality for new observations. Second, we strongly suspect that our metrics explain some, but not all of the reasons for lateness or budget overdraws. Therefore we recommend to define new metrics that could contribute to a better understanding of these reasons.

The measure points

Except of a few of our models we observed that concerning the optimal variable subsets we always achieve a better prediction with the 60% measure point. Although this hypothesis is not statistically proven we can recognize a clear trend: later measurement improves the prediction quality. This is also confirmed by the statistical test below. This discovery might appear to be trivial to an observer, but it provides us another confirmation that our metrics contain the appropriate information.

Significance of individual metrics

As explained earlier the procedures to directly obtain an optimal subset of variables demand a multivariate normal distribution. The same is true for the following test. But, as stated in [FHT96], it is usually also applied to other data to obtain a trend.

For Fisher's discriminant function different mean vectors of the two populations are essential. We perform a statistical test (F-test) with significance level, α , and the hypothesis, H_0 : "the mean values of the metrics are equal for both populations" against the alternative, H_1 : "the mean values are different", using the Mahalanobis distance [FHT96]. The following table shows the α -values that are not to be exceeded to reject H_0 . 1.0 is the best obtainable value and means that the mean values of this variable for both population differ almost for sure. For better readability we underlined values above 99% and set those below 95% in

italics. We also included the same test of our other investigations with the data set: *time 117* means a more than 17% overdraw of the planned project time, *time 2m.* an overdraw of at least 2 months. *hours 110* is the investigation concerning an overdraw of the planned hours for at least 10%, *account 110* means the study in this paper and finally *account 100* is the same study except that the populations are divided along the 100% budget line.

	time 117		time 2m.		hours 110		account 110		account 100	
	50	60	50	60	50	60	50	60	50	60
M1	<u>1.0000</u>	<u>1.0000</u>	<u>0.9999</u>	<u>0.9999</u>	<i>0.7849</i>	<i>0.7849</i>	0.9716	0.9716	0.9502	0.9502
M2	<i>0.9321</i>	<i>0.9321</i>	0.9874	0.9874	<i>0.5407</i>	<i>0.5407</i>	<i>0.5032</i>	<i>0.5032</i>	<i>0.1943</i>	<i>0.1943</i>
M3	<i>0.9018</i>	<i>0.9018</i>	0.9827	0.9827	<i>0.6547</i>	<i>0.6547</i>	<i>0.7892</i>	<i>0.7892</i>	<i>0.2689</i>	<i>0.2689</i>
M4	<u>0.9915</u>	<u>0.9998</u>	0.9896	<u>0.9999</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>
M5	0.9812	<u>0.9995</u>	0.9794	<u>0.9999</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>	<u>1.0000</u>
M6	<i>0.9291</i>	<i>0.9276</i>	0.9507	0.9654	<i>0.7557</i>	<u>0.9917</u>	<i>0.2955</i>	<i>0.9354</i>	<i>0.1410</i>	<i>0.7923</i>
M7	<i>0.3280</i>	<i>0.8032</i>	<i>0.6974</i>	0.9712	0.9782	<u>0.9930</u>	0.9899	<u>0.9938</u>	<i>0.9376</i>	<i>0.9194</i>
M8	<i>0.1933</i>	<i>0.8230</i>	<i>0.6012</i>	0.9746	0.9876	<u>0.9926</u>	<u>0.9936</u>	<u>0.9936</u>	<i>0.9333</i>	<i>0.9157</i>
M9	<i>0.0868</i>	<i>0.0868</i>	<i>0.7823</i>	<i>0.7823</i>	0.9590	0.9590	0.9631	0.9631	0.9668	0.9668
M10	0.9865	0.9872	<i>0.9310</i>	0.9652	0.9718	0.9739	0.9502	0.9549	<i>0.9241</i>	<i>0.9042</i>

First, this table shows that (except of M6) a high significance of the 50% measures comes along with a high significance of the 60% ones. Comparing the values in the table with the number of appearances of each variable in the optimal subsets we can observe a coincidence at least for the prediction we performed in Section 4.

Finally we discuss the suspicion (cf. Section 2) that the delay and relative height of the first movement of the accounts is an indicator of a project's outcome. The optimal subsets reveal a high significance of this value: each subset includes at least one of both, M7 and M8. Removing both metrics from our measurement vector we obtain evidently worse values for AER and ECM for new optimal subsets (original values in brackets): AER_{50} 29.18% (24.62%), AER_{60} 26.75% (23.71%), ECM_{50}^2 0.38602 (0.37690), ECM_{60}^2 0.36474 (0.35866), ECM_{50}^3 0.54711 (0.50456), ECM_{60}^3 0.48632 (0.47720).

6 Summary and future work

By this study we show that routine observations of project accounts contain enough statistically usable information for the prediction of the outcome of projects. We are fully satisfied with the hit proportion of 75% and we are convinced this can be improved by additional, well aimed measurements. We are encouraged to continue the research on this field.

With Fisher's method we provide a decision scheme that can even be used by a non experienced person (without the use of specialized software) to decide if a new project tends to overdraw its budget (becomes late, etc.) or not. Additionally this decision is statistically substantiated and we can anticipate the involved error. The decision can be optimized taking into account the cost of misclassification. Previously, all this had to be done by an expert, based on the knowledge he gathered over years. With the statistical methods we formalized this experience within a simple function.

Our further studies will concentrate on the inclusion of new metrics as well as tests for suitability of other discriminant analysis methods.

References

- [FHT96] Ludwig Fahrmeir, Alfred Hamerle, and Gerhard Tutz. *Multivariate statistische Verfahren*. de Gruyter, Berlin, New York, 2nd edition, 1996.
- [JW82] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, Inc., 1982.
- [Wad89] Harrison M. Wadsworth. *Handbook of Statistical Methods for Engineers and Scientists*. McGraw-Hill, 1989.

Slide 1

Year 2000 Compliance Testing

John Corden
Chair, CSSA Y2000 Vendor Group
Millennium Director, CYRANO

November 1998

CYRANO

Slide 2

Why test?

CYRANO

Excuses (for not testing)

- My supplier has guaranteed that the system is Year 2000 compliant
- The system was developed, from the outset, to use 8 digit dates
- Insufficient time
- Too difficult
- Can't be bothered

CYRANO

Some common misconceptions

- A system which stores dates using 8 digits is Year 2000 compliant
- A system which stores dates as 6 digits isn't Year 2000 compliant
- You can tell whether a system is Year 2000 compliant by examining the source code
- I can sit back and do nothing and, if it breaks, I can sue my supplier for damages

CYRANO

So **TEST** to:

- Find out if a system is Year 2000 compliant
- Mitigate your losses
- Prove that you've exercised due care

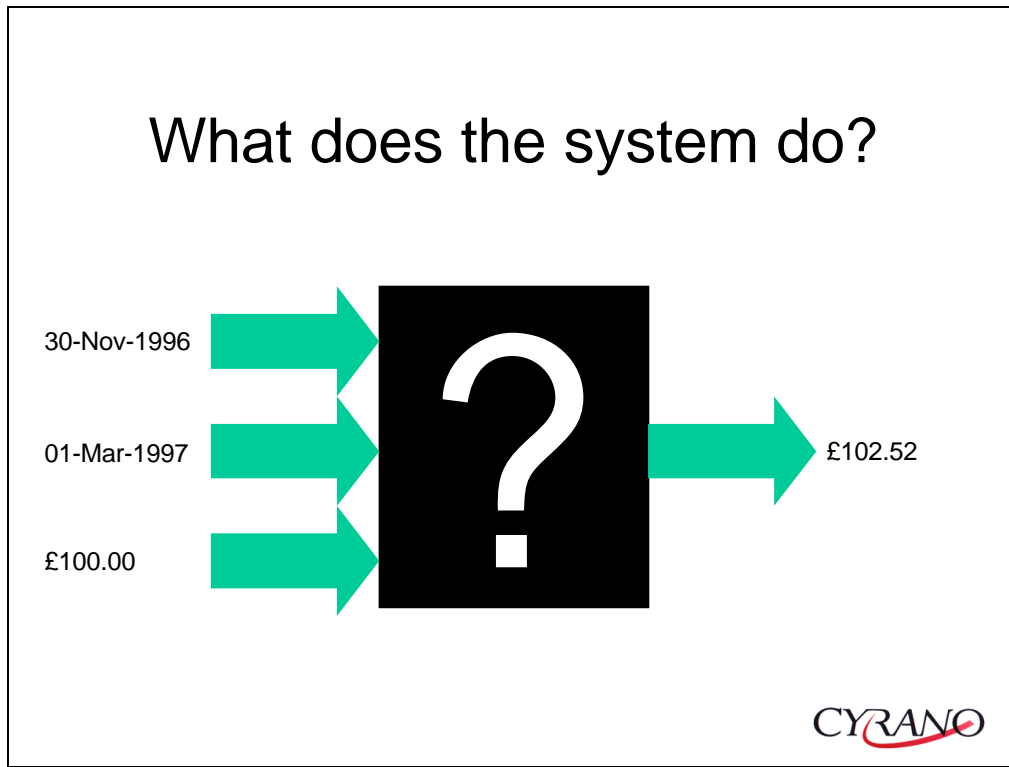
CYRANO

What do we test?

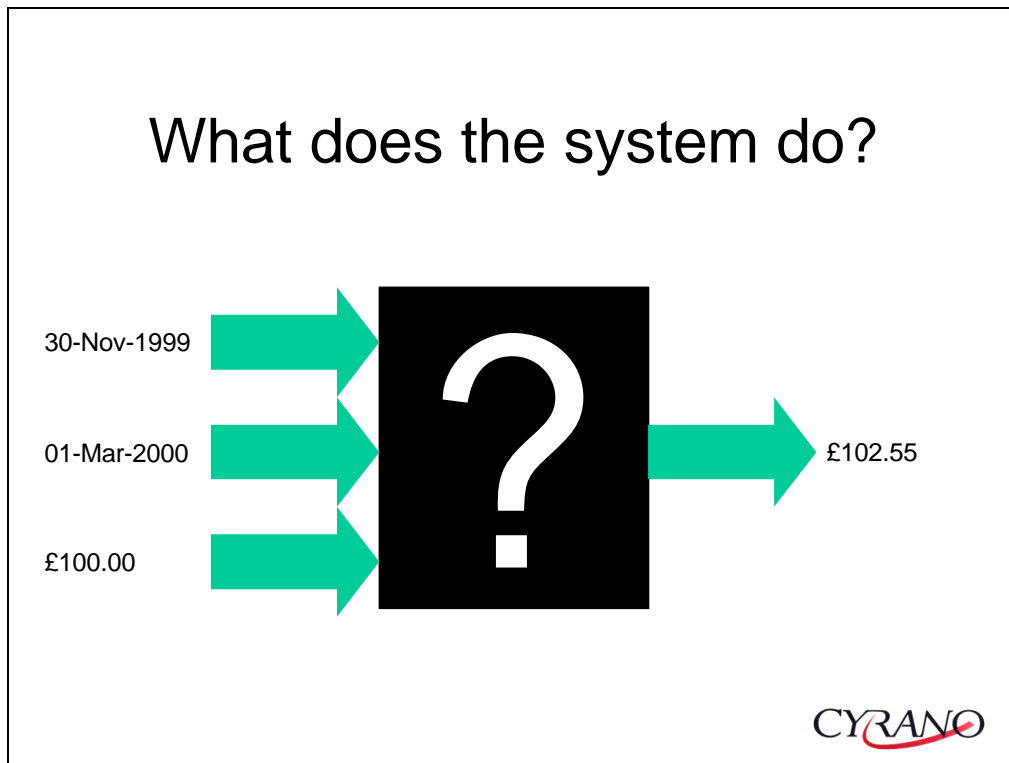
How do we test?

CYRANO

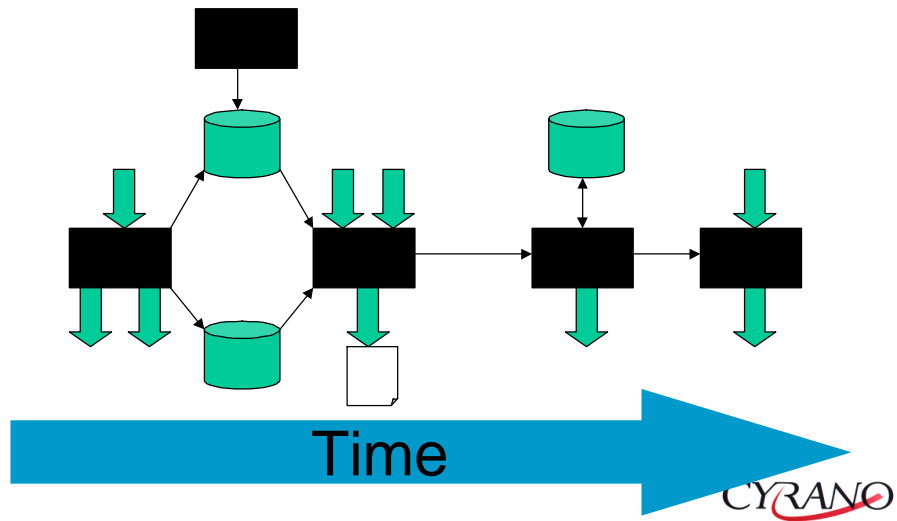
Slide 7



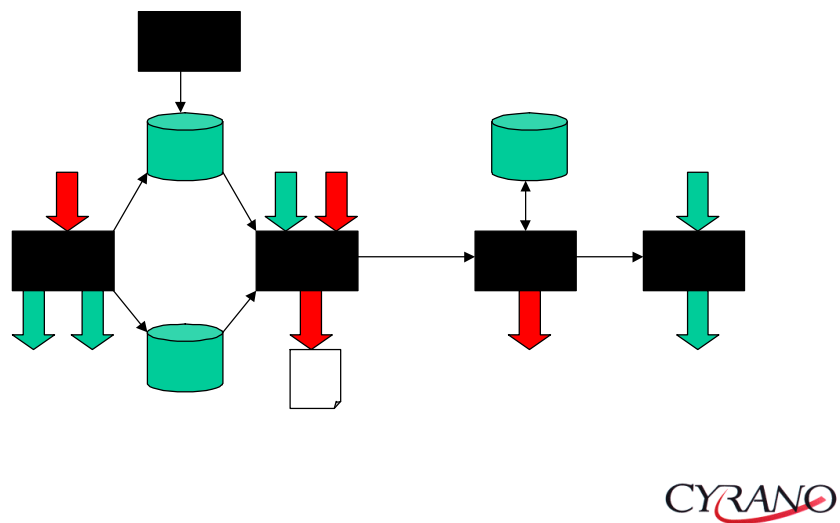
Slide 8



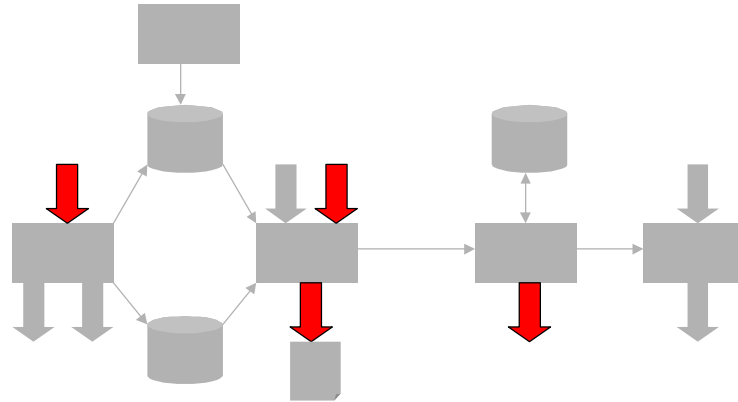
What does the system do?



Identify the date sensitive I/O

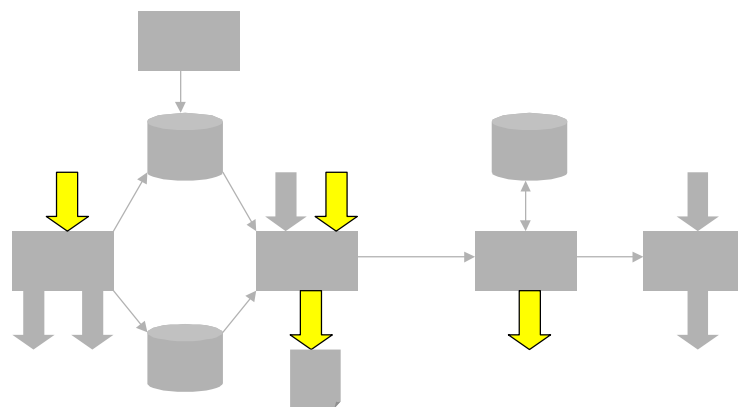


And separate it from the work-flow

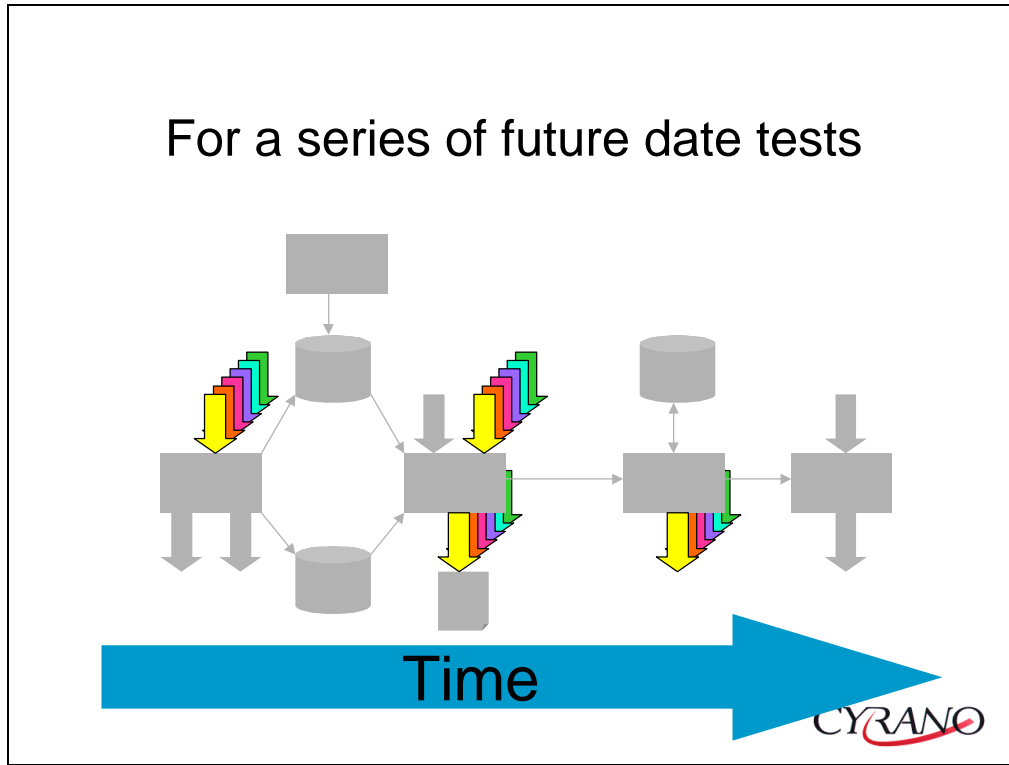


CYRANO

We can then substitute new inputs and expected outputs



CYRANO



- Key dates?
- 9/9/1999
 - 31/12/1999
 - 1/1/2000
 - 28/2/2000
 - 29/2/2000
 - 1/3/2000
 - 31/12/2000
 - 1/1/2001
 - etc
- CYRANO

Key dates?

- 9/9/1999
- 31/12/1999
- 1/1/2000
- 28/2/2000
- 29/2/2000
- 1/3/2000
- 31/12/2000
- 1/1/2001
- etc
- 4/11/1997
- last Thursday
- today
- tomorrow
- next month
- 2/1/2000
- 15/4/2000
- 1/1/2001
- etc

CYRANO

A Year 2000 Project.....



CYRANO

Jan. 24, 1998

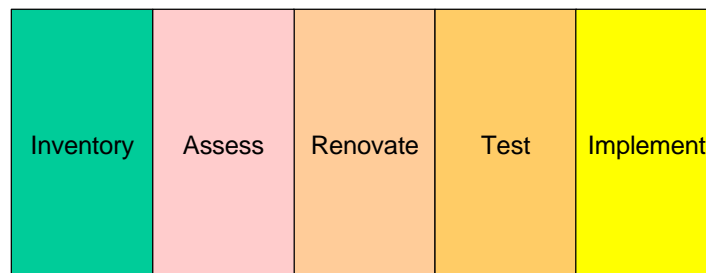
IRS describes how Year 2000 bug bungled tax instalment agreements

N-J wire services

WASHINGTON — The IRS uncovered an unintended side effect of its effort to eliminate the Year 2000 computer bug: About 1,000 taxpayers who were current in their tax instalment agreements were suddenly declared in default due to a programming error.

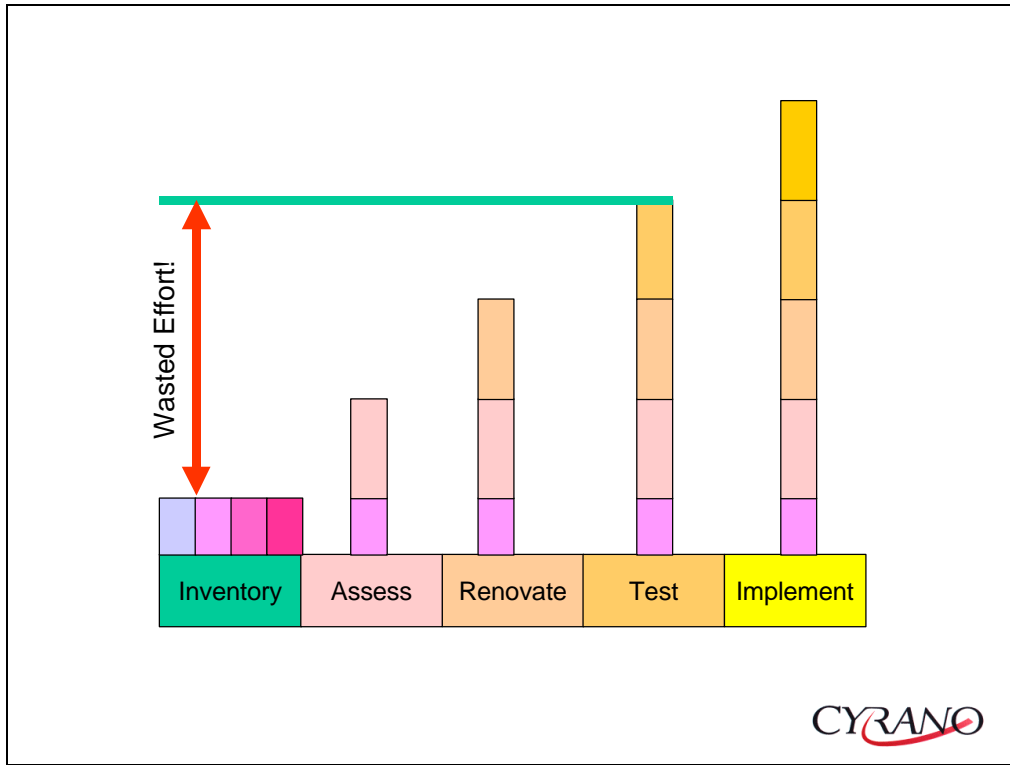


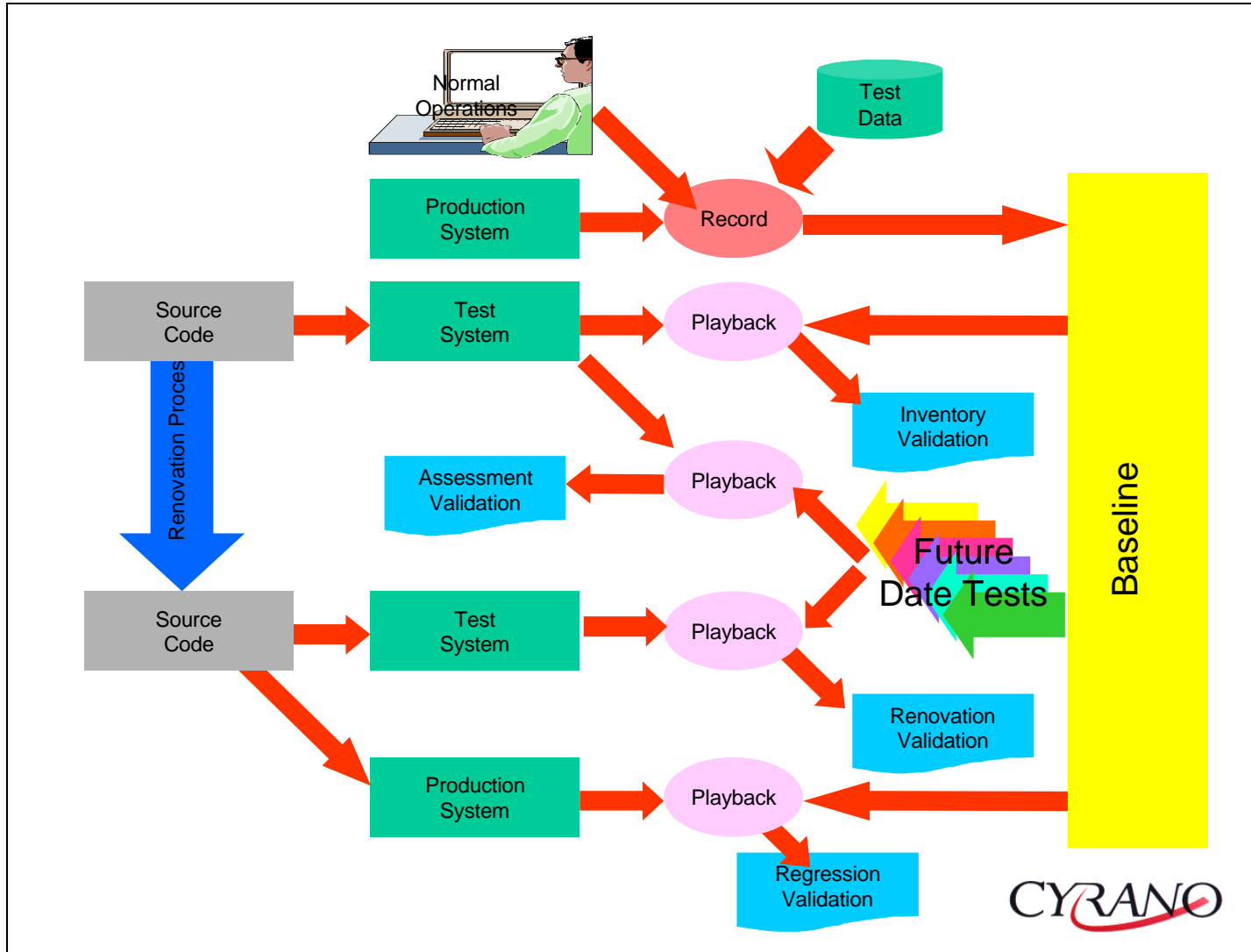
Conventional Y2000 Project Plan



What's wrong with this approach?







Slide 21

Conditions
Test Data
Expected Results
Date Simulation

CYRANO

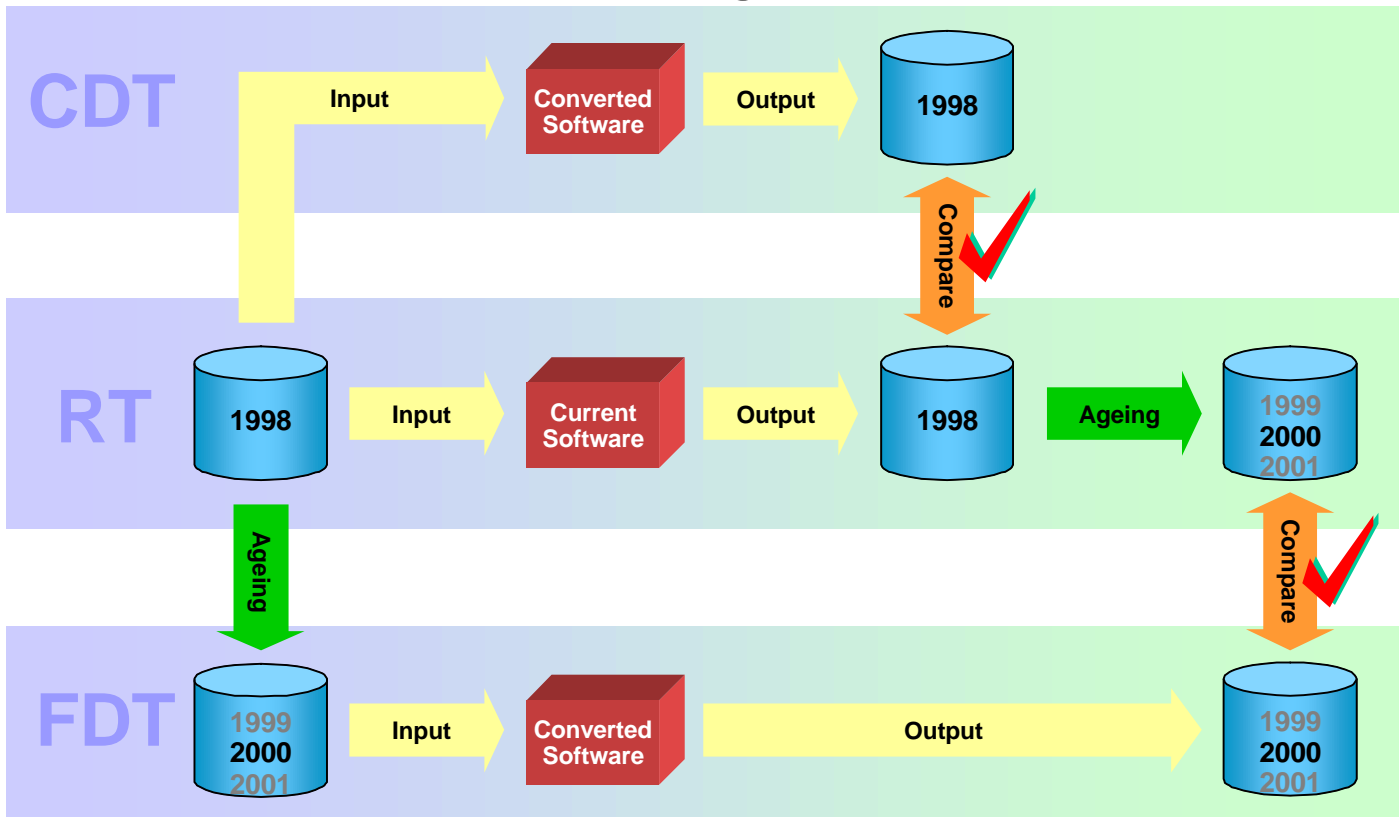
Slide 22

What about the data?

CYRANO

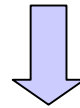
Why age test data?

Year 2000 testing: General idea



Example: COBOL program today

System date 21.08.98



DataBase with a birthday



```
COBOL Program
085100***  PEOPLE OLDER THAN 18 YEARS WILL
          PROCEED
085110***  DETERMINE AGE AND ASK FOR 18
085120***  TODAY-DATE IS STORED IN PIC X(6) AS
          YYMMDD
085130***  BIRTHDAY IS STORED IN PIC X(6) AS
          YYMMDD
085140
085150      MOVE TODAY-DATE      TO      TEMP
085160      SUBTRACT BIRTHDAY    FROM    TEMP
085170
085180      IF TEMP      >=      180000
085190          GO TO      FUNCTION-X
085200      ELSE
085210          GO TO      FUNCTION-Y
085220      END-IF
```

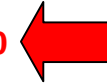


17

Age is **17**
(younger than 18!)

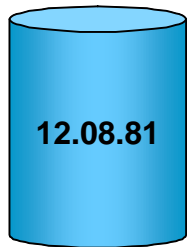
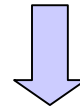
Test COBOL program for Y2000

System date 01.01.00



System date 21.08.98

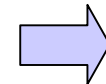
manually change



DataBase with a birthday



```
COBOL Program
085100***  PEOPLE OLDER THAN 18 YEARS WILL
          PROCEED
085110***  DETERMINE AGE AND ASK FOR 18
085120***  TODAY-DATE IS STORED IN PIC X(6) AS
          YYMMDD
085130***  BIRTHDAY IS STORED IN PIC X(6) AS
          YYMMDD
085140
085150    MOVE TODAY-DATE      TO      TEMP
085160    SUBTRACT BIRTHDAY    FROM    TEMP
085170
085180    IF TEMP      >=    180000
085190        GO TO    FUNCTION-X
085200    ELSE
085210        GO TO    FUNCTION-Y
085220    END-IF
```



- 81

Age is - 81
(younger than 18!)

Change sources (e.g. Windowing)

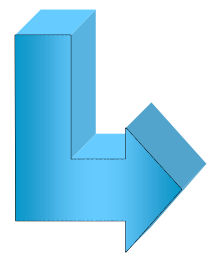
```

085100***  PEOPLE OLDER THAN 18 YEARS WILL PROCEED
085110***  DETERMINE AGE AND ASK FOR 18
085120***  TODAY-DATE IS STORED IN PIC X(6) AS YYMMDD
085130***  BIRTHDAY IS STORED IN PIC X(6) AS YYMMDD
085140
085150      MOVE TODAY-DATE      TO      TEMP
085160      SUBTRACT BIRTHDAY    FROM    TEMP
085170
085180      IF TEMP    >=    180000
085190          GO TO    FUNCTION-X
085200      ELSE
085210          GO TO    FUNCTION-Y
085220      END-IF
    
```

```

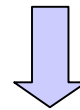
085100***  PEOPLE OLDER THAN 18 YEARS WILL PROCEED
085110***  DETERMINE AGE AND ASK FOR 18
085120***  TODAY-DATE IS STORED IN PIC X(6) AS
            YYMMDD
085130***  BIRTHDAY IS STORED IN PIC X(6) AS YYMMDD
085131***  VARIABLE WINDOW IS DEFINED AS 20
            (=200000)
085140
085150      MOVE  TODAY-DATE      TO      TEMP
085151      IF TEMP    <=    WINDOW
085152          ADD   20000000    TO      TEMP
085153      ELSE
085154          ADD   19000000    TO      TEMP
085155      END-IF
085156      MOVE  BIRTHDAY      TO      TEMP-B
085160***  SUBTRACT  BIRTHDAY    FROM    TEMP
085161      IF TEMP-B    <=    WINDOW
085162          ADD   20000000    TO      TEMP-B
085163      ELSE
085164          ADD   19000000    TO      TEMP-B
085165      END-IF
085166***  FORMER LINE 85160
085167      SUBTRACT  TEMP-B      FROM    TEMP
085170
085180      IF TEMP    >=    180000
085190          GO TO    FUNCTION-X
085200      ELSE
085210          GO TO    FUNCTION-Y
085220      END-IF
    
```

Insert a window „20“



Test COBOL program for Y2000 (1)

System date 01.01.00 ← System date 21.08.98

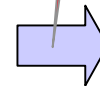
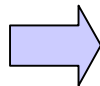


Converted COBOL Program

```
085100*** PEOPLE OLDER THAN 18 YEARS WILL PROCEED
085110*** DETERMINE AGE AND ASK FOR 18
085120*** TODAY-DATE IS STORED IN PIC X(6) AS YMMDD
085130*** BIRTHDAY IS STORED IN PIC X(6) AS YMMDD
085131*** VARIABLE WINDOW IS DEFINED AS 20 (-200000)
085140
085150         MOVE TODAY-DATE      TO TEMP
085151         IF TEMP <= WINDOW
085152             ADD 20000000 TO TEMP
085153         ELSE
085154             ADD 19000000 TO TEMP
085155         END-IF
085156         MOVE BIRTHDAY      TO TEMP-B
085160*** SUBTRACT BIRTHDAY FROM TEMP
085161         IF TEMP-B <= WINDOW
085162             ADD 20000000 TO TEMP-B
085163         ELSE
085164             ADD 19000000 TO TEMP-B
085165         END-IF
085166*** FORMER LINE 85160
085167         SUBTRACT TEMP-B FROM TEMP
085170
085180         IF TEMP >= 180000
085190             GO TO FUNCTION-X
085200         ELSE
085210             GO TO FUNCTION-Y
085220         END-IF
```



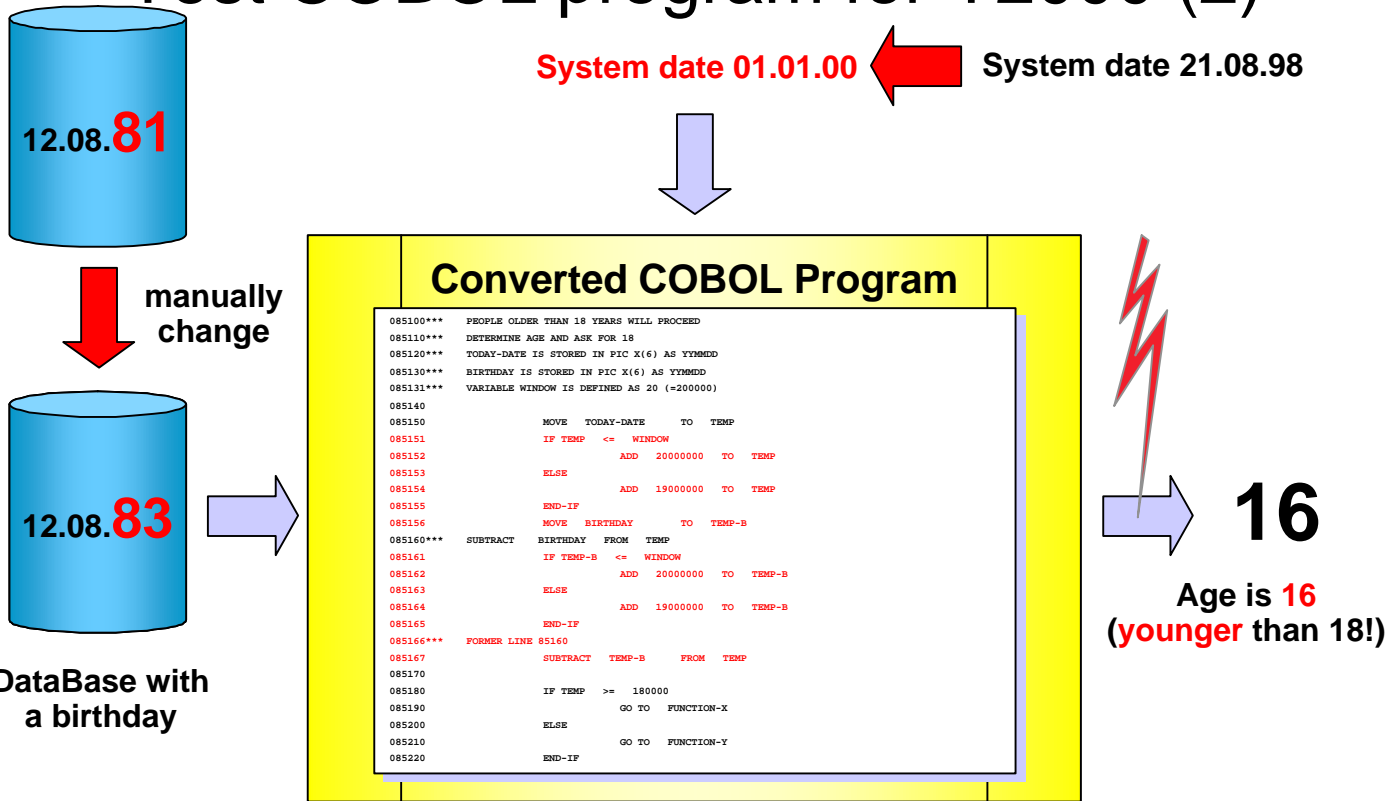
DataBase with a birthday



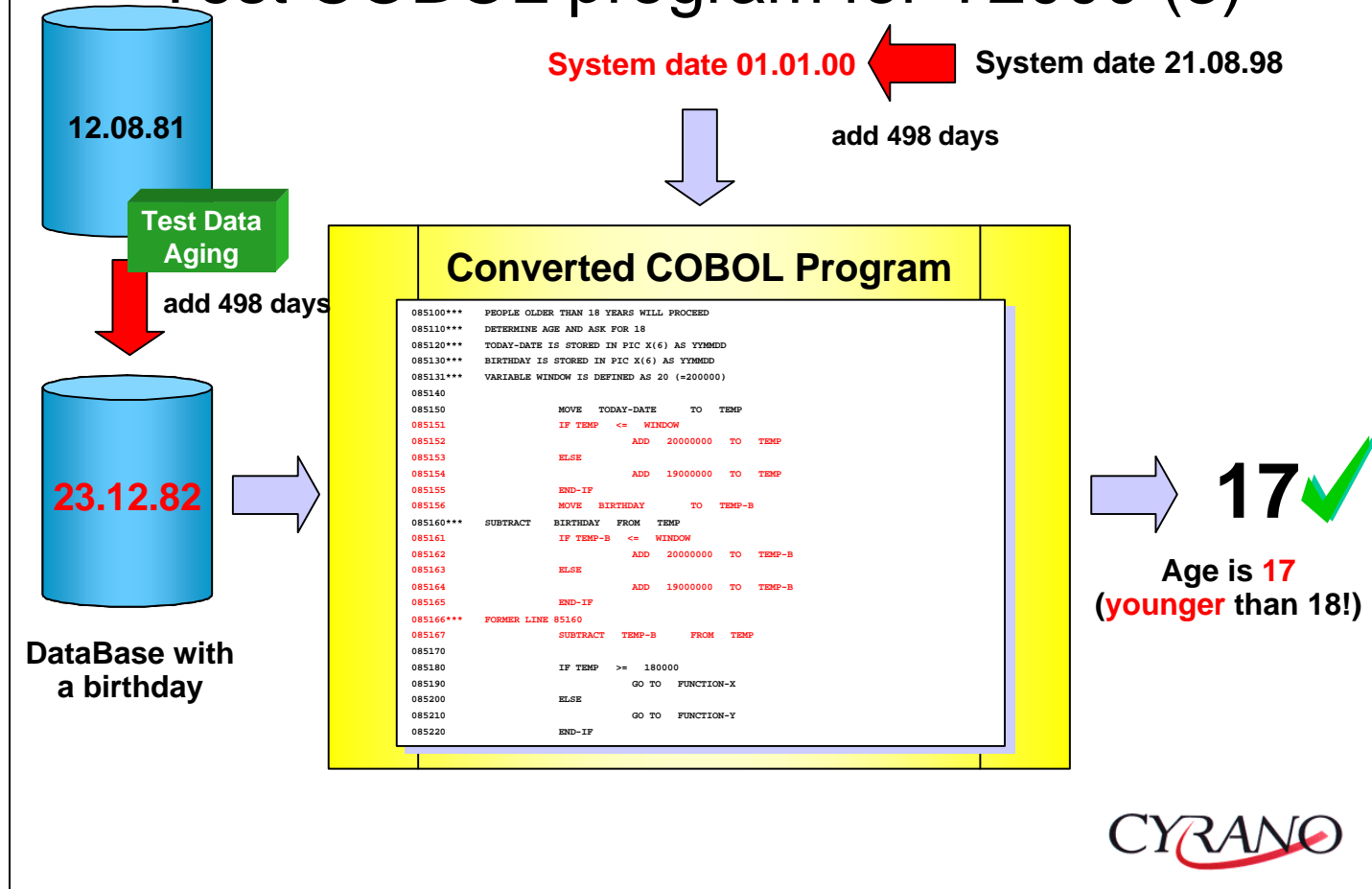
18

Age is 18
(equal 18!)

Test COBOL program for Y2000 (2)



Test COBOL program for Y2000 (3)

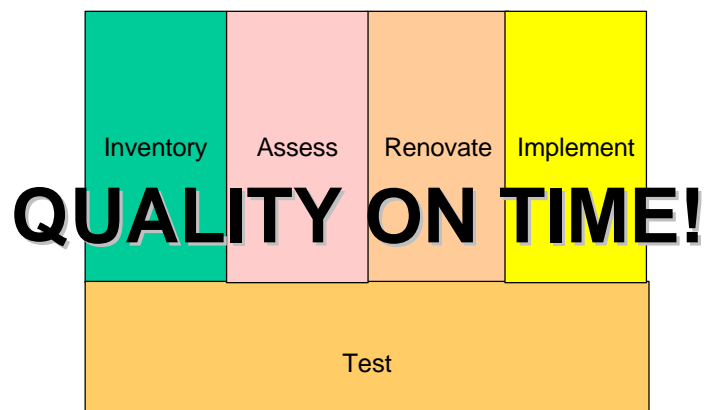


CYRANO Y2000 Products

- CYRANO MillenniumTest
 - record business processes using contemporary dates
 - replay using future dates
 - automatically validate date sensitive I/O for Y2000 compliance
- CYRANO DataAge
 - Semantic data date ageing
- CYRANO DateWarp
 - Date Simulation




De-risked Y2000 Project Plan



EIF

(joint CSSA & EIF initiative)

Code of Practice
Year 2000 Software Product &
System Testing



Has it been tested in
accordance with the EIF Code
of Practice?

CYRANO

EIF Code of Practice

<http://www.cssa.co.uk/home/pubs/practice/y2ktest.htm>

BSI DISC PD2000-1

www.bsi.org.uk/disc/year2000.html


CYRANO

Testing Budget



Testing Budget

- Overall Y2000 remedy costs \$1 to \$5 per LOC
Testing = 50% of overall cost
- Testing = \$500k to \$2.5m per million LOC (=circa 800 programs)
- Testing = 5 to 25 man years per million LOC
- So, for example, a major retail bank might be spending \$150m on testing!




Slide 39

BUT!

The logo for CYRANO, featuring the word in a black sans-serif font with a red swoosh underline under the 'O'.

Slide 40

Using automated test tools

The logo for CYRANO, featuring the word in a black sans-serif font with a red swoosh underline under the 'O'.

Case study - Major Bank (D)

- 40 million LOC
- Overall budget \$40-100m
- <80 man years testing planned
- Testing 8 to 20% of planned overall spend

CYRANO

Case study - HP TMO

- 14 million LOC
- Overall budget \$10m to \$30m (!!)
- <10 man years testing planned
- Testing <10% of planned overall spend

CYRANO

So...

- Planned testing spend is around 10 to 20% of overall budget
- 1million LOC = 1 to 2 man years of testing = £60-100k

CYRANO

Conclusions

- Test to:
 - determine Year 2000 compliance
 - mitigate losses
 - prove that you've exercised due care
 - prove that you haven't introduced new defects as a side effect of Year 2000 renovation
- Use automated test tools to reduce testing effort by 60-80%

CYRANO

jcorden@cyrano.com

www.cyrano.com

www.cssa.co.uk

CYRANO

Slide 1

Req. Mgmt. - Simple Tools ... Simple Processes



Leslie Allen Little
Aztek Engineering
lal@aztek-eng.com

Slide 2

Outline for this talk



- Why build rather than buy?
- Keys to successfully building tools
- Examples of tools to build
- ReqTrack - A simple but powerful tool
- ReqTrack - Costs to implement


Why you should consider building Simple Tools

- Powerful and robust application environments are available (MS OFFICE)
- With realistic goals, costs are minimal
- By leveraging the application environment, maintenance is minimal
- Ability to adapt/evolve your tools
- Ability to reach higher productivity
- Commercial tools expensive/unstable

Why you should keep Processes Simple

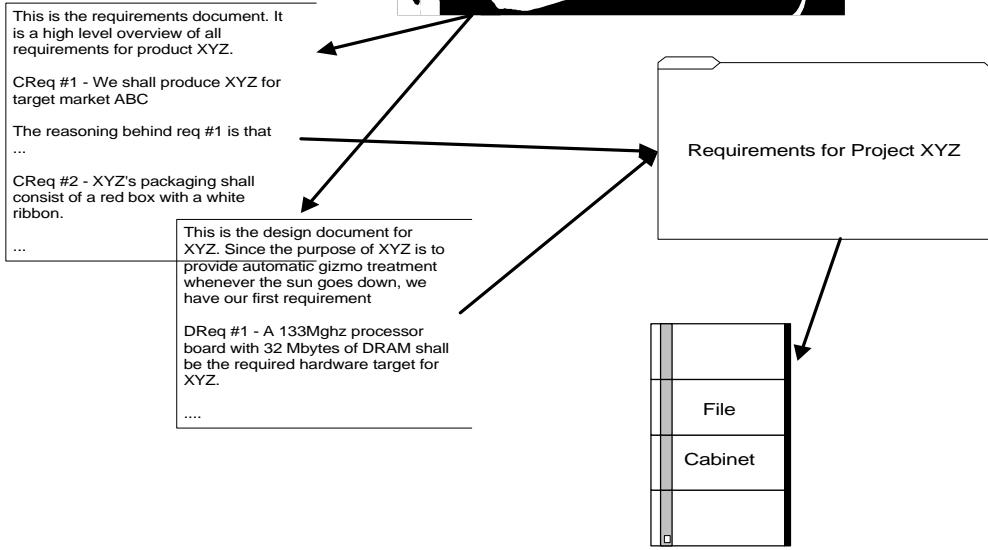
- The simpler the processes, the more likely it will be used
- Simple processes do not require complex tool support
- Simple processes are implemented fast
- Starting simple and learn by example ... then expand!

Example of Simple Tools to aid Processes

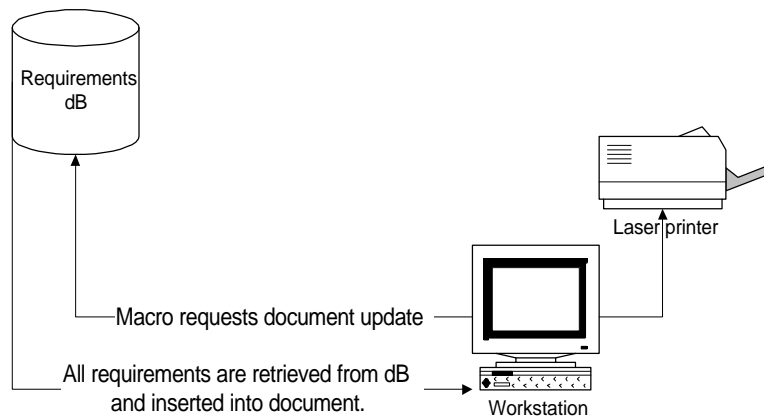


- Automating Software Reviews and Inspections
- Human Resource Allocation for project management tracking process
- Defect Tracking
- Requirement and Test Case Management

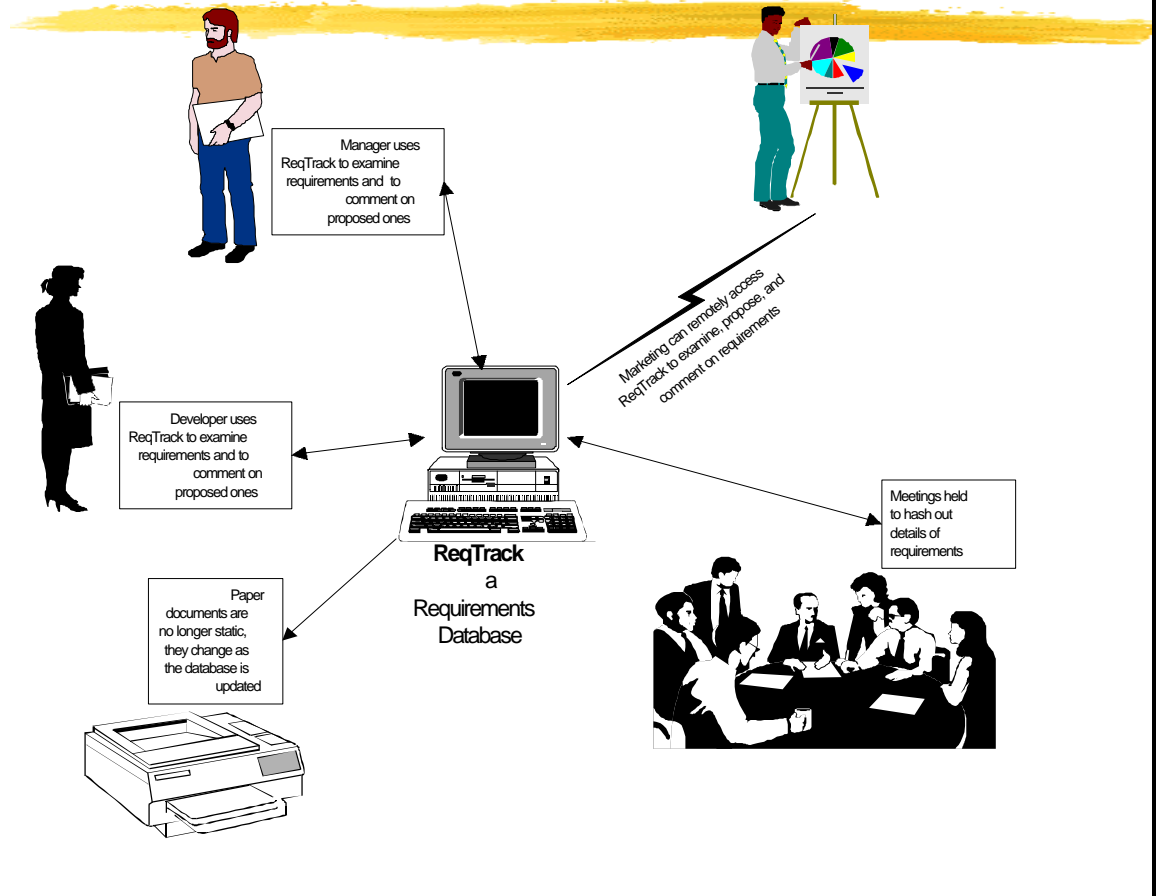
The Typical Requirements Process Model



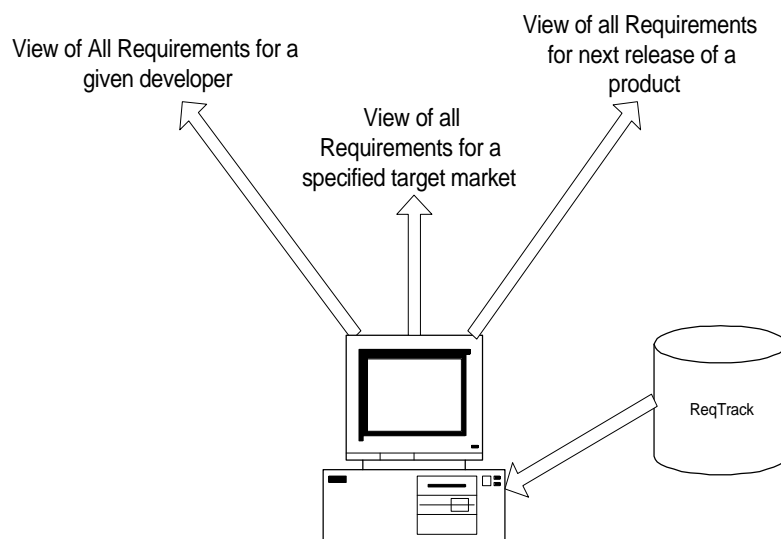
Why not this Requirements Process Model?



Now everyone can have access ... all the time!



... and everyone can see what they need easily ...



ReqTrack - A Closer Look

- Requirement Management
 - Linkages between Word and Access DB
 - Linkages between requirements and tests
 - Definitions Repository
 - Proposal Mechanism
 - Review Mechanism
 - Recorded History of Requirement Changes
 - Online Help
 - Ability to easily change behavior

ReqTrack - A Closer Look

- Test Case Management
 - Linkages between requirements and tests
 - Definitions Repository
 - Proposal Mechanism
 - Review Mechanism
 - Recorded History of Test Case Changes
 - Recorded History of all Test Case Executions
 - Online Help
 - Ability to easily change behavior

Implementation Costs

- MS Access VBA Code
 - 1200 NCSL (Non-Commented Source Lines)
 - ~400 NCSL automatically generated
- MS Word VBA Code
 - 350 NCSL
- Development Effort for ReqTrack
 - 6 man-months effort

Final Thoughts

- You should consider building simple tools because
 - With today's tools ... the costs are right
 - Implementation costs reasonably low
 - Maintenance costs very low if tool kept simple
- Process tool aids are prime candidates
- ReqTrack is free for you to start with

Requirement Management - Simple Tools ... Simple Processes

Leslie Allen Little
Aztek Engineering
2477 55th St. Suite 202
Boulder, CO 80301

lal@aztek-eng.com www.aztek-eng.com

Table of Contents

1	ABSTRACT.....	4
2	INTRODUCTION.....	4
3	WHAT IS A REQUIREMENT MANAGEMENT PROCESS?	5
3.1	WHY IT IS IMPORTANT.....	5
3.2	HOW TO CREATE THE PROCESS	5
4	REQTRACK AND REQUIREMENT MANAGEMENT	6
4.1	A WORD ABOUT THE TOOL ... REQTRACK	6
4.2	ESTABLISHING A FORMAL REQUIREMENT PROCESS	6
4.2.1	<i>Simple Yet Comprehensive</i>	6
4.2.2	<i>Recording of Definitions and Terms</i>	6
4.2.3	<i>Avoiding Ambiguous Requirements</i>	7
4.2.4	<i>Gatekeeper and Champion of the Process</i>	8
4.2.5	<i>Discussion Mechanism for Proposed Requirements</i>	8
4.2.6	<i>Ability to Maintain Proposed Requirement States</i>	10
4.2.6.1	Requirement Process Flow	10
4.2.7	<i>Timeliness in Handling of Proposals</i>	11
4.2.8	<i>Clearly Communicated Vision of the Process and Management Buy-in</i>	11
4.2.9	<i>Requirements as Living Organisms</i>	12
5	COMMON PITFALLS OF REQUIREMENT MANAGEMENT.....	12
5.1	NO REQUIREMENTS DISCUSSIONS.....	12
5.2	CAN'T AGREE UPON REQUIREMENTS LANGUAGE.....	12
5.3	REQUIREMENTS STORAGE EXCLUSIVELY USING PAPER DOCUMENTS.....	13
5.4	NO ACCOUNTABILITY.....	13
5.5	NOT ASSOCIATING REQUIREMENTS WITH TESTS	13
5.6	DEPTH VERSUS BREADTH.....	13
5.7	DEFINING ALL TERMS	13
5.8	MINIMIZING THE EXPECTATION GAP - SETTING THE RIGHT EXPECTATIONS	13
6	CONCLUSIONS	14
7	ACKNOWLEDGMENTS	14
8	TRADEMARKS.....	14
9	APPENDIX 1.....	14
10	REFERENCES.....	16

Table of Figures

Figure 1 - ReqTrack Definitions Screen 7

Figure 2 - Proposal Comments 9

Figure 3 - Comments Report 10

Figure 4 - Default Proposed Requirement Process 11

1 Abstract

There appears to be a certain misconception regarding what commercially available requirement management packages will do for an organization. Commercially available packages are still young to the market¹, and as such are reasonably expensive and evolving. Where they will end up and what features they will support are still largely uncertain. Many of the essential capabilities, offered by these packages, are easily duplicated using Microsoft Office as a development platform.

The objective of this paper is to share simple, workable ideas that when accompanied by Microsoft Office based software provide many of the capabilities available today in commercially available requirement management packages. In this way, organizations can try out a requirement management system, learn to change the work habits and processes prior to, or instead of, purchasing a system. Thus, this paper discusses

- The requirements management process—what it is; why it is important; how to effectively create such a process
- Common misconceptions and pitfalls of the requirement management process
- Using ReqTrack as the tool to aid in the requirement management process

2 Introduction

Over the years, numerous studies have indicated that the cost of correcting errors increases exponentially the longer the error remains in the system. Conclusions from these findings suggest that attacking the process associated with formulation and management of requirements may be the most cost-effective manner in reducing software product costs.

A handful of tool vendors have recognized this and have created products to facilitate the requirement management process. Unfortunately, the fact remains that for the commercially available requirement tools, there are

- A limited selection of these tools
- There is a wide range of approaches to requirement management
- There is no connectivity between requirements and test cases
- The majority of the tools are reasonably expensive (see Appendix 1).

When one examines the current requirement management processes instituted at most organizations, it is apparent that a requirements management tool alone will not solve the requirement management process problem. What is required is a holistic approach to requirement management, some simple tools to aid the process, a champion of the ideas, and management buy-in. This paper takes this approach by discussing the requirement management process with examples taken from ReqTrack, a requirement management tool the author has created using Microsoft Office products.

It is the author's hope that many of those reading this material will decide to learn about requirement management through the use of ReqTrack. Managing your requirements is critical to the success of all projects since they are the definition of the product.

¹ Infancy when you consider that the earliest surviving products became available during the past few years.

3 What is a Requirement Management Process?

A requirement management process is the formal act of defining how requirements are created, modified during the life cycle of the product, and retired. A more specific definition by way of INCOSE² is

Requirement management is the identification, derivation, allocation, and control in a consistent, traceable, correlatable, verifiable manner of all the system functions, attributes, interfaces, and verification methods that a system must meet including customer, derived (internal), and specialty engineering needs.¹

Requirements management, when properly performed, reduces the time that engineers spend finding the information that they need to do their job, eliminates version mix-ups, and reduces errors.

3.1 Why It is Important

The implementation of any tool requires a process and requirement management is no different. In fact, you probably already have processes associated with your requirement management system now, be it formal or informal. The more formal the process, i.e., how requirements come into existence, how they are reviewed and adapted, how the information generated during this process is captured or lost, etc., the more likely your process will meet its intended objectives. Using a tool to aid in this process increases the likelihood that the process will succeed as long as the tool is flexible and can be melded to fit your process.

3.2 How to Create the Process

The creation of a requirement management process is most easily performed by

- Reviewing the common capabilities of requirement management tools (see section 9 for a list of current tools)
- Examining your organizations methods and modes of work habits
- Combining your organizations work methods with accepted practices from requirement management research and development

As you will see in the remainder of this paper, there are a number of requirement management process issues to consider, with some of the more important ones being the need to

- Provide information to the interested parties for requirement proposals
- Provide definitions for all potentially ambiguous terms
- Link requirements into a hierarchy of requirements (parent and children requirements)
- Identify responsibility for requirements implementation via components and ultimately developers
- Link requirements with test cases to provide for accountability

² INCOSE is the International Council on System Engineering, is a professional organization of systems engineers that fosters the definition, understanding, and practice of systems engineering

4 ReqTrack and Requirement Management

The purpose of ReqTrack is to provide a simple tool to aid in the process of requirement management. The tool has wider applications in that it also serves as a repository for test cases that are linked to requirements. The following paragraphs describe the tool as part of describing the process.

4.1 A Word About the Tool ... ReqTrack

ReqTrack is implemented using VBA in both the Access and Word modules of Microsoft Office. The ReqTrack system requirements are as follows:

- Pentium processor with 16 MB of RAM
- 2 MB of free disk space
- Windows 95 on each user PC using Word Extensions
- Word for Windows 97 on each user PC
- Access 97 installed on each user PC
- ODBC driver for Access on each user PC using Word Extensions

The application is relatively easy to set up and features online help. To request a copy, contact the author at lal@aztek-eng.com. Enhancements are incorporated from the user community and modifications should be submitted back to the author for consideration.

4.2 Establishing a Formal Requirement Process

If one starts with the assumption that product quality is desirable, then formal processes and tools must be developed or obtained. These processes and tools can be quite simple or complex. Contrary to popular belief, obtaining the latest gizmo from a tool vendor will not necessarily solve your problem. Without forethought and the implementation of a formal requirement process, the latest costly requirement tool is most likely destined for failure irrespective of how expensive it is. The implementation of the tool **and a process** are the key ingredients to success.

On the other hand, with a formal process in place, the simplest of tools can aid and abate the process of forming, refining, and managing requirements. The remainder of this document discusses a requirement management process along with how to use ReqTrack to aid in that process.

4.2.1 Simple Yet Comprehensive

Most things one undertakes in life are governed by the 80/20 rule. You try to resolve 80% of your problems with 20% of the effort. The same is true of a requirement management process. For most products, such a system is sufficient. One should try to create a reasonably comprehensive system with a minimum of resources. The typical mistake is to build a grandiose system that never can be deployed while your products suffer from having no formal process in place. Start small and add the increased comprehensiveness as you mature the product and processes.

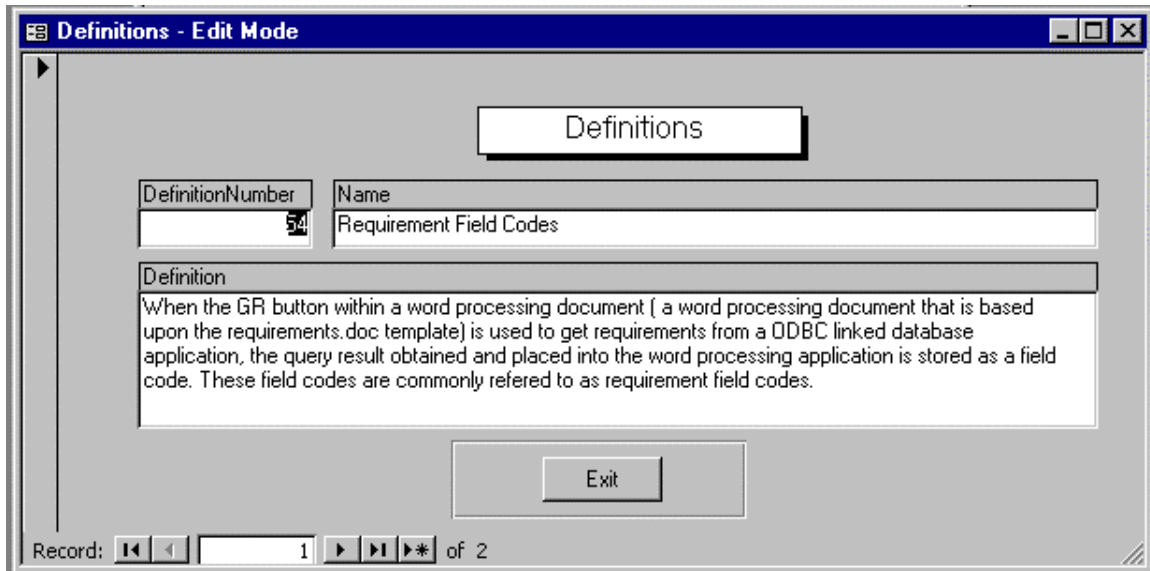
4.2.2 Recording of Definitions and Terms

Most requirement specifications are written in native languages. A direct result of using native languages to specify requirements is the fact that they are inherently riddled with the problem of ambiguity.

One alternative to the problem of ambiguity is to write requirements with mathematical rigor. Unfortunately, this alternative is neither palatable nor plausible for most systems and engineers.

Another approach is to instill discipline in the requirement formulation process such that ambiguities are minimized. The simplest and most common method of doing this is to make liberal use of a requirement definitions table. Figure 1 shows a screen capture of the definition table within ReqTrack. This mechanism is used liberally by ReqTrack users to define ambiguous terms in an attempt to minimize confusion with respect to requirements.

Figure 1 - ReqTrack Definitions Screen



4.2.3 Avoiding Ambiguous Requirements

The source of ambiguities is not limited to misunderstandings about definitions but also includes general clarity issues. When trying to properly word a requirement, it is the author's responsibility to concisely express the essence of what is required of the system. Unfortunately, in most cases, the initially proposed requirement is not always correctly understood. To avoid this problem, a simple but formal process where requirements can be proposed, reviewed, and approved is critical.

Another important method of avoiding ambiguities is to provide context. In most natural language requirement systems, the requirement writer adds a lot of context to the surrounding text of the requirement. This rich context provides a better understanding of what the requirement is all about. ReqTrack provides for this capability by allowing users to link requirements from the ReqTrack database to a Word document.

Another area where context is developed and many times lost is in the discussion phase of requirement proposals. Again, ReqTrack provides a method for storing proposal discussions regarding a requirement for later perusal (see section 4.2.5).

4.2.4 Gatekeeper and Champion of the Process

Every process, particularly when new, must have a champion. To prevent requirements creep³, a gatekeeper should scrutinize each and every requirement being proposed to decide if it is worthy and really required. In many projects, the gatekeeper is a quality assurance representative for the project. The gatekeeper and champion of the process doesn't necessarily have to be the same person but there needs to be enough overlap in the roles to support it. The following are the key responsibilities of both a gatekeeper and champion of the process:

- Makes sure that proposals are properly aired; that the correct stakeholders are participants in the process; and that the process keeps moving. In other words, the gatekeeper is the facilitator of the requirement process.
- Encourages the use of the process and continues to improve upon it. In other words, they are the champions of the requirement process.
- Examines all requirements for redundancy, conciseness, ambiguity, testability, etc.

4.2.5 Discussion Mechanism for Proposed Requirements

It's quite natural to consider the process of proposing requirements and approving them as an initial state that occurs when a project is just beginning and it is true that this occurs. Unfortunately, some people think that this is where requirement management stops. Requirements are proposed, or at least should be easily proposed, throughout the product life cycle. ReqTrack facilitates this process by allowing *any participant* to the project to propose requirements at *any time* during the project. As part of proposing requirements, ReqTrack supports a discussion forum of written discussions.

The exact nature of how the proposal process works is, to some degree irrelevant, since the essence of the process is that there is a process and a reasonably suitable tool to aid in the process. What's important is that there **is a process and that it is used**.

Figure 2 shows an example of the comment form within ReqTrack while Figure 3 provides an example of a report where requirements were grouped on a filter and proposal status state.

³ The undesirable phenomenon of requirements being slowly added to a product over time without conscious acceptance of the new requirements

Figure 2 - Proposal Comments

Requirement Change Proposal Comments - Edit Mode

Requirement Change Proposal Comments

Date/Time	Requirement #	Version	User
7/22/98 9:18:57 AM	1016	0	lal

Change Proposal Comments

Rejected this requirement because it is about the details of the database rather than a description of the behaviors desired. Subsequent requirements will tackle the behaviors.

Description

ReqTrack shall consist of the following requirement related tables: a child requirements table for identifying child and parent requirements; a comment table containing all commentary regarding changes to requirement status and requirement descriptions; a filters table where requirements can be arbitrarily related on similar commonalities; a markets table where requirements can be targetted for particular markets that...

Exit

Record: 20 of 54

There are several advantages of having recorded discussions for proposed requirements with two of the more important being

- Recorded histories of decisions made with respect to a product are important, particularly in the maintenance phase of a project. As you know, many times the developers that implement a product move to new projects once development has completed. Maintenance programmers then join the project to maintain what others have developed. Without a history of why certain decisions were made, these new developers must guess as to the reasoning behind certain decisions.
- Preserving the link between a customer's requirement definitions and the eventual requirements used to build the product provides the ability to ensure the quality of a software system beyond its first release².

Figure 3 - Comments Report

Requirement Proposal Comments By Req and Date

RequirementNumb 1002

Revision 0

Description There shall be a log in window from which the users login ID is gained and used as defaults for all data entry related to the field "user" throughout the application.

<i>Date/Time</i>	<i>User</i>	<i>Comments</i>	<i>RequirementStatus</i>
98 8:57:26 AM	lal	Added requirement.	Internally Approved

RequirementNumb 1003

Revision 0

Description ReqTrack shall be implemented using Microsoft's database product, Access, and the word processing product, Word for Windows. The application language used for software development for both of these products shall be Visual Basic for Applications.

<i>Date/Time</i>	<i>User</i>	<i>Comments</i>	<i>RequirementStatus</i>
8 10:38:36 AM	lal	Adding requirement.	Internally Approved
98 5:05:22 P M	lal	Just changed the wording to be more clear.	Internally Approved

Friday, September 04, 1998 Page 1 of 12

Page: 1

4.2.6 Ability to Maintain Proposed Requirement States

As discussed previously, ReqTrack provides a change proposal process for requirements. In order to support such a process, though, ReqTrack must track three things

- Who needs and has reviewed the proposed requirement
- The proposed requirement's state at any given time
- Who is currently responsible for continuing to move the requirement forward

Given these three items, a reasonably simple process diagram can be managed as shown in Figure 4.

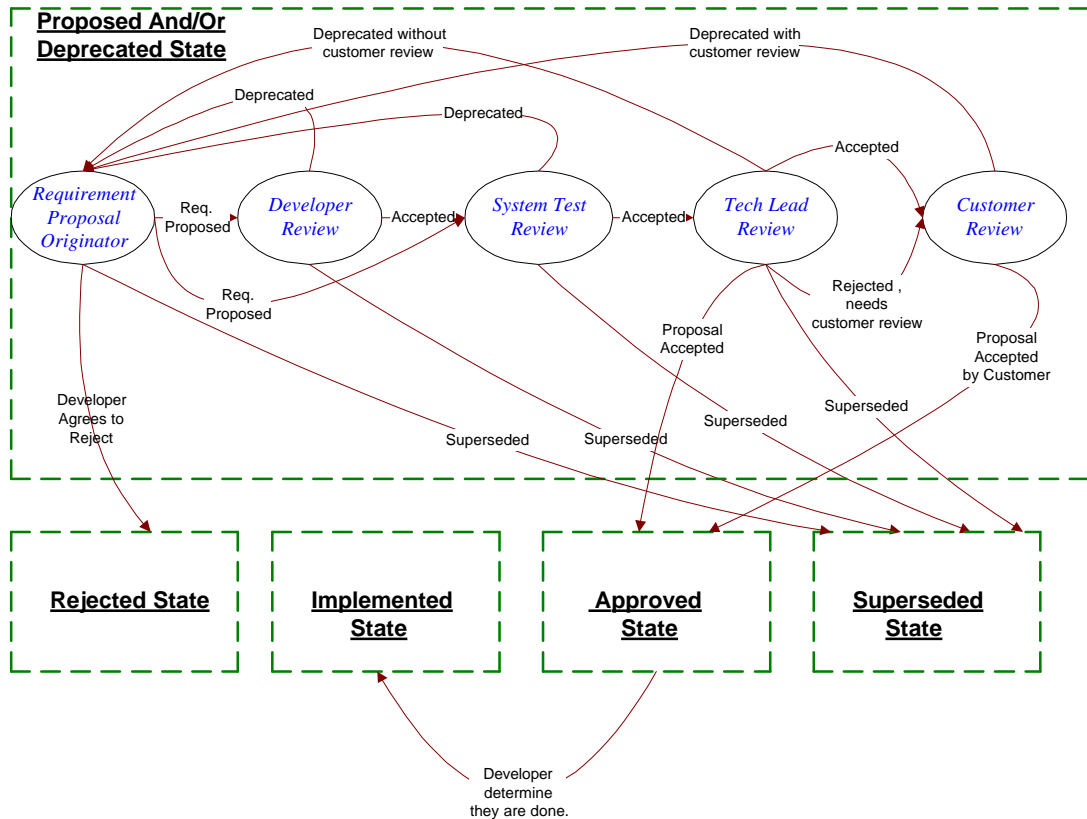
4.2.6.1 Requirement Process Flow

The following is a high-level process flow for requirement state transition

1. Requirements are placed either in a Word document, in ReqTrack, or both. When entered into a Word document, requirements that have a given state can be highlighted for easy identification.
2. Requirements are usually reviewed internally prior to being reviewed by the customer. Note that all requirements to be reviewed must be entered into ReqTrack prior to review.
3. Issues and comments as a result of reviewing the requirements are entered into ReqTrack as proposal comments.

4. After initial review, requirements can be rejected or accepted. If further review is necessary (such as by the customer), then this can be indicated.
5. Almost any state transition is reasonable given the states that exist. Reports and queries can be easily constructed to aid in any requirement state analysis needed.

Figure 4 - Default Proposed Requirement Process



4.2.7 Timeliness in Handling of Proposals

The objective is to ratify or reject requirement proposals within a reasonably short period of time. For a system to run smoothly, the project team has to decide what the reasonable gestation period is and then strive to keep things moving smoothly without great delays; otherwise, the process is subjected to a loss of faith by the participants. With ReqTrack, reports and queries can be easily created to aid in this process.

4.2.8 Clearly Communicated Vision of the Process and Management Buy-in

Informally or formally, all participants need to have a sense of belonging to the project. Without it, the requirement management effort will be wasted. Formally maintaining requirements is beneficial to all those who work on the project. Once the database of requirements is created, the benefits will begin to be realized.

For this to happen though, make certain to document a process⁴ and then stick to it. Tweaks can be made to smooth the process as you go but be certain to feedback the changes to all involved.

Lastly, management must provide adequate time to implement and utilize the process. Without management buy-in, participants will be pressured to ignore or minimize the potential usefulness of the process.

4.2.9 Requirements as Living Organisms

As mentioned previously, a common problem facing today's developer is that of the rapid pace that software projects evolve. By implication, this means that the requirements of the project evolve quite rapidly also. Combined with this rapid evolution of requirements is that software projects are perpetually caught in a crunch mode. There simply isn't enough time to write and test the code, let alone to keep documentation up-to-date. Unfortunately, this quite often leads to an abundance of implicit requirements being introduced into the project.⁵ Implicit requirements are a poor project characteristic. Implicit requirements are, by their very nature, behaviors that developed systems exhibit yet no requirements exist documenting those behaviors. By their very existence, they add considerable hidden costs to the project, confusion to your customers and an endless source of frustration to your test effort. This author has examined this problem in detail and has proposed methods to deal with it (see ³ and ⁴). In general, implicit requirements come about as a by-product of incomplete requirements. The methods to prevent the introduction are

- To provide complete requirements specifications prior to coding
- To recognize that requirements are incomplete and to provide processes and tools to incorporate implicit requirements as explicit ones

This author has argued strongly for the latter solution since the belief that one can ever provide comprehensive requirements prior to coding is unrealistic. All of the material presented in this paper addresses the problem of treating requirements as living organisms. ReqTrack is all about evolving requirements.

5 Common Pitfalls of Requirement Management

There are many potential pitfalls on the path to requirement management. At the risk of sounding repetitive, a number of the more important issues are listed below.

5.1 No Requirements Discussions

Discussions between customers, developers, management, etc., all are critical parts to the success of the requirements process. To not capture, record and preserve this rich data content is a crime. Future developers will need this data when making future decisions.

5.2 Can't Agree upon Requirements Language

So many times, requirements languish because agreement cannot be reached upon the actual wording or the essence of the requirements themselves. This is a very costly mistake. In the end, if you are to build a product, you will make decisions concerning what the requirements are either formally or informally.

⁴ Document it as simply as possible. The more there is to read, the more people will not read it.

⁵ Implicit requirements are defined as requirements of the system that have never been documented.

By allowing this to take place informally you are facilitating the creation of implicit requirements and greatly increasing the odds of a failed project.

5.3 Requirements Storage Exclusively Using Paper Documents

Historically, requirement storage was exclusively the domain of paper documents. In today's world, it just doesn't make sense to do so. The benefits from storing requirements in databases far outweigh any inconvenience. The power of the database is tremendous and greatly enhances the probabilities of a successful product.

5.4 No Accountability

Most developers want to do what's right, they just need the tools and support to do it. By associating requirements with components and associating components with developers, it is easy for developers to understand the requirements of the components they are developing. It is easy for them to add requirements when they are missing, modify existing ones that are no longer correct and remove outdated ones.

5.5 Not Associating Requirements with Tests

Quality assurance focuses upon verification and validation. When testing a product, if you don't have requirements, how can you adequately test the product? If you don't have a mapping between requirements and test cases, then how can you know that you have completed your testing? Not associating requirements with test results in shipping a product with no real understanding of its quality.

5.6 Depth versus Breadth

One of the most common pitfalls of requirements is that the deeper you dig, the more you find. Every project must decide *a priori*, how detailed they want their requirements to be. The tradeoff between too much detail versus too little is a classic dilemma. Management and the development community must decide how much detail is enough, develop some guidelines on how to make these decisions and then stick with them. Like all process decisions, if circumstances require it, the process can be modified.

5.7 Defining all Terms

In every industry, there are special meanings given to certain words. Similarly, there is an overabundance of acronyms that must be defined. Empowering all project participants to add and retrieve from a common terminology pool is another essential element to a successful requirement management process.

5.8 Minimizing The Expectation Gap - Setting the Right Expectations

A big part of the requirement management problem is underestimating the job. Setting the correct expectations is the best way to avoid disappointment. Creating and managing requirements is a big job and one not to be underestimated. Requirement work typically requires a good 20% of overall recorded product effort. This doesn't include unrecorded product effort where requirement issues are the real source of problems and the time used, although the effort is recorded to other activities such as design or coding.

Lastly, do not expect everything to be perfect. The requirement management system is simply a tool to increase the likelihood that your project will meet with success, not a guarantee of success.

6 Conclusions

Requirement management is an essential piece of verification and validation. It involves two key components ... process and tools. The process does not have to be complicated, in fact, this author argues that you should keep it as simple as possible in the beginning and slowly add more complexity as is needed. Similar to the process argument of simplicity, the tool should also be just that, simple. Given the cost of commercial tools, this author has presented a viable alternative that is free. The tool implements the essential pieces of a requirement management system and is quite simple to use. It is the author's hope that you, and many others like you, provide the process that fits with your organizations strengths and weaknesses and utilize ReqTrack to implement the process.

7 Acknowledgments

It is always with my children in mind that I continue to dedicate my time and efforts. Sharing my ideas and efforts with others in this field of endeavor provides the broadest opportunity for success in all of our efforts and thus provides a better world for all of our children.

8 Trademarks

All brand names and product names used in this document are trademarks, registered trademarks, or trade names of their respective holders.

9 Appendix 1

The following is a list of currently available requirement management tools that the author is aware of. These tools continue to evolve as new ones enter the market. The author has only peripheral knowledge of most of these tools and their listing here should not be construed as a recommendation to purchase and use them. If anything, this author argues that you are better off to either use ReqTrack, that is free, reasonably robust and mostly defect free, or to develop your own ReqTrack lookalike.

<i>Product Name</i>	<i>Major Features</i>	<i>Average Cost Per User</i>
Core™, Vitech Corp, (www.vtcorp.com)	A desktop CASE solution for engineering, analyzing and documenting systems and processes. CORE's automated system engineering capabilities include: requirement analysis; traceability management; system modeling; design verification; and document generation. Uses RDT as notation language. Performs the complete life-cycle function.	\$27,000 for first user license, server and database the \$10,000 per user license.
Cradle™, Structured Software Systems, (www.3sl.co.uk/sssl)	Requirement document management. Traces requirements directly onto system models and other information types. Provides full lifecycle traceability.	\$4500 per user.
DOORST™, Quality Systems & Software, (www.qssinc.com)	Creation of single documents, document trees, linking requirements from unstructured information, etc. Integrated with Aonix StP toolkit.	\$14,000 per user.
RequisitePro™, Rational Software Corp., (www.rational.com)	Integrated with Rational Rose, ClearCase, MS SourceSafe, Intersolve PVCS. Traceability to test plans.	\$1559 per user.
RDD-100™, Ascent Logic Corporation, (www.alc.com/products.html)	Full text editing, report execution, and configuration management using the multi-user Merge feature. Graphical display is limited to Functional Flow Block Diagrams, N-squared Charts, and Hierarchy Views.	
RDT™, GEC-Marconi Systems Pty Ltd, (www.world.net/gecm)	Allows the formulation of requirements; traceability and management of requirements throughout the development cycle of a contract; test procedure definition and allocation to requirements throughout the development cycle of a contract and office management of documentation and drawings associated with a contract.	\$4550 for first 5 users.
SLATE™, TD Technologies, (www.tdtech.com)	Symbolic references, Automated Trace Tables, Process enforcement.	\$5000 per floating license.
VITAL LINK™, Compliance Automation Inc., (tlmworks.com/cai)	Multi-user requirement management software designed to support the documentation of requirements. Imports and parses existing documents while retaining formatting, figures, and graphics. On-line access to documents for creation and update, and provides you with the attributes needed to maintain requirements over the life cycle.	\$1750 per user.
Xtie-RT™, Teledyne Brown Engineering, (www.tbe.com/products/xtie)	Systems and project management. Formal training not required. Requirement & functional & risk analysis. Also supports testing.	Server \$1000 Clients \$400 per user.

10 References

¹ David A. Jones, Pradip C. Kar, James R. van Gaasbeek, Frank Hollenbach, Marty Bell, and Dr. Robert S. Ellinger, Interfacing Requirements Management Tools In The Requirements Management Process - A First Look, Submitted for Volume 2 of Proceedings of 7th Annual International Symposium of the INCOSE, August 1997.

² Moores, T. T. and Champion, R. E. M., Software Quality Through the Traceability of Requirements Specifications, Software Testing, Reliability and Quality Assurance, International Conference on Software Testing and Reliability and Quality Assurance, December 1994 (New Delhi).

³ Leslie Allen Little, Taking a Peek Inside the Black Box, 1997 (STAR, 1997) Proceedings of the Sixth International Conference on Software Testing, Analysis & Review.

⁴ Leslie Allen Little, Making Implicit Requirements Explicit. An Application of the Peeking Methodology, Fifteenth Annual Pacific Northwest Software Quality Conference, October 28 - 29, 1997, (Portland, Oregon).

Bellcore

Year 2000: Catalyst for Better Ongoing Testing

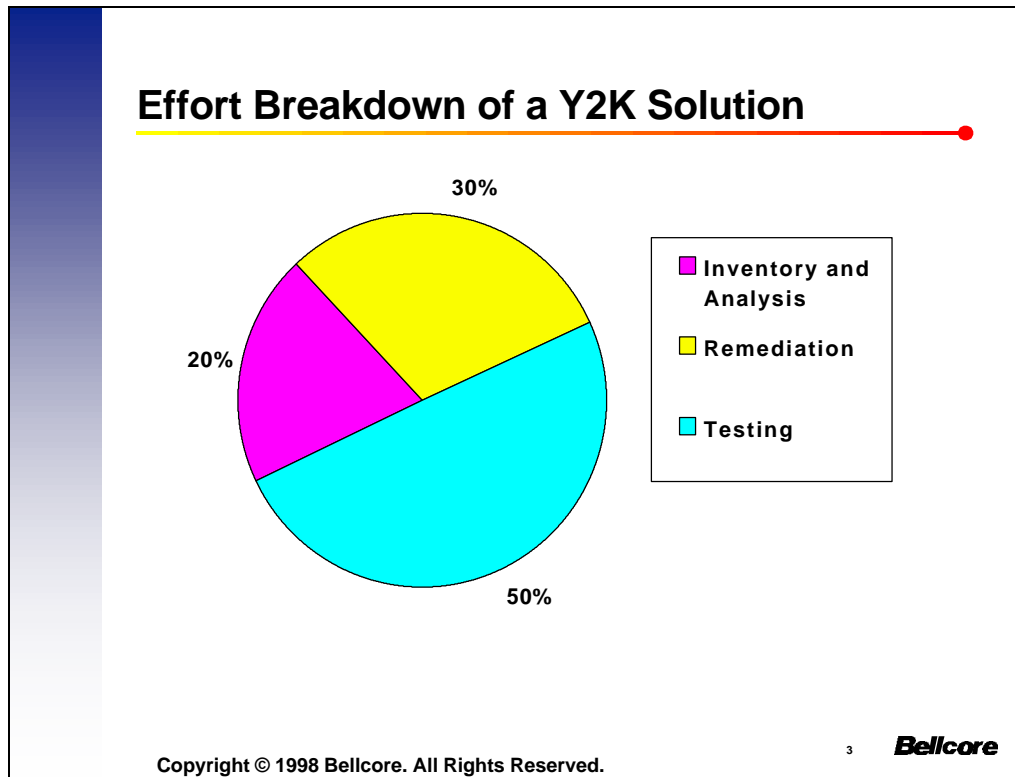
Nathan H. Petschenik
Principal Consultant
Bellcore
(732) 699-8249
nhp@superlink.net

Copyright © 1998 Bellcore. All Rights Reserved.

Outline

- Year 2000 Testing Challenge
- Solution Approach: Repeatable, Date Dimensional Tests
- An Example
- Value Beyond Year 2000
- Summary and Conclusions


Copyright © 1998 Bellcore. All Rights Reserved. 2 **Bellcore**



- ### Year 2000 Testing
- | | |
|--------------------------|---|
| Test Environments | <ul style="list-style-type: none">• Additional facilities required for date sensitive testing |
| Date Simulation | <ul style="list-style-type: none">• Tools available• But not for all platforms |
| Setting the System Clock | <ul style="list-style-type: none">• Licenses, passwords, embedded dates, ... |
- Copyright © 1998 Bellcore. All Rights Reserved. 4 **Bellcore**

Year 2000 Testing

Magnitude of Test Cases




- Test Cases that exercise the mission critical capabilities of your systems
- Typical dates
- Century Transition dates
- Leap Year dates
- Windowing boundaries
- Holidays
- Time Zones
- Rollover situations
- User inputs, Data Bases, Electronic Interfaces, ...

Copyright © 1998 Bellcore. All Rights Reserved. 5 **Bellcore**

Year 2000 Testing

Tests need to be Repeated




- To ensure that Y2K changes do not break existing capabilities
- For high risk settings of the system clock
- As problems are found and fixed
- When source code with Year 2000 changes is merged with source code for other new features
- In a final "dress rehearsal" for the Century Change
- As future changes are made to the system

Copyright © 1998 Bellcore. All Rights Reserved. 6 **Bellcore**

Y2K Testing Challenge

- Impact pervades virtually all system features
- Existing tests don't emphasize date sensitivities
- Large number of Y2K date related test situations
- Tests have to be repeated multiple times

 "too much to do in too little time"

Copyright © 1998 Bellcore. All Rights Reserved. 7 **Bellcore**

Solution Approach

*Supplement your existing tests with additional tests that emphasize the **date sensitivities** of **important business processes** in a manner that they can be easily **repeated** for a variety of **risky date situations***

Copyright © 1998 Bellcore. All Rights Reserved. 8 **Bellcore**

Repeatable, Date Dimensional Tests

- Tests that focus on the date sensitivities of business processes
- Tests that have been designed so that key dates can be easily shifted in time
- High quality, well-documented tests in which
 - key expected results have been identified
 - actual results can be easily compared against expected results
 - repeatability problems have been solved
- Automation optional but highly recommended

Example: Order, Receipt, Payment Process

Step 1 - Requisition

On-line screen used to request equipment, materials, supplies

Step 2 - Order Placement

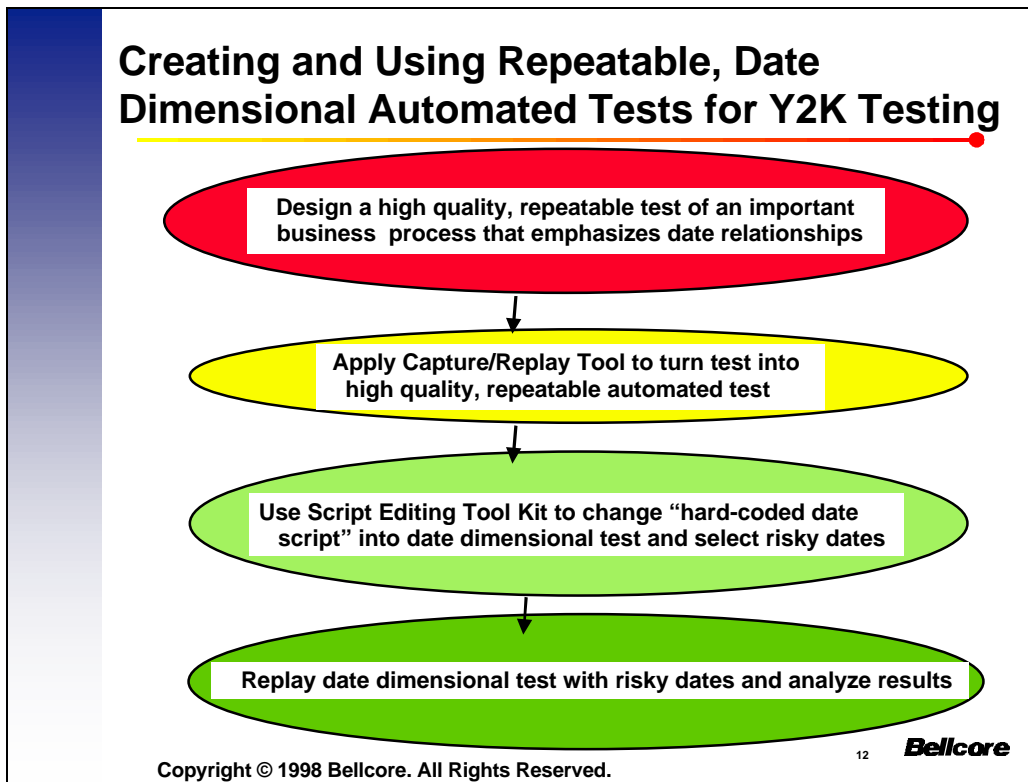
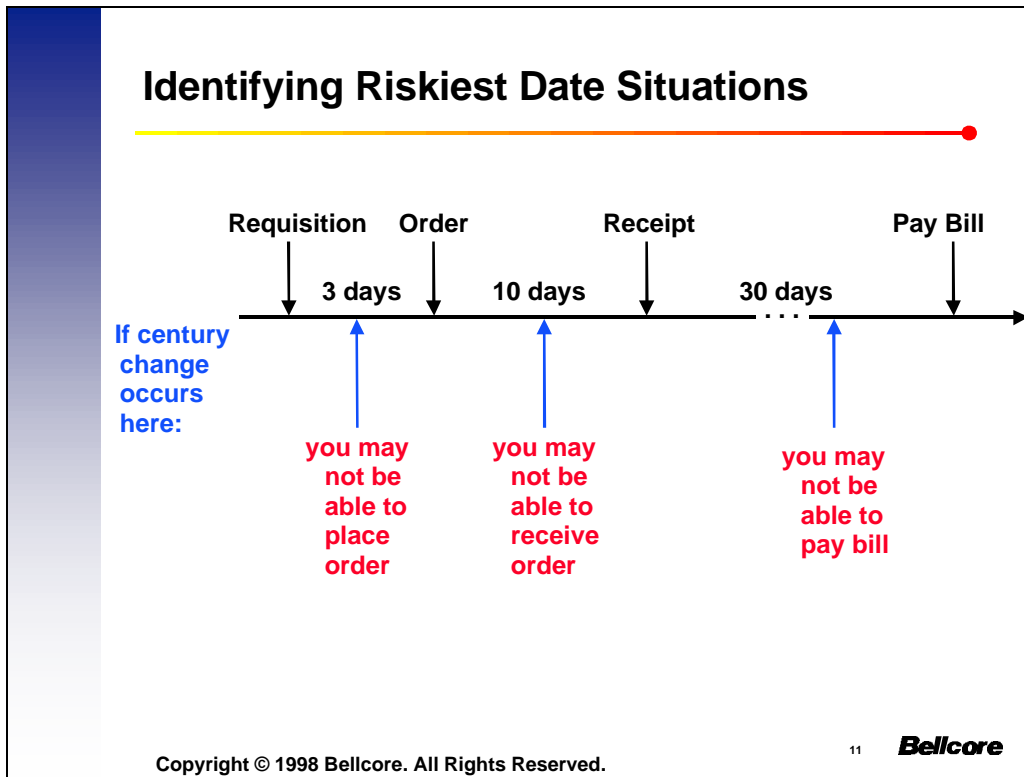
A batch run turns requisitions into vendor orders

Step 3 - Receipt of Goods

On-line screen used to record receipt of goods (and automatically compute when payment should be made)

Step 4 - Payment

A batch run initiates payments as they are due.



Developing High Quality, Date-Sensitive Test of Order, Receipt, Payment Process

- Create a “test story” that defines a typical flow with typical data
- Plan approach for emphasizing date relationships
- Figure out what results are to be observed and compared with what is expected on each execution
- Solve repeatability problems
- Develop comments/documentation that need to become part of script to ease maintainability

On-line screen used to request equipment, materials, supplies

↓

A batch run turns requisitions into vendor orders

↓

On-line screen used to record receipt of goods (and automatically compute when payment should be made)

↓

A batch run initiates payments as they are due.

On-line screen used to request equipment, materials, supplies

↓

A batch run turns requisitions into vendor orders

↓

On-line screen used to record receipt of goods (and automatically compute when payment should be made)

↓

A batch run initiates payments as they are due.

13 **Bellcore**

Copyright © 1998 Bellcore. All Rights Reserved.

Turning a High Quality, Repeatable Test into an Automated Script

Req. Order, Receipt, Payment Test

- Prepare test data base
- Bring up Req entry screen
- Enter Req
- Hit PF Key 1
- Check for success
- Place order 3 days later
- Check for Success
- Bring up Req query screen
- Check for Success
- .
- .

Capture/Replay Tool

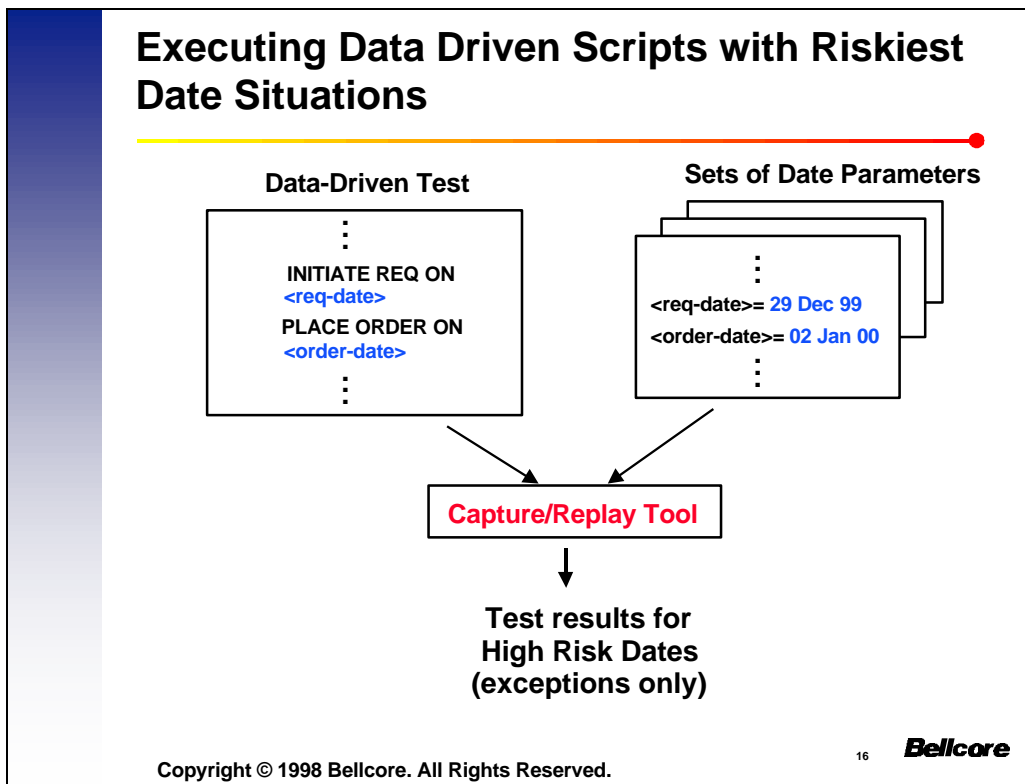
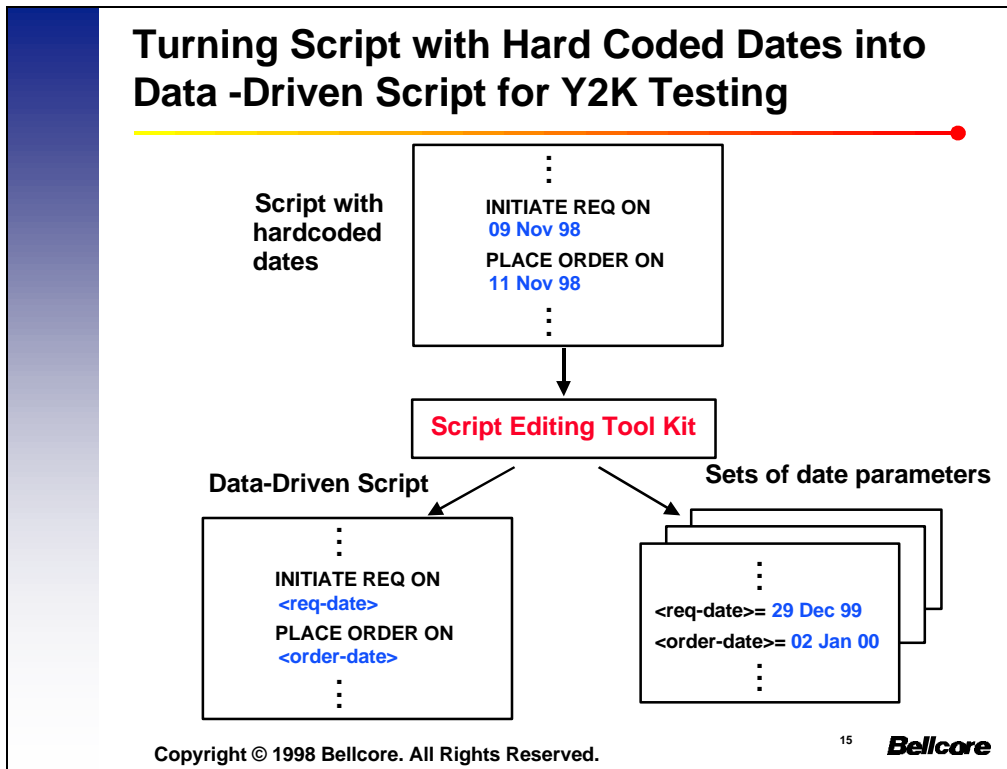
System under test

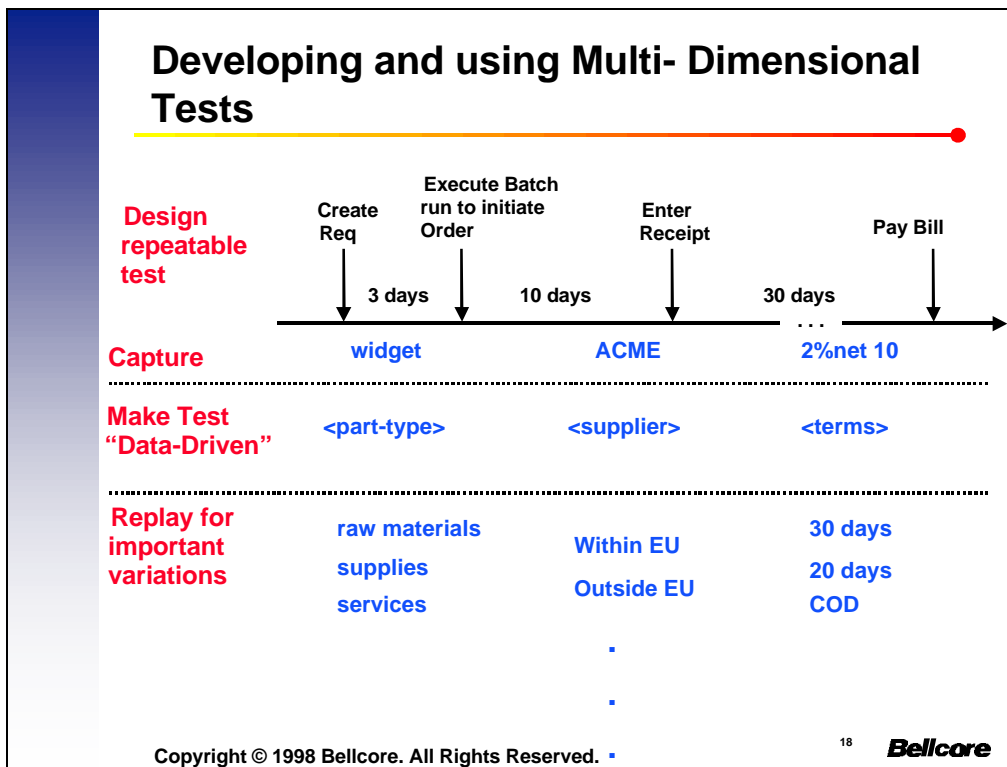
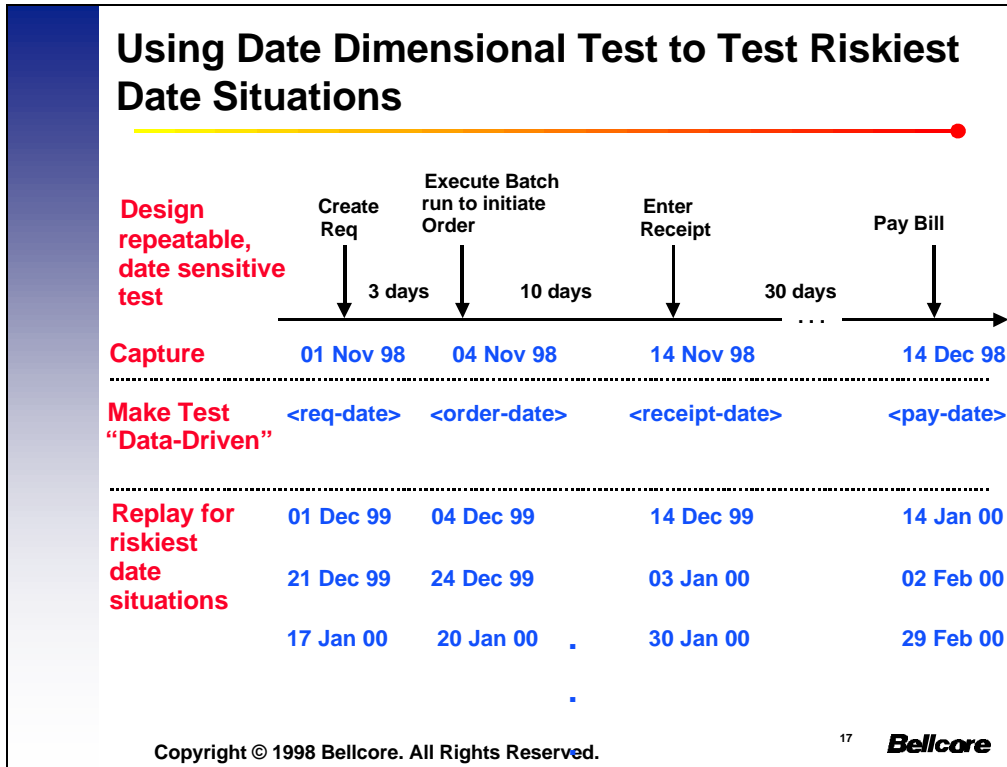
Automated Script with hardcoded dates

(e.g. “PLACE ORDER ON 11 Nov 98”)

14 **Bellcore**

Copyright © 1998 Bellcore. All Rights Reserved.





Value Beyond Year 2000

1. Business flow-through testing being performed
2. High quality, repeatable tests of business flows
3. High quality, repeatable, automated tests of business flows
4. High quality, repeatable, automated, multi-dimensional tests of business flows

Expanded Test Environments

Value

Copyright © 1998 Bellcore. All Rights Reserved. 19 **Bellcore**

Summary and Conclusions

- The Y2K Testing Challenge is “too much to do in too little time”
- An effective solution approach is to reuse repeatable, date dimensional tests across the highest priority risk situations
- The Y2K environments, tests, and test development framework can produce lasting value in your ongoing testing program

Year 2000 can be an investment in better testing and higher quality systems

Copyright © 1998 Bellcore. All Rights Reserved. 20 **Bellcore**

Making Industrial plants Y2K-ready: Concept and Experience at ABB

Juan S. Jaliff (ABB Corporate Research, Sweden)

Wolfgang Eixelsberger (ABB Corporate Research, Norway)

Arne Iversen (ABB Industri AS, Norway)

Roland Revesjö (ABB Industrial Products, Sweden)

International Software Quality Week Europe 1998

© 1998 ABB



ABB Y2K Organization

Juan S. Jaliff, November '98



Outline

- Introduction
- Background
- Y2K upside in process control
- Y2K downside in process control
- Y2K problem in two dimensions
- Advant in-house testing
- Advant software and Y2K
- Plant readiness process
- Field testing
- Y2K concept at ABB
- ABB experience to date
- Tips from the experience
- Lessons beyond 2000



ABB Y2K Organization

Juan S. Jaliff, November '98



Introduction

- Industrial plants throughout the world are heavily dependant on continuous production
- The production process is controlled by computers
- ABB is a major supplier of Process Control equipment and plant systems
- Many plants have critical missions
 - safety of operators
 - functioning of society
 - critical supply to customers

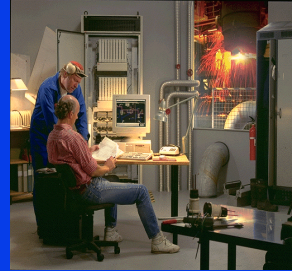
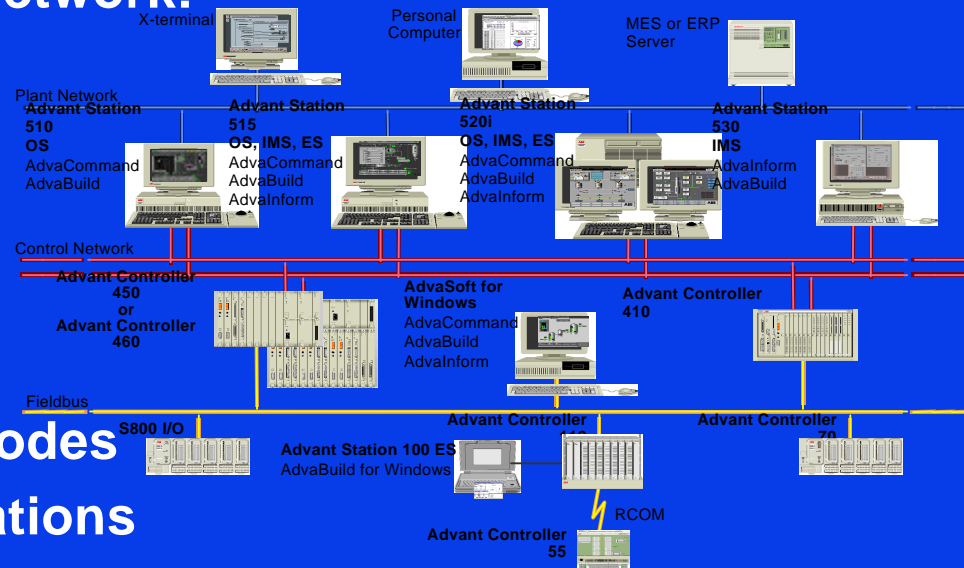


ABB Y2K Organization Juan S. Jaliff, November '98



Background

- **Advant[®] family of products distributes functions throughout a network:**



- **Controller nodes**
- **Operator stations**
- **Information management stations**



Y2K upside in process control

- Potential Y2K error mechanism is simple:
 - year stored without century
- Process control is rarely affected:
 - control loops do not rely on year



ABB Y2K Organization Juan S. Jaliff, November '98



Y2K downside in process control

- Complex interaction among many network nodes
- Systems from different vendors often involved
- Misinformation could lead to wrong operator intervention due to, e. g. :
 - missing alarms
 - alarms sorted in wrong sequence
 - missing events (valve open/close, etc.)
 - event chain in wrong sequence



ABB Y2K Organization Juan S. Jaliff, November '98



Y2K problem in two dimensions

Hierarchical depth/node

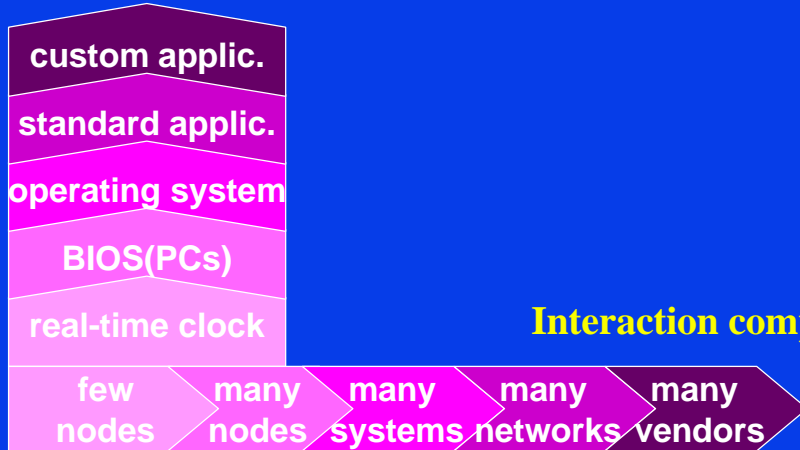


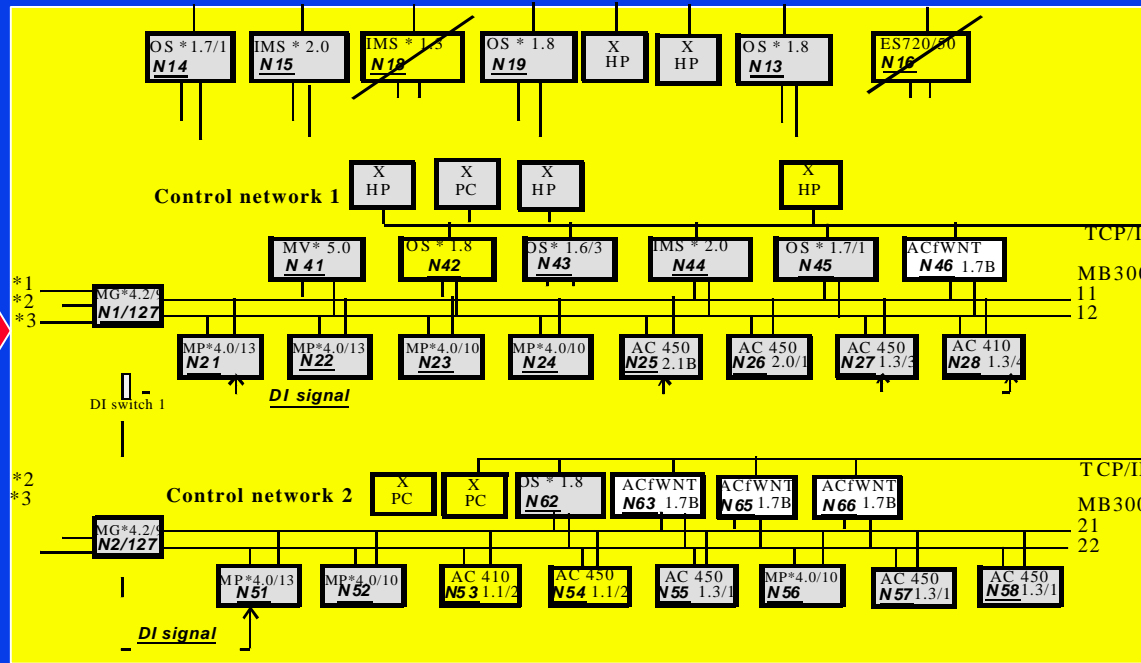
ABB Y2K Organization

Juan S. Jaliff, November '98



Advant in-house testing

- At ABB labs along the two dimensions, from component testing to large configurations



Advant software and Y2K

- DCS (distributed control system) with open architecture
- Controller nodes have pre-defined function blocks (supplied with AMPL, the ABB Master Programming Language):
 - standard usage is Y2K-compliant
 - large address space (for dates, etc.)
 - date usage in only five blocks
 - ABB tools for Y2K code checks
- High-level languages at some plants
 - ABB has evaluated third-party tools for Y2K code checks



ABB Y2K Organization Juan S. Jaliff, November '98



Plant readiness process

- Produce *inventory* of Y2K-sensitive equipment and systems
- Obtain *vendor statements* of Y2K-compliance
- Decide upon *upgrade paths* to compliant versions
- *Procure/install* the compliant versions
- *Update application software* accordingly
- Identify internal/external *data links*
- Carry out *on-site tests*



ABB Y2K Organization Juan S. Jaliff, November '98



Field testing

- Scope of on-site tests defined by plant operator, taking into account:
 - safety relevance per (sub)system
 - production relevance
 - cost of production losses
 - documented Y2K workarounds
- Extent of on-site tests defined taking into account:
 - number of data connections among different systems
 - Y2K in-house test scope per vendor



ABB Y2K Organization Juan S. Jaliff, November '98



Y2K concept at ABB

- Success factors for Y2K plant-readiness:
 - plan early
 - involve high-level management
 - involve operation/maintenance engineers

- Four pillars at ABB:

Y2K Plant Readiness



Product investigation



Plant inventory



Pilot projects (50)

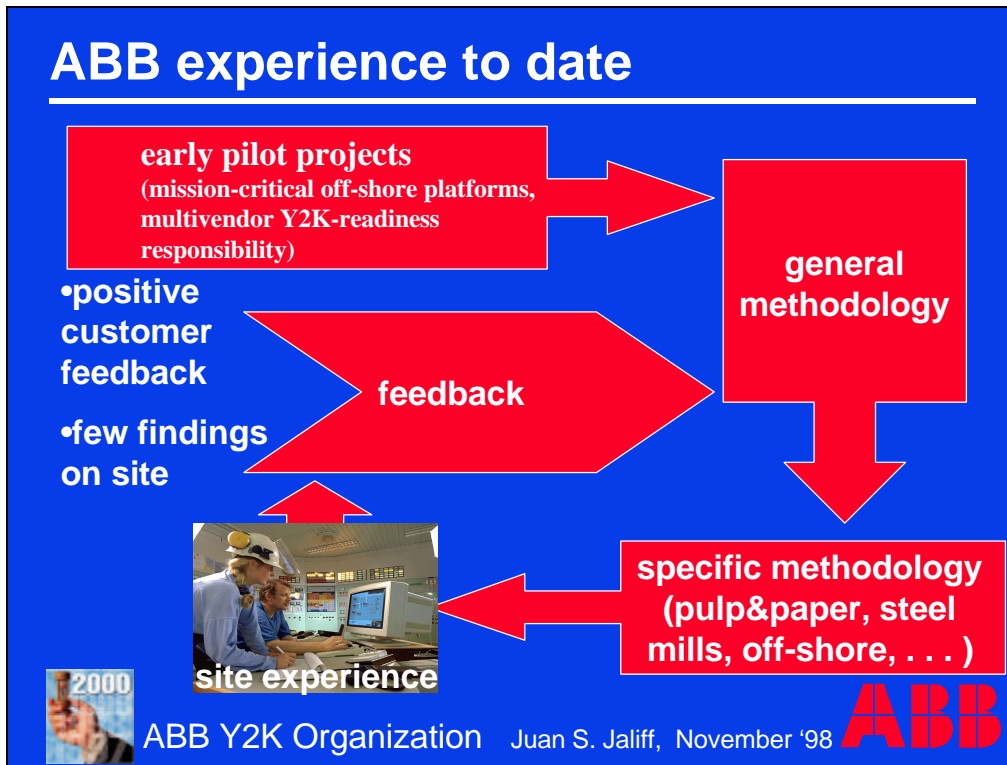


Training



ABB Y2K Organization Juan S. Jaliff, November '98





- ## Tips from the experience
-
- ASAP: inventory & vendor Y2K statements
 - Identify time windows for upgrades/tests
 - Analyze the full impact of upgrades
 - Analyze vendor in-house tests
 - Plan for some workarounds
 - Plan standby system support for 2000
- 2000
- ABB Y2K Organization Juan S. Jaliff, November '98
-

Lessons beyond 2000

Better Config. Management



- live documentation
- few program variants
- less custom code

Less System interdependency

- open architectures
- generic data communications

robustness

transparent maintenance

 ABB Y2K Organization Juan S. Jaliff, November '98 



Lessons beyond 2000

operators

requirements

vendors

platforms

 ABB Y2K Organization Juan S. Jaliff, November '98 

Making Industrial plants Y2K-ready: Concept and Experience at ABB

Juan S. Jaliff (ABB Corporate Research, Sweden)

Wolfgang Eixelsberger (ABB Corporate Research,
Norway)

Arne Iversen (ABB Industri AS, Norway)

Roland Revesjö (ABB Industrial Products, Sweden)

e-mail:

juan.jaliff@secrc.abb.se

wolfgang.eixelsberger@nocrc.abb.no

arne.iversen@noina.abb.no

roland.revesjo@seipr.mail.abb.com

Prepared for

INTERNATIONAL SOFTWARE

Quality Week Europe 1998

Copyright 1998 ABB

Permission to use, copy and distribute this document is hereby granted on the condition that each copy contains this copyright notice in its entirety and that no part of the documentation is used for commercial purposes but restricted to use for information purposes within an organization.



Introduction

Every day of the week, industrial plants throughout the world produce quality goods at a fast pace. Their operation through the turn of the millenium is critical to their income streams. In cases like commodity producers, it is also important to prevent production losses for customers further up the value-added chain. Utilities must ensure continuous operation in order to preclude risks to the functioning of society at large.

At the heart of these plants, a process control system is in charge. A multitude of processors, firmware and application software cooperate in a network of networks to control the production equipment and inform operators and managers about the status of the plant. ABB produces such process control systems and has installed them at a large number of customer sites around the world, along with other related plant systems (e.g. fire & gas and emergency shutdown systems for the oil & gas industry). The interesting question is now: how can we make sure that these complex process control systems will work properly into the year 2000?

Technical Background

The ABB family of products for industrial process control is Advant OCS (Open Control System). It is a DCS (Distributed Control System) capable of handling large amounts of input/output signals through a network of nodes, with the following main functions:

- Controller nodes acquire process signals and send outputs based on local process control blocks and/or operator action
- Operator Stations present process information from controller nodes in a graphical manner and enable operator actions on the process
- Information Management Stations store historical process information in a database system for later retrieval and analysis, either locally or as servers for external client systems

Communication among these nodes is carried by a process bus designed for real-time applications. Systems consisting of many nodes are usually divided into sub-networks, often reflecting the number of production lines and/or sections in the plant. Some plants have interfaces to non-ABB process control equipment, for example Programmable Logic Controllers



(PLCs) or application-specific high-level control systems (e.g. setpoints from production planning systems).

Y2K Problem dimensions

In each and every node, Y2K-related modes of failure are actually not different from those already identified for other types of information systems. The elementary error mechanism is failure to represent and/or store a year in more than 2 digits. Fortunately, year information is very rarely used to control the process, so misrepresentation of the year will most often not lead to errors in process control itself.

Nevertheless, problems can arise at the turn of the century as nodes running on hardware, operating systems or firmware that fails come to a halt and/or refuse to restart. Such nodes could theoretically bring the network to a halt, given certain conditions. A more subtle but nonetheless dangerous failure mode can arise from erroneous sorting of alarm and event logs. Operators could fail to react to alarms or do so based on the wrong process information.

In order to identify potential modes of failure for the whole system, it is useful to map the technical problem domain along two dimensions. The first one is a "vertical" dimension representing the hierarchical levels of a single control system node. Some of these are:

- customized application software
- standard application software
- operating system
- BIOS (as in PCs)
- real-time clock
-

At each and every one of these levels, date functions may be Y2K-compliant or not. In general, several versions of software, firmware or hardware exist for each level, and each combination must be analyzed independently regarding Y2K.

The second dimension is an interaction dimension, referring to the dynamic communication among the multitude of nodes in a network. Problems related to exchange of time-tagged data and synchronization of real-time processes could potentially arise as a result of Y2K.

This second dimension can become quite complex in the case of sites where the DCS interacts with other vendors' systems. Some examples of such systems in various industries are:



- fire alarm systems
- emergency shutdown systems
- PLC-based systems
- manufacturing execution systems

In-house Testing at ABB

The two aforementioned dimensions are basically independent from each other, and two types of Y2K testing can be done:

- a) Component testing (one node at a time)
- b) Communication and interaction testing (two or more nodes at a time)

ABB started early with Y2K testing at component-level, and made fast progress due to the fact that the Advant OCS product release policy had aimed at reducing the number of hierarchical version combinations at nodes in the field. Fast test progress in itself meant that tests (b) were carried out by and large on nodes which had passed (a). It was thus possible to do (b) as BCT (Big Configuration Tests) with large number of nodes, giving two side benefits:

- combinatorial reduction of the number of tests needed
- real-scale representation of the systems at customer sites

A typical BCT setup at ABB labs is shown on Fig. 1. Such a large configuration test represents quite well the conditions in the field, providing customers a good reference in their own Y2K analysis.

Field Testing

Decisions as to the scope of on-site testing required are up to each individual plant operator. It is not possible to establish general rules, but at least the following factors must be taken into account:

- safety relevance of the various systems
- production relevance of ditto
- cost of production losses
- access to technical staff for workarounds on site



Once these factors have been assessed, it is possible to identify which systems are critical for Y2K tests. The extent of on-site testing which is adequate must then be defined. Two factors affecting this are:

- number of data connections among different systems
- scope of Y2K in-house testing carried out by system vendors

ABB provides customers, at no cost, access to the results from its BCTs. The large configurations tested have shown Y2K compliance, once the intervening components have been updated to Y2K-compliant levels (most of the latter-year releases fulfill this). By comparing their own installations to those tested in-house, it is possible to reduce the extent of on-site testing.

Plant Readiness

The process of making a plant ready for the millenium shift is complex and involves organizational as well as technical issues. To start out, it is often difficult to find the as-is status of the plant documented with enough information as to enable Y2K analysis.

As-built documents must be revised with information from later projects where extensions, modifications and retrofitting of the plant have taken place. The external suppliers involved have not produced uniform documents. These will at any rate not readily be found in a central archive. More likely they will be spread throughout the organization, among project, purchasing, operation and other departments. At any rate, an exhaustive inventory of each plant must be performed, identifying:

- all date-sensitive systems
- all date-sensitive components
- versions of all software, firmware and hardware
- all data communication links (both internal and external)

ABB has already accumulated substantial experience in Y2K plant walkthroughs. Important early references include oil platforms in the North Sea (Shell, Amoco). In some complex plant cases, customers have contracted for the whole process of Y2K-readiness, even for non-ABB equipment. Whether subcontracting parts or all of the project, a plant operator will have to address the following activities:



- produce **inventory** of Y2K-sensitive equipment and systems
- obtain **vendor statements** of Y2K-compliance
- decide upon **upgrade paths** to compliant versions
- **procure/install** the compliant versions
- **update application software** accordingly
- identify internal/external **data links**
- carry out **on-site tests**

Software-specific Issues

The application software installed at an industrial plant will typically comprise a large number of source-code statements. Different programming languages will often be used for the various hierarchical levels already mentioned. In the case of the Advant OCS, controller nodes are programmed in a homogeneous proprietary language called AMPL.

The AMPL language is function-block-oriented. Function blocks, called PC-elements (for Process Control), are completely standardized and do not change from plant to plant. Applications for each plant are achieved by selection and parameter-setting of the appropriate elements. This turns out to be a great advantage for Y2K source code control, since date processing is limited to just five elements.

Standard usage of these elements involves dates with complete year and century information, and is strictly observed by ABB programmers. However, new applications can be made at the plant freely, and complete year information cannot be guaranteed a priori. Therefore, ABB has now developed automated software tools to scan AMPL code, identify usage of date-sensitive elements and document it. Manual analysis at these points is then used. Hardly any Y2K-non-compliant usage has been found so far.



Higher-level applications vary a lot from industry to industry. Some do not have any at all. In other cases, languages like C++, C, Pascal or Fortran are used. ABB has made an evaluation of different software tools available on the market to assist Y2K control of source code in such languages.

A very important conclusion from these controls is that use of standardized software at a plant reduces significantly the costs and time needed for Y2K controls. Plants which have implemented many ad hoc solutions might have substantial difficulties in finding a proper software upgrade path conducive to Y2K-compliant versions. This is yet another instance where large variance in version management impacts negatively upon software life cycle costs.

Y2K Concept at ABB

The main thrust behind ABB's approach to the Y2K problem has been to turn it into a win-win game, not a zero-sum one. In order to achieve this, ABB considers it essential to develop a partnership with the customer¹. Both vendor and customer have a lot to earn from a systematic walkthrough of a plant documenting the interdependencies among subsystems and their impact upon production. Some of the important factors for a successful Y2K project at a plant are:

- early planning between customer and vendor
- high-level management involvement
- active participation of plant operation/maintenance engineers

Plant engineers feel ownership for the various systems, know their as-is status and can identify the most appropriate time slot during or outside planned maintenance outages for the performance of tests on site. They are also in the best position to judge whether some workarounds might be more cost-effective. For example, a shutdown-and-reboot alternative could minimize production losses if carefully planned.

ABB has therefore developed a Y2K concept based on the four pillars below, to support its customers. It is worth mentioning that all of the work that is not plant-specific has been carried out at own expense. All customers have access to the results at no cost. Should they choose to

¹ Ragaller, K., "Adopting a Partnership with your Process Control Vendor," IQPC Conference, Amsterdam, July 14th, 1998



contract Y2K services for their plant from ABB, the regular engineering rates are applicable.

Pillar 1: Product Investigation

The first pillar is a thorough investigation of all ABB products. In some cases this involves the aforementioned component testing and BCT. Using the widely accepted BSI compliance definition², products are rated:

- compliant (have been tested and results documented)
- non-compliant (nature of non-compliance documented, as well as plans and time schedule to achieve compliance)
- not applicable (lack of date functions documented)

The results are documented on a large database available online to all ABB staff. The database also includes test procedures, contact persons, Y2K-responsibles, external links, etc. A global Y2K team has been appointed, reporting directly to ABB's executive management. They have designed the database and initiated actions worldwide at the product-responsible companies in the group. Part of the information is also made available to customers online through the World Wide Web.

Pillar 2: Plant Inventory

The second pillar is the establishment of an inventory of ABB products installed at all plants. This information is requested from operators on as-available basis. Its main purpose is scheduling of service engineering staff, training courses and production planning for product upgrades (computer boards, PROMs, distribution kits, etc.).

Customers returning detailed version and release information will be automatically provided with product status (see above). Others will be offered help to find the relevant information.

Pillar 3: Pilot Projects

Some 50-odd customers were asked to participate in pilot projects, trying to cover all major plant types and world areas. The amount of site testing is discussed with them on a case-by-case basis, identifying how mission-critical each subsystem is and when it can be put offline. Results to date are very encouraging. The systematic approach, which includes external communication interfaces, can also be applied to other vendors'

² <http://www.bsi.org.uk/disc/year2000/2000.html>



equipment. Some customers have requested ABB to take overall responsibility for multivendor system examination and testing.

Pillar 4: Training

It is most important to train both customer and ABB engineers. The experience from the pilot projects is being rolled out to other plants through training seminars and workshops. Hands-on training is also very important. One way to achieve this is by letting ABB staff from other world regions participate as observers in the ongoing pilot projects.

Cost and Manpower Aspects

Time and cost estimation is an essential activity in the planning of a Y2K plant readiness project. Specific and widely accepted methods for estimation of time and costs for Y2K audits are currently not available. Historical data, which is in most cases the base for cost estimation, is often not available. Empirical estimation models can therefore not be used.

The Gartner Group³ estimates the costs for making applications Y2K compliant, as \$1.10 per executable line of code (LOC). Such information is however too general to be of help for specific applications. It is necessary to have a closer look at the different factors influencing the costs to formalize the cost and labor estimation process.

Y2K audits are relatively complex activities with many factors that must be taken into consideration. Some of these are:

- Direct/Indirect costs – Direct costs are costs that can be assigned to a specific product or installation. Indirect costs are costs that can be assigned to a set of products or family of systems (ABB does not pass these on to customers).
- Inventory list – Especially for installations made a number of years ago, the inventory may be incomplete or not up-to-date. Producing complete and correct inventory lists can be a time-consuming and costly process. Missing information is typically the release/version number.

³ Cassell, J., Schick, K., Hall, B., Phelps, J. "Management Edge: Year 2000 - Top View." The Gartner Group, Stamford, CT, 1997



- Maturity of method – Y2K test methods contain a set of procedures and checklists and provide information on how to use these tools for performing a Y2K audit. A mature testing method along with experienced testers from pilot projects helps ABB significantly reduce the effort for testing systems.
- Access to installation – On-site Y2K tests must be performed since installations simulating the system under consideration are often not available. The system must be in a safe state allowing the testers to perform Y2K-relevant actions as e.g. changing the system date.
- Third-party equipment – real-time and embedded systems are permanently exchanging data with the environment and other systems. Interfaces to other systems must be checked for existence of date-related information and the influence on the system under consideration. In most cases, it is also necessary to ask vendors of third party systems for Y2K compliance certification documentation.
- Certification process – effort depends on the test method and available tools. Manual code inspection may be very time-consuming. On-site costs depend on how many dates will be tested and if time changes are automatically propagated or not.
- Compliance effort – Specific actions must be carried out to make systems Y2K compliant. Such actions may be change of hardware or upgrade of software to compliant releases. System tests and additional certification processes may be necessary to make the system Y2K compliant.

These factors cannot be seen as independent but depend on and influence each other. ABB has accumulated enough experience from pilot projects to make reliable estimates of total costs to the customer. The pricing policy adopted is such that customers do not find it constraining in any way. The bottlenecks are actually access to test engineers and to plant systems at the right time.

Conclusions from Experience

We will try to capture here some of the lessons learned from the Y2K process at ABB and the pilot projects so far, in the hope that operators of industrial plants might benefit.



(Way) Before the Year 2000

Some friendly advice, in a nutshell:

- make an inventory of your plant and ask your vendors for Y2K statements ASAP
- identify when your system can be down for tests/upgrades (probably just a couple of times before 2000)
- analyze carefully the consequences of upgrades (e. g. new operating system versions might be required; how does this impact the rest of the system?)
- order your upgrades right away (typical delivery times are 5-7 weeks for firmware/hardware; might get longer close to 2000)
- analyze your vendors' in-house tests (so you can shorten your site tests)
- plan for some workarounds (can save time and money)
- establish standby system support for New Millennium evening (for example, ABB will have staff on call)

Beyond 2000

Y2K projects all over the world will certainly teach many lessons regarding how to develop software systems that are more robust and transparent to maintain. Both vendors and operators should learn. Some conclusions can already be drawn from experience to date:

Configuration management:

- up-to-date documentation of installed system status must be kept
- the number of software variants on site should be reduced
- applications unique to a site should be exceptions, not the rule
- operators will select vendors who can meet their requirements with mainstream software versions
- operators will work with vendors who put their specific future needs in today's development plans

System interdependency:

- the number of independent software architectures on site should be reduced
- ad hoc data communication links should not be used



- industry-standard data communication models should be promoted

Remarks

The authors hope to have shed light on some practical aspects of making the software systems at industrial plants ready for the year 2000. The information was intended to be non-partisan. Readers are welcome to contact them at the e-mail addresses provided.

Grateful acknowledgement is made of the material made available by Messrs. Klaus Ragaller, Håkan Bergman and Gunnar Liveborn, all with the ABB global Y2K organization.

ALL INFORMATION IN THIS DOCUMENT IS PROVIDED AS IS WITHOUT ANY REPRESENTATION OR WARRANTY OF ANY KIND EITHER EXPRESS OR IMPLIED INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES FOR MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. ANY ABB DOCUMENTATION MAY INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES AND ADDITIONS MAY BE MADE BY ABB FROM TIME TO TIME TO ANY INFORMATION CONTAINED HEREIN.



Standard communication TCP/IP

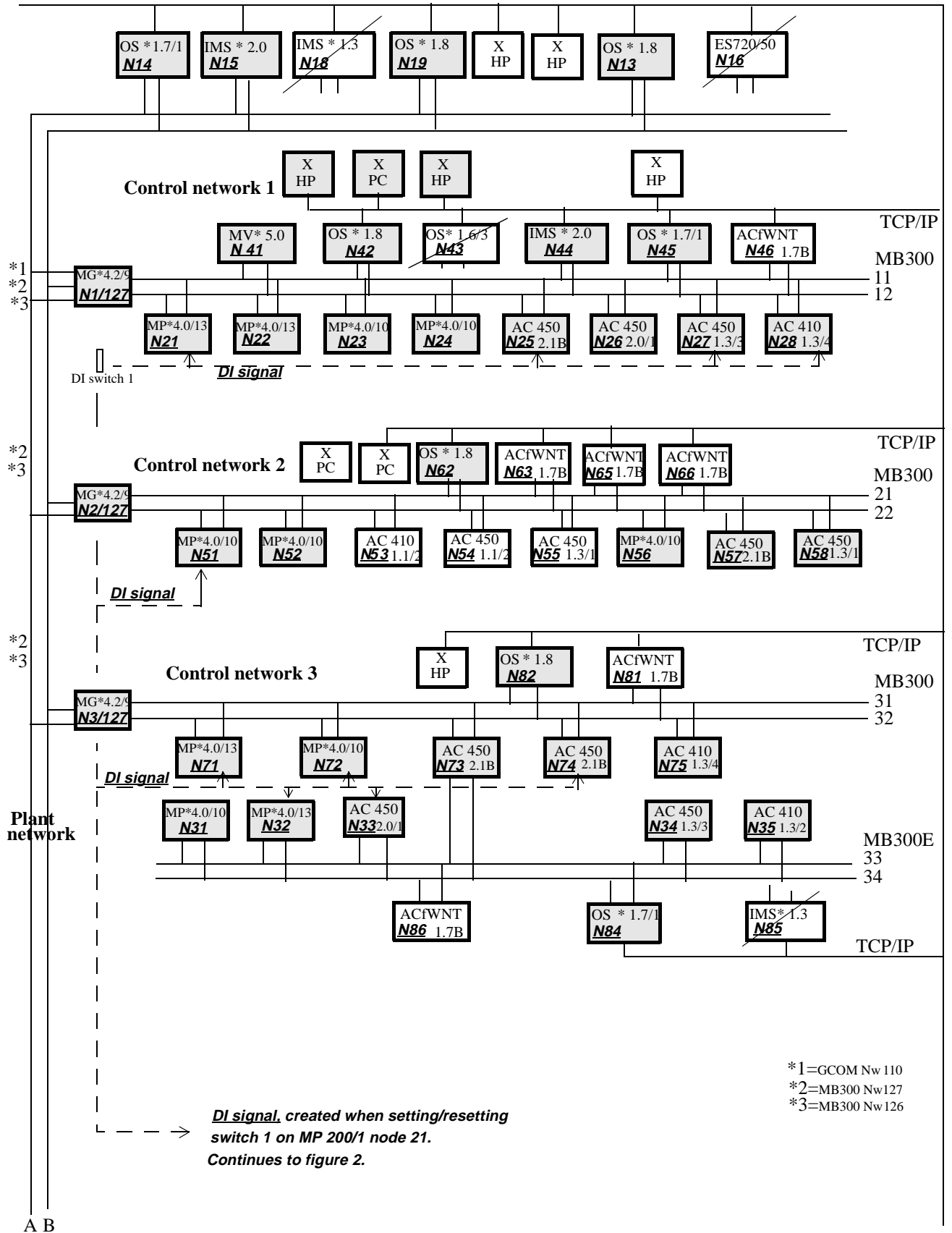


Figure 1 Test phase 1 & 3: The configuration used continues on next page

3BSE001270D00002/G



ABB Automation and Drives

Document number
3BSE015370

Lang. Rev. ind. Sheet
en - 11

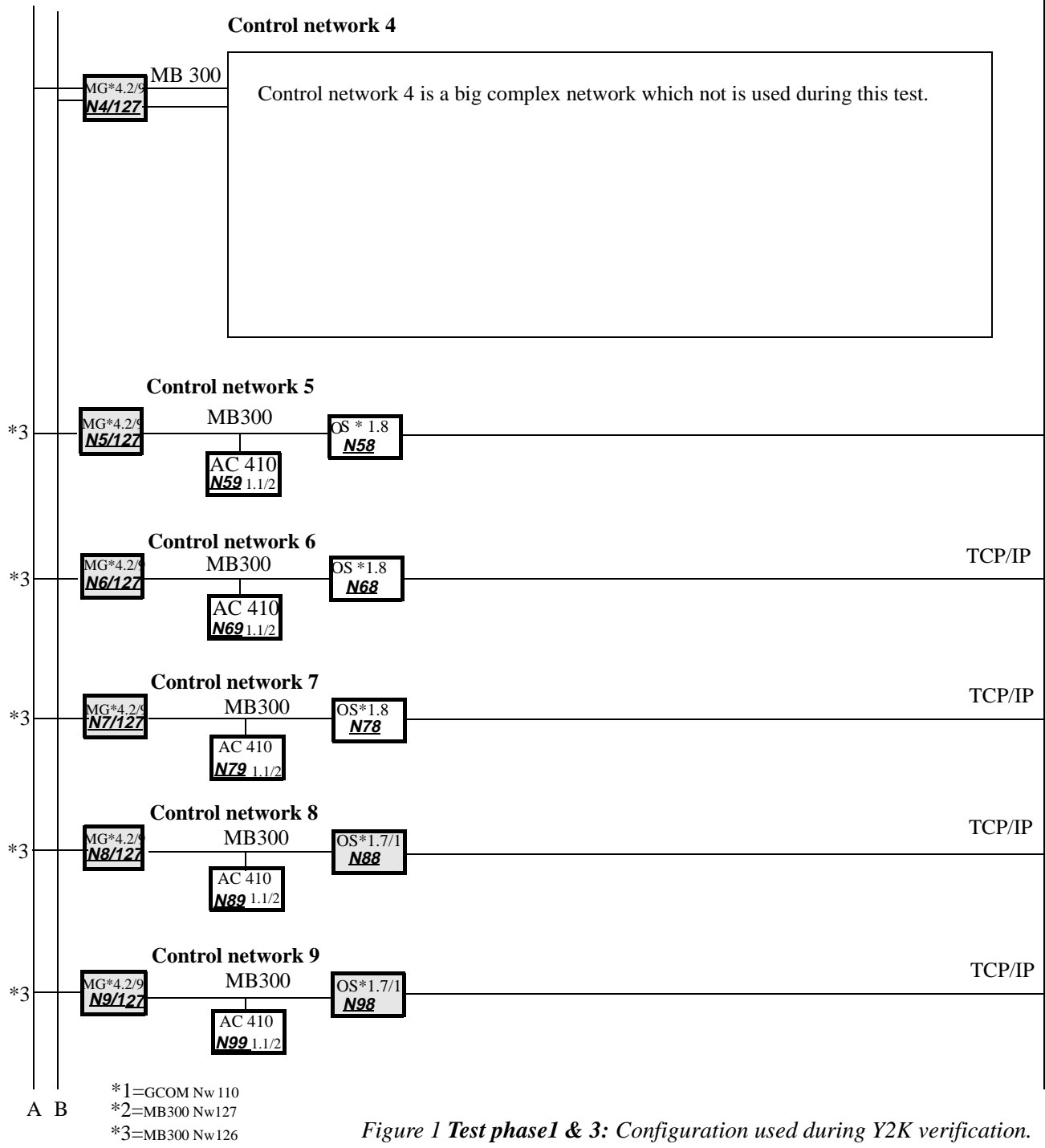



Figure 1 Test phase1 & 3: Configuration used during Y2K verification.

3BSE001270D00002/G

1




Risk Based Testing

**Risk Analysis Fundamentals for software testing,
Test Process Improvement: Plan and Manage,
Save Time and Money**

By Ståle Amland
Avenir (UK) Ltd., Reading, Great Britain
2nd International Software Quality Week Europe '98
9-13 November 1998 in Brussels, Belgium

Risk Based Testing




2

Presentation Outline

- ◆ The Challenge - *Time and Money*
- ◆ The Strategy - *Risk Based Approach*
- ◆ The Risk Analysis - *practise and theory*
- ◆ The Process - *planning, tracking and Estimating to Complete*
- ◆ Automated Testing - *recommendations*

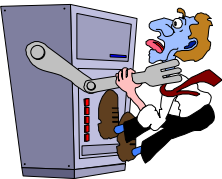
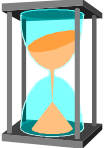
Risk Based Testing




3

The Challenge - Time and Money


- ◆ Time Constraints
- ◆ Resource Constraints
- ◆ Risk factors:
 - New technology
 - New environment
 - “Green Beans”
 - Large project



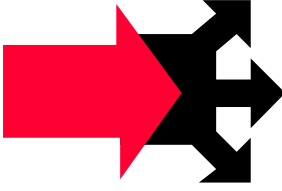
Risk Based Testing 


4

The Challenge - New Strategy

 What to do?

1. **Strategy:** Define a new risk based test approach
2. Perform a ***risk analysis***
3. Improve the system test ***process*** and develop the ***organisation***
4. Implement ***automated testing*** (optional)



Risk Based Testing 

1. Strategy → 2. Risk anal. 3. Process improv. 4. Auto. testing

The Strategy

5

- ◆ Risk Based Approach to testing:
 - Performing a Risk Analysis to identify areas of high risk
 - All functionality should be tested to a “minimum level”
 - “Extra testing” performed in identified high risk areas

Risk Based Testing *Avenir*

1. Strategy → 2. Risk anal. 3. Process improv. 4. Auto. testing

Risk Analysis and Testing

6


```
graph TD; TP[Test Plan] --> RS[Risk Strategy]; TI[Test Item Tree] --> RI[Risk Identification]; RI --> RA[Risk Assessment]; M[Matrix: Cost and Probability] --> RA; RA --> RM[Risk Mitigation]; RA --> TR[Testing, Inspection etc.]; RM --> RR[Risk Reporting]; TR --> RR; RR --> RP[Risk Prediction]; RP --> TM[Test Metrics]; RP --> RA; RR --> TP;
```

Risk Based Testing *Avenir*

1. Strategy
➔
2. Risk anal.
➔
3. Process improv.
➔
4. Auto. testing

7

The Risk Analysis - Theory




◆ The Formula

$$R(f) = P(f) * \sum (C(c) + C(v))$$

- R(f) - Calculated risk of function f
- P(f) - Probability of a fault in function f
- C(c) - Customer's cost related to a fault in function f
- C(v) - Vendor's cost related to a fault in func. f

Risk Based Testing




1. Strategy
➔
2. Risk anal.
➔
3. Process improv.
➔
4. Auto. testing

8

Risk Analysis - practise


Prior to test execution:
identify critical transactions

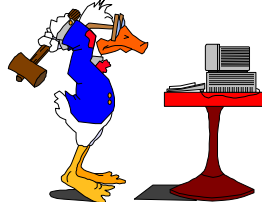
①



Test Execution identifies
"bad" transactions

②






Extra Testing:

- Additional testing by product specialist
- Automated regression testing

③

Risk Based Testing



1. Strategy

2. Risk anal.

3. Process improv.

4. Auto. testing

The Risk Analysis - analytical

9

◆ Ranking the functions based on

- The Cost of an Error
- The Probability of an Error

$$R(f) = P(f) * \frac{C(c) + C(m)}{2}$$

◆ Example:

Func.	Cost				Probability						Risk of func. R(f)
	C(m)	C(c)	Avr. C	Norm. C	New Func. 5	Design Quality 5	Size 1	Compl. 3	Weight Avrg. 7,75	Probability P(f) 0,74	
Close Acct.	1	3	2	0,67	2	2	2	3	7,75	0,74	0,50

Risk Based Testing

1. Strategy

2. Risk anal.

3. Process improv.

4. Auto. testing

Risk Reporting

10

Risk Based Testing

11

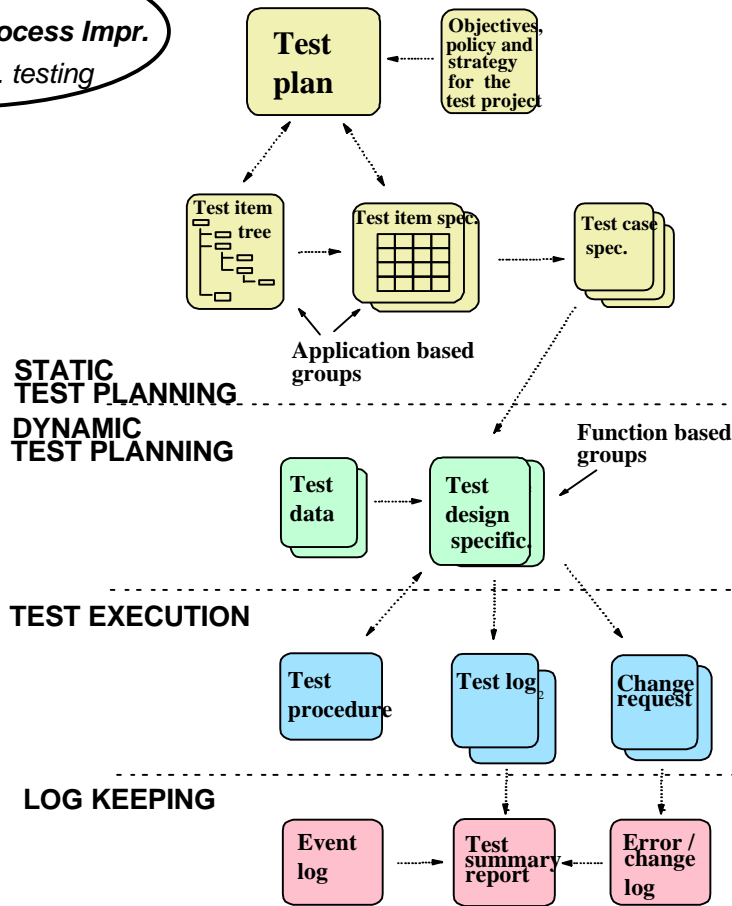
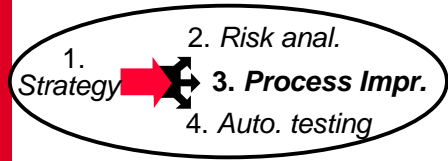
The Process

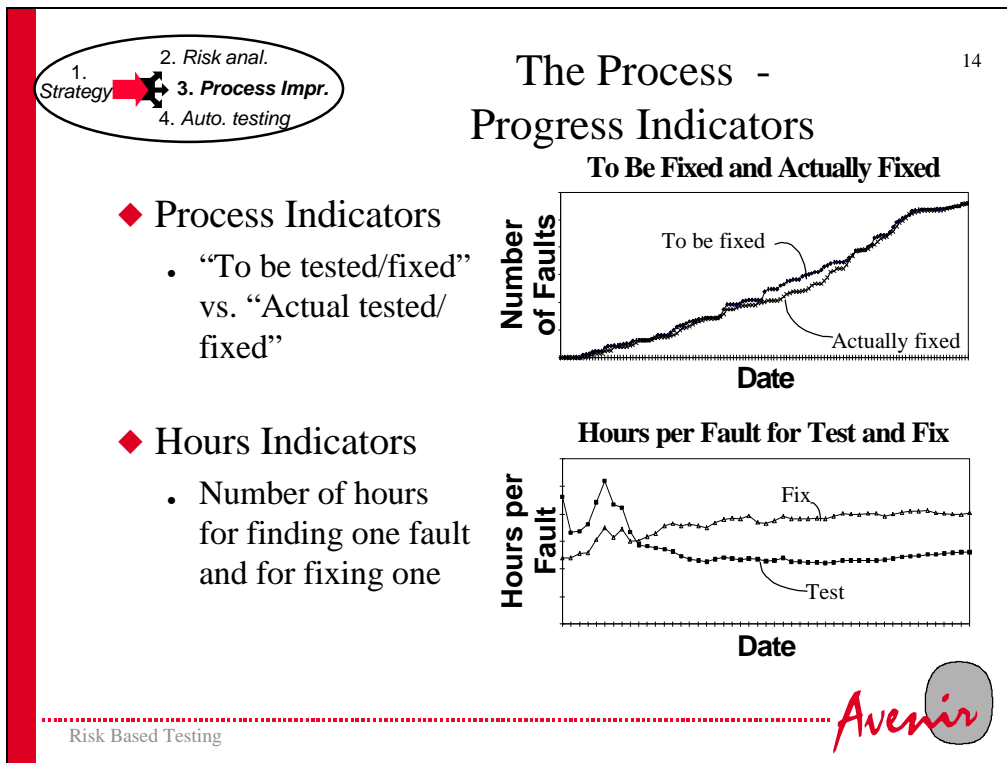
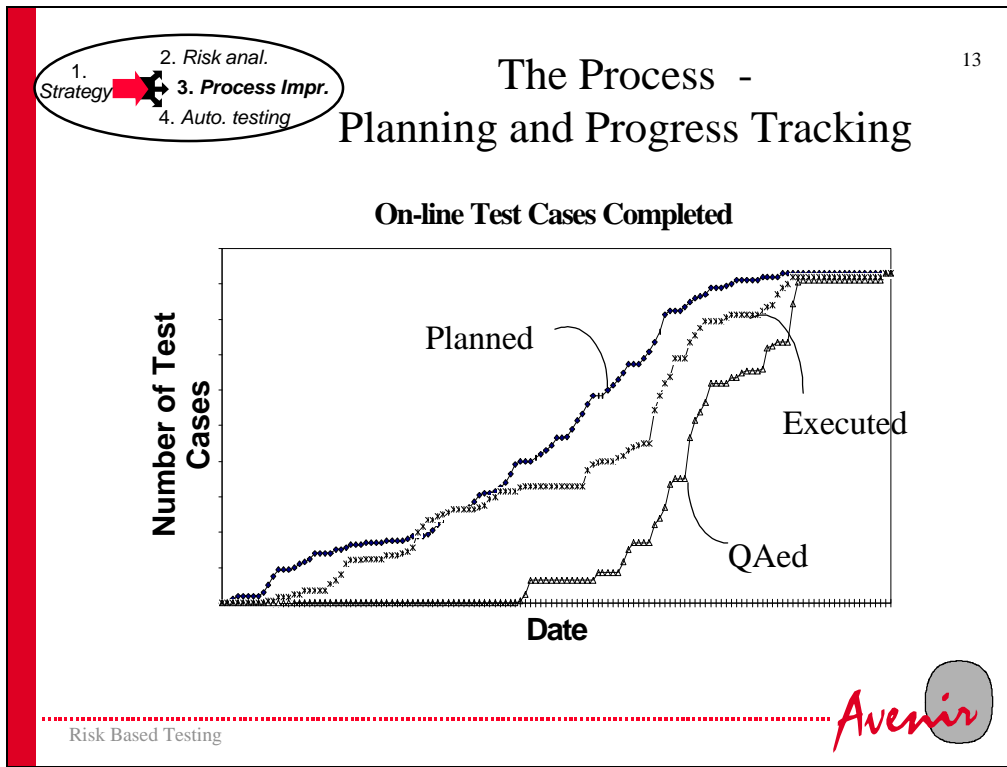
1. Strategy → 2. Risk anal. → 3. Process Impr. → 4. Auto. testing

- ◆ Limited time and resources require well defined:
 - Interfaces to design, construction and unit test stages
 - Deliverables with defined quality standards
 - Test preparation and execution procedures stating entry and exit criteria for each phase
 - Control procedures to handle scope and issues
 - Organisation and responsibility
 - Progress Planning and Tracking with ETC indicators

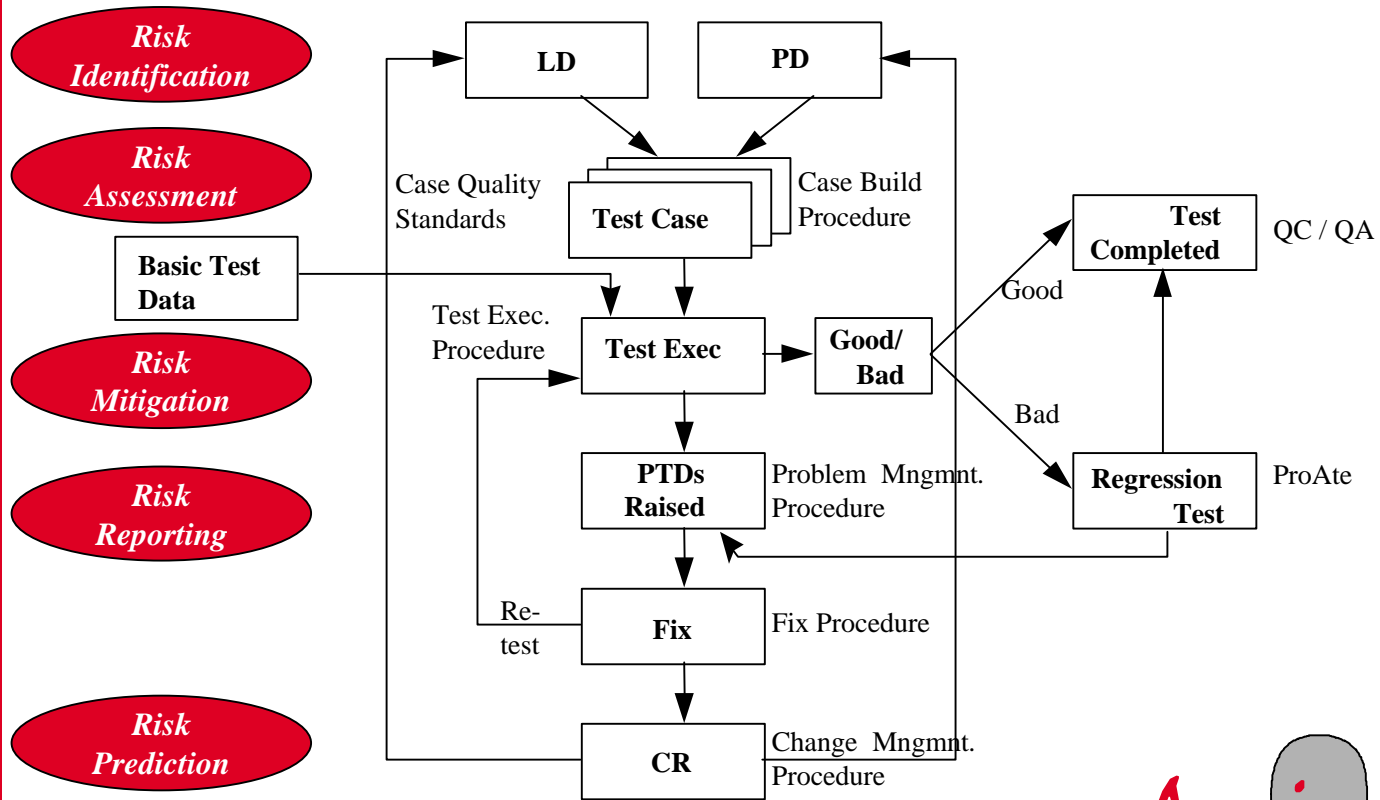
Risk Based Testing

Avenir





Test Process Work Flow



- Risk Identification
- Risk Assessment
- Basic Test Data
- Risk Mitigation
- Risk Reporting
- Risk Prediction

Risk Based Testing



1. Strategy

2. Risk anal.


3. Process Impr.

4. Auto. testing

16

Test Automation

- ◆ Identify “minimum level of testing”
 - Priority 1: everything must be tested
 - Priority 2: test automation
- ◆ Do not automate “everything”!
 - Identify most critical area / function
 - Identify “Top-20” for automatic testing first
- ◆ Start with “simple” tests - e.g. user interface
- ◆ Identify “simple” automated tests you will benefit from, i.e. save time and money!
- ◆ Performance testing useful to Automate in AT

Risk Based Testing


1. Strategy

2. Risk anal.


3. Process Impr.

4. Auto. testing

17

Summary

- ◆ New Strategy: Risk Based Approach
 - Focused Testing
 - Reduced Resources
 - Improved Quality
- ◆ Process and Organisation must support the new strategy
- ◆ Test Automation
 - Most critical areas
 - Performance testing

Risk Based Testing


Contact Details

18

**Ståle Amland
Avenir (UK) Ltd.
Beacontree Plaza, Gillette Way,
Reading, RG2 0BS, Great Britain**

Phone: +44 118 9757286 FAX: +44 118 9866161

E-mail: sa@avenir.co.uk



Risk Based Testing

Risk Analysis Fundamentals for software testing including a Financial Application case study

Stef Amland
Avenir (UK) Ltd.
Beacontree Plaza, Gillette Way,
Reading, RG2 0BS, Great Britain

Phone: +44 118 9757286 FAX: +44 118 9866161
E-mail: sa@avenir.co.uk

Abstract

This paper provides an overview of risk analysis fundamentals, focusing on software testing with the key objectives of reducing the cost of the project test phase and reducing future potential production costs by optimising the test process. The phases of Risk Identification, Risk Strategy, Risk Assessment, Risk Mitigation (Reduction) and Risk Prediction are discussed. Of particular interest is the use of indicators to identify the probability and the consequences of individual risks (errors) if they occur.

The body of this paper contains a case study of the system test stage of a project to develop a very flexible retail banking application with complex test requirements. The project required a methodology that would identify functions in their system where the consequence of a fault would be most costly (either to the vendor's customers or to the vendor) and also a technique to identify those functions with the highest probability of faults.

A risk analysis was performed and the functions with the highest risk, in terms of probability and cost, were identified. A risk based approach to testing was introduced, i.e. during testing resources would be focused in those areas representing the highest risk. To support this approach, a well defined but flexible, test organisation was developed.

The test process was strengthened and well defined control procedures were implemented. The level of test documentation produced prior to test execution was kept to a minimum and as a result, more responsibility was passed to the individual tester. To support this approach, good progress tracking was essential to show the actual progress made and to calculate the resources required to complete the test activities.

1. Introduction

The risk based approach to testing is explained in five sections:

1. **Risk Analysis Fundamentals:** Chapter 2 contains a brief introduction to risk analysis in general with particular focus on using risk analysis to improve the software test process.
2. **The Case:** Chapter 3 is the first chapter of the case study. It explains the background of how the methodology was implemented in one particular project
3. **The Challenge:** Chapters 4 and 5 further summarise what had to be done in the case project, why it should be done and how it should be done.
4. **The Risk Analysis:** Chapter 6 explains how the probability of a fault and the cost of a fault was identified and how the risk of a given function was calculated to identify the most important functions as an input into the test process.
5. **The Process and Organisation:** Chapter 7 goes through the test process and discusses improvements made to the process and to the organisation to support the risk based approach to testing in the case project.

In addition, chapter 8 briefly discusses the importance of automated testing as part of a risk based approach. Some areas for further research and of general interest are listed in chapter 9.

2. Risk Analysis fundamentals in software testing

This chapter provides a high level overview of risk analysis fundamentals and is only intended to be a basic introduction to the topic. Each of the activities described in this chapter are expanded upon as part of the included case study.

According to Webster's New World Dictionary, risk is the chance of injury, damage or loss; dangerous

The objective of Risk Analysis is to identify potential problems that could affect the cost or outcome of the project.

The objective of risk assessment is to take control over the potential problems before the problems control you, and remember: "prevention is always better than the cure."

The following figure shows the activities involved in risk analysis. Each activity will be further discussed below.

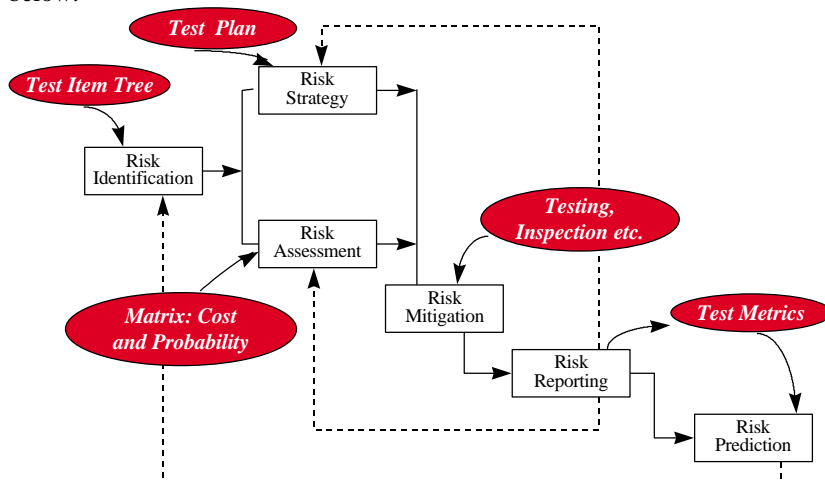


Figure 1: Risk analysis activity model. This model is taken from Karolak's book 'Software Engineering Risk Management'[6] with some additions made (the oval boxes) to show how this activity model fits in with the test process.

2.1 Risk Identification

The activity of identifying risk answers these questions:

- Is there risk to this function?
- How can it be classified?

Risk identification involves collecting information about the development project and classifying it to determine the amount of potential risk in the test phase and in production (in the future).

The risk could be related to system complexity (i.e. embedded systems or distributed systems), new technology or methodology involved that could cause problems, limited business knowledge or poor design and code quality.

2.2 Risk Strategy

Risk based strategizing and planning involves the identification and assessment of risks and the development of contingency plans for possible alternative project activity or the mitigation of all risks. These plans are then used to direct the management of risks during the software testing activities. It is therefore possible to define an appropriate level of testing per function based on the risk assessment of the function. This approach also allows for additional testing to be defined for functions that are critical or are identified as high risk as a result of testing (due to poor design, quality, documentation, etc.).

2.3 Risk Assessment

Assessing risks means determining the effects (including costs) of potential risks. Risk assessments involves asking questions such as: Is this a risk or not? How serious is the risk? What are the consequences? What is the likelihood of this risk happening? Decisions are made based on the risk being assessed. The decision(s) may be to mitigate, manage or ignore.

The important things to identify (and quantify) are:

- What indicators can be used to predict the probability of a failure?
The important thing is to identify what is important to the quality of this function. That could be design quality (e.g. how many change requests had to be raised), program size, complexity, programmers skills etc.
- What are the consequences if this particular function fails?
Very often is it impossible to quantify this accurate, but using low-medium-high (1-2-3) might be good enough to rank the individual functions.

By combining the consequence and the probability (from risk identification above) it should now be possible to rank the individual functions of a system. The ranking could be done based on "experience" or by empirical calculations. Examples of both are shown in the case study later in this paper.

2.4 Risk Mitigation

The activity of mitigating and avoiding risks is based on information gained from the previous activities of identifying, planning, and assessing risks. Risk mitigation/avoidance activities avoid risks or minimise their impact.

The idea is to use inspection and/or focus testing on the critical functions to minimise the impact a failure in this function will have in production.

2.5 Risk Reporting

Risk reporting is based on information obtained from the previous topics (those of identifying, planning, assessing, and mitigating risks).

Risk reporting is very often done in a standard graph like the following:

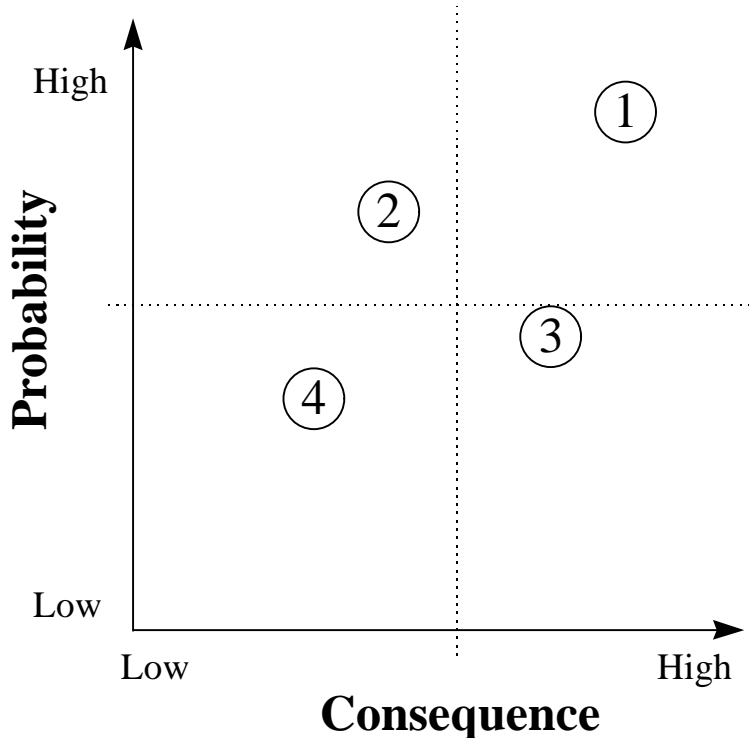


Figure 2: Standard risk reporting - concentrate on those in the upper right corner!

In the test phase it is important to monitor the number of errors found, number of errors per function, classification of errors, number of hours testing per error, number of hours in fixing per errors etc. The test metrics are discussed in detail in the case study later in this paper.

2.6 Risk Prediction

Risk prediction is derived from the previous activities of identifying, planning, assessing, mitigating, and reporting risks. Risk prediction involves forecasting risks using the history and knowledge of previously identified risks.

During test execution it is important to monitor the quality of each individual function (number of errors found), and to add additional testing or even reject the function and send it back to development if the quality is unacceptable. This is an ongoing activity throughout the test phase.

3. The Case

The rest of this paper will discuss a case study using the risk based approach to software testing, relating the different activities to the activity model discussed in the previous chapter.

3.1 The Application

This paper is based on the system test stage of a project developing a retail banking application. The project included an upgrade of a Customer Information System being used by clients as a central customer, account and product database, and a complete reengineering of a Deposit Management System. The project scope included reengineering of the data model, technology change from IMS/DL1 to CICS/DB2, rewrite from JSP COBOL to COBOL-2 and a completely new physical design. During this rewrite large investments were done in productivity tools, design, quality assurance and testing.

The project started in June 1994 and was delivered in October 1995. The project total was approximately 40 man years over 17 months. This paper documents experiences from the system test stage, which consumed approximately 22% of the total project resources.

The applications consist of approximately 300 on-line transactions and 300 batch programs, a total of 730.000 SLOC¹ and 187 dB2 tables. This is the server part only, no client-GUI was tested in this project.

3.2 The Scope

The system test stage included:

1. **Technical System Test.** I.e. what is usually referred to as environment test and integration test. Due to differences between the development environment and the production environment, the system test stage had to test all programs in the production environment. During system test the test team had to do the integration test of the on-line system by testing and documenting all on-line interfaces (called modules). The team also had to do the integration test of the batch system by testing and documenting that all modules had been called and also testing the complete batch flow.
2. **Functional System Test.** I.e. black box testing of all programs and modules to detect any discrepancies between the behaviour of the system and its specifications. The integration test verified that all modules had been called, and that the functional system test was designed based on application functionality.
3. **Non-functional System Test.** The system test also tested the non-functional requirements, i.e. security, performance (volume- and stress-test), configuration (application consistency), backup and recovery procedures and documentation (system, operation and installation documentation).

As for all projects, the time and resources were limited. At the beginning of construction (programming), the system test strategy was still not agreed upon. Since the development project was a very large project to the vendor and therefore consumed nearly all available resources, the number of people with experience available for test planning was limited.

The final system test strategy for the system test was agreed approximately one month before end of construction, and the time for planning was extremely short. A traditional approach to system test planning based on test preparation done in parallel with design and construction, could therefore not be used.

The following project stages were executed before the system test²:

- **Project Initiation - PI** (organising the project, staffing and development environment)
- **Requirement Analysis - RA** (documents the functional requirements to the application)
- **Logical Design - LD** (data model and process model)

¹ SLOC = Source Line of Code, excluding comments

² The methodology was based on LBMS'Systems Engineering, see ref. Systems Engineering.

- **Physical Design - PD** (program design - executed as part of construction)
- **Construction and Unit Test - CUT** (programming and testing, including a 100% code coverage test)

4. The Challenge

Why did the vendor need a Risk Based Approach to the System Test?

Because:

- **The Available Calendar Time was limited.** The development project had a very short time frame. The construction and unit test stage was also delayed, so the system test had to become even shorter! The applications are very flexible, and therefore very complicated to test. The calendar time did not allow for a thorough testing of all functions. Focus had to be put on those areas representing the largest risk if a fault occurred. The vendor needed a methodology to identify the most critical areas to test.
- **There were limited available resources before the end of construction!** Midway through construction a limited number of senior analysts were available for system test preparation, one for on-line and one for batch in addition to management. The estimates to build all identified test scripts were 8 - 10 people in each of the teams (on-line and batch) until system test start! A test methodology had to be developed based on limited resources during test preparation and using most resources during test execution.
- **There were several risk factors.** Due to strict quality control of the unit test (including 100% code coverage requirements), the modules were expected to be of good quality when entering system test.

However, a lot of other factors indicated trouble:

- The project utilised a new development environment for programming, debugging and unit test based on Microfocus COBOL Workbench on PC, running CICS with OS/2. The exact effect of the learning curve on the programmer's productivity by starting with a tool all new to the organisation, was unknown.
- The development environment for on-line proved to be fairly good. However, the JCL-support for large batch streams was poor. Therefore, batch integration tests between units were not extensive and represented a risk to the integration test part of the system test.
- The system test was exclusively executed on the IBM mainframe after the source code was transferred. A test bench was developed on the IBM mainframe, executing transactions as a "dumb-terminal" client. This represented a risk since The vendor did not have any experience of the difference between the PC development environment and the mainframe environment. Differences in SQL-implementation were discovered during the project.
- Because of the vendor's expansion, a lot of the programmers were new to the company and though well educated, most of them were new to the application area and to the technology being used.
- The number of people involved was high (approximately 50 at peak) and the development time was short (planned to 17 months), this was one of the largest projects done by the vendor ever. Even though the vendor has been conducting large projects in the past, available experience from projects with this size in a compressed time frame, was limited.

What did the vendor do?

- **The System Test Strategy** document had to be rewritten. The customer did receive the preliminary version of the strategy explaining a "traditional well documented test" with everything documented prior to test execution. We had to convince the customer that this new approach was "as good as the original one, except that the new one was feasible, given the calendar time and resources available." The System Test Strategy would define the "minimum level of testing" including level of documentation for all functions and identify how a Risk Analysis would be used to identify functions to be focused on during test execution.
- We had to perform a **Risk Analysis** to identify the most critical areas both to the customer and to the vendor. A methodology had to be identified and implemented.
- **The System Test Process and Organisation** had to be improved, or even "optimised." This included defining the test process by preparing procedures, planning the test, controlling the process, progress tracking, and defining roles and responsibilities. The vendor had to convince the customer about the

feasibility of the new strategy and prove the quality of the process and the product to be delivered. To document the test progress and to communicate this to the customer, became a key issue.

- **Automated Testing** was part of the contract. Initially we intended to use automated testing for all on-line functions. As the resources and time became very limited, automated testing became part of the risk based strategy, i.e. it was used to handle those on-line transactions with most faults.

The rest of this paper will document the implementation of the risk based strategy by the vendor, showing required changes to the system test process and the organisation.

5. The Strategy

The project started with a **Traditional Approach to testing**, i.e. the test should be prepared with input and output as part of the design and construction stages, prior to system test start. However, it was obvious as time passed by and only limited resources were available to prepare the System Test, that this strategy was impossible to fulfil.

The original system test strategy document (based on a traditional test approach), identified the following test structure for both on-line and batch testing:

1. **System Test Plan**, documenting the test scope, environment and deliverables, test control procedures, test tools to be used, test schedule and phases, and listing start and stop criteria related to each phase.
2. **Test Specification**, i.e. a detailed break down of the application into testable units.
3. **Test Cases**, i.e. documentation of what to test, basically listing all requirements enabling a tester to easily read them.
4. **Test Scripts**, i.e. documentation of how to test 'step by step,' including test data to be used by the tester.

Implementing a structure like the one above is very time consuming, especially step 4 - documenting test scripts.

Midway through construction it became obvious that it was impossible to document everything before end of construction. Either the project would be delayed, or the test planning process had to be improved.

The main problem at this stage was the preliminary system test strategy document delivered to the customer. How do you have the customer accept that you will not be able to document all tests prior to test execution as thoroughly as you originally intended to? By convincing him that the new process will improve the product quality!

The key words became **Risk Based Approach** to testing. We agreed with the customer (*reference to the risk activity model in chapter 2 is given in italic*):

1. The vendor will test all functionality in the application to 'a minimum level' (in addition to all interfaces, and all non-functional tests). This will not be documented prior to the test, but logging of details for all tests (i.e. input, expected output and actual output), will after test execution, prove this 'minimum level of **Risk Strategy**).
2. All test cases ('what to test') will be documented prior to test start and will be available for the customer to review (**Risk Strategy**).
3. Only **highly qualified testers**, i.e. system analysts experienced in the application area, will be utilised for testing, and the testers will be responsible for planning all 'test shots,' including providing test data and documenting the executed tests. (Tools were available to the tester for documenting the tests) (**Risk Strategy**).
4. The vendor will do a risk analysis together with the customer to identify those areas of highest risk, either to the customer or to the vendor (**Risk Identification and Risk Assessment**).
5. Based on the Risk Analysis, the vendor will focus 'extra testing' in those areas of highest risk (**Risk Mitigation**).

6. Extra testing will be planned and performed by a specialists in the application area, that are not involved (**Risk Mitigation and Risk Reporting**).

The 6 bullet points above cover all activities from Risk Identification through Risk Reporting. How risk reporting was used as input to risk prediction is explained later.

The customer approved the idea, and the vendor was ready to start. The project was now in a hurry and had to do the following:

1. Complete the documentation of all test cases (what to test) for on-line and batch
2. Perform the Risk Analysis for on-line and batch and plan any extra testing
3. Document the new risk based test process, including procedures, check lists, training the testers and preparing the test organisation.

6. The Risk Analysis

(Risk Identification, Risk Strategy and Risk Assessment)

The risk analysis was performed prior to system test start, but was continuously updated during test execution. Separate analysis was performed for on-line and batch.

The vendor developed a model for calculating the risk based on:

- the probability of an error (in the on-line transaction or batch program), and
- the cost (consequence) of an error in the corresponding function, both to the vendor and the customer (in production).

Similar methodologies have been documented by others, see Østedal and Ståhane [1992]. However, this paper explains a practical implementation of the methodology.

There are three main sources to the Risk Analysis:

1. **Quality of the function** (area) to be tested, i.e. quality of a program or a module. This was used as an indication of the probability of a fault - $P(f)$ ³. The assumption is that a function suffering from poor design, inexperienced programmer, complex functionality etc. is more exposed to faults than functions based on better design quality, more experienced programmer etc.
2. **The consequences of a fault in the function as seen by the customer** in a production situation, i.e. probability of a legal threat, losing market place, not fulfilling government regulations etc. because of faults. This consequence represents a cost to the customer - $C(c)$.
3. **The consequences of a fault in the function as seen by the vendor**, i.e. probability of negative publicity, high software maintenance cost etc. because of a function with faults. This consequence represents a cost to the vendor - $C(v)$.

The assumption was that the cost to the customer is equally important in the risk analysis to the cost of the vendor, and the calculated Risk of a function $R(f)$ would then be:

$$R(f) = P(f) * \frac{C(c) + C(v)}{2}$$

An example of areas with different risk profiles is **area of interest calculation** and **area of printing internal reports**. A fault in interest calculation could easily end up with a legal threat for the bank using the software, while a fault in the printing of internal reports not had to be known by the public at all.

³ According to B. Beizer: **fault** - incorrect program or data object; a bug, **error** - incorrect behavior resulting from a fault; a symptom. See Beizer [1990].

6.1 Risk Analysis - On-line

At the time the on-line risk analysis was performed very little information was still available from the construction teams. The process was very simple, however not compromising the software quality and satisfying to the customer.

On-line Risk Analysis steps:

1. **Prior to System Test:** The vendor asked the customer to set up a Top-20 list of on-line transactions which they viewed as the most critical transactions. All transactions on that list would go through extra testing.
2. **Throughout System Test:** The vendor would add to the list of critical transactions based on the number of faults found, those transactions were also subjects for extra testing. The number of faults could vary due to specification and design quality, programmers lack of knowledge in the application area, lack of knowledge about the development / production environment etc.

Extra Testing consisted of two elements:

1. **Additional testing by product specialist.** A separate checklist was developed, and the exit criteria for the transaction to pass were at least one hour of continuous testing without any errors detected. All transactions on the customer's Top-20 list and all transactions with totally more than 10 faults (the bad ones) found in the system test by the vendor, would go into this process. The total of 10 faults could include faults detected because of environment problems, configuration problems in the build process, functional faults and errors in the documentation.
2. **Regression Testing Executed in an Automated Test Tool.** All transactions with more than 4 functional errors would go through full regression testing after last bug-fix was proved to be correct. The regression test would replay all tests executed by the tester during the test execution process, using the test tool AutoTester, see reference AutoTester.

6.2 Risk Analysis - Batch

The risk analysis of the batch areas was done when construction was midway through. The project had now gained substantial information about the application area and the test and production environment. The quality in the design stages had also been proved by implementation.

The application's batch areas were split into sub-areas (or functions) as part of the logical design stage. Those functions were now used in the risk analysis.

A sample list of batch functions is

- Interest Calculation
- Penalty Calculation
- Profitability Analysis
- Delete of Data
- Reporting
- Capitalisation

According to the formula:

$$R(f) = P(f) * \frac{C(c) + C(v)}{2}$$

the challenge was to identify the cost of a fault and to identify indicators of quality, i.e. what is the probability of a fault in this function?

We used a very simple, though satisfying method to develop these two elements, i.e. cost of a fault and probability of a fault, for all batch functions. The vendor invited the project management, product specialists, designers, programmers, people working with application maintenance and test management into a meeting. This team had to agree on the cost of a fault and the probability of a fault for each function.

1. **The cost of a fault** was indicated by a number from 1 to 3 where 1 represented minimum cost in case of a fault in this function. Elements to be considered were:
 - Maintenance resource to allocate if a fault occurred during customer's production (given the vendor would provide 24-hours service).
 - Legal consequences by not fulfilling government requirements.
 - Consequences of a bad reputation."
2. **The Probability of a fault** was indicated by giving 4 indicators a number varying from 1 to 3, where 1 was good," i.e. the probability of a fault was low. The indicators were:
 - **Changed or New Functionality.** The project was a reengineering of an existing application and the change of functionality varied from function to function. The programs had to be rewritten anyway, but if the functionality was not changed, at least there was an exact specification of the function.
 - **Design Quality** would vary, depending on the function (some functions were all new), and by design and application experience of the designer. This was measured by counting number of Change Requests to the design (for further explanation of the project process, see section The Process).
 - **Size.** We assumed that the number of sub-functions within a function would affect the number of faults introduced by the programmer.
 - **Complexity** (logical). The programmer's ability to understand the function he was programming will usually effect the number of faults.

For the cost related to each batch function we used the average of the Customer's Cost $C(c)$, and the Vendor's Cost $C(v)$.

The indicators used to calculate the Probability of a fault for a particular function $P(f)$, were weighted, i.e. the weight would vary from 1 to 5, rating 5 as the most important indicator of a function with poor quality.

The weights used by The vendor were:

1. Changed or New Functionality - 5
2. Design Quality - 5
3. Size - 1
4. Complexity - 3

An example of calculated risk for the batch function 'Close Account' is shown in the figure below. The Risk $R(f)$ was calculated for all batch functions and the list was sorted to identify those areas to be focused during testing. The Probability $P(f)$ is calculated as the Weighted Average of a particular function divided by the highest Weighted Average of all functions, giving the probability in the range $[0,1]$.

Func.	Cost			Probability						Risk of funct. R(f)
	C(v)	C(c)	Avg. C	New Func. 5	Design Quality 5	Size 1	Compl. 3	Weight Avg. 7,75	Proba- bility P(f) 0,74	
Close Accnt.	1	3	2	2	2	2	3	7,75	0,74	1,48

Figure 3: Example of calculated Risk for the batch function "Close Account". The vendor's cost of a fault is low (1), but the customer's cost is supposed to be high (3). The average cost is then 2 (Avg. C). The probability is calculated by calculating the weighted average (Weight Avg.) divided by the highest Weighted Average for all functions (which in this example was 7,00), giving a probability in the range [0,1]. The Risk R(f) is the multiplication of the Average Cost and the Probability.

7. The Process

(Risk Strategy, Risk Mitigation, Risk Reporting and Risk Prediction)

The limited time and resources available made it very important to have a well defined test process. This includes:

- Interfaces to design, construction and unit test stages
- Deliverables with defined quality standards
- Test preparation and execution procedures stating entry and exit criteria for each test phase
- Control procedures to handle scope changes and issues
- Well defined organisation and responsibility including training of the testers
- Progress Tracking

7.1 Interface to Design and Construction and Unit Test

The overall test process was based on Logical Design (LD) and Physical Design (PD) giving the base for the build of the Test cases. Those test cases had to adhere to defined quality standards. The test cases were given to the tester together with the Test Execution Procedure. The tester would prepare test data and report any problems during test execution. The problems would be documented in a Problem Tracking Document and would be passed on to the test team leader for verification before going to the fix team leader.

We called it a "problem" document because it could be any kind of problem, including program faults, but also environmental problems, test data problems, change requests etc.

After the problem was fixed the problem tracking document was passed back to the test team for re-test. If the test passed, the function would be evaluated as "Good" or "Bad," based on number of errors discovered in this function. If this function did include a high number of faults, then extra testing would be applied, either by building a complete set of regression test scripts for the test tool AutoTester⁴ or by additional manual testing. Finally, the quality of the function would be evaluated as "good," and the test result would go through extensive quality control (QC), i.e. checking completeness and accuracy and finally verifying all formalities in a quality assurance process (QA).

A Fix might also include a Change Request (CR) to either the Logical or Physical Design. The test team leader would typically originate the CR for Logical Design whilst the fix team leader would typically originate the CR for Physical Design.

⁴ The program AutoTester from AutoTester Inc., see reference AutoTester.

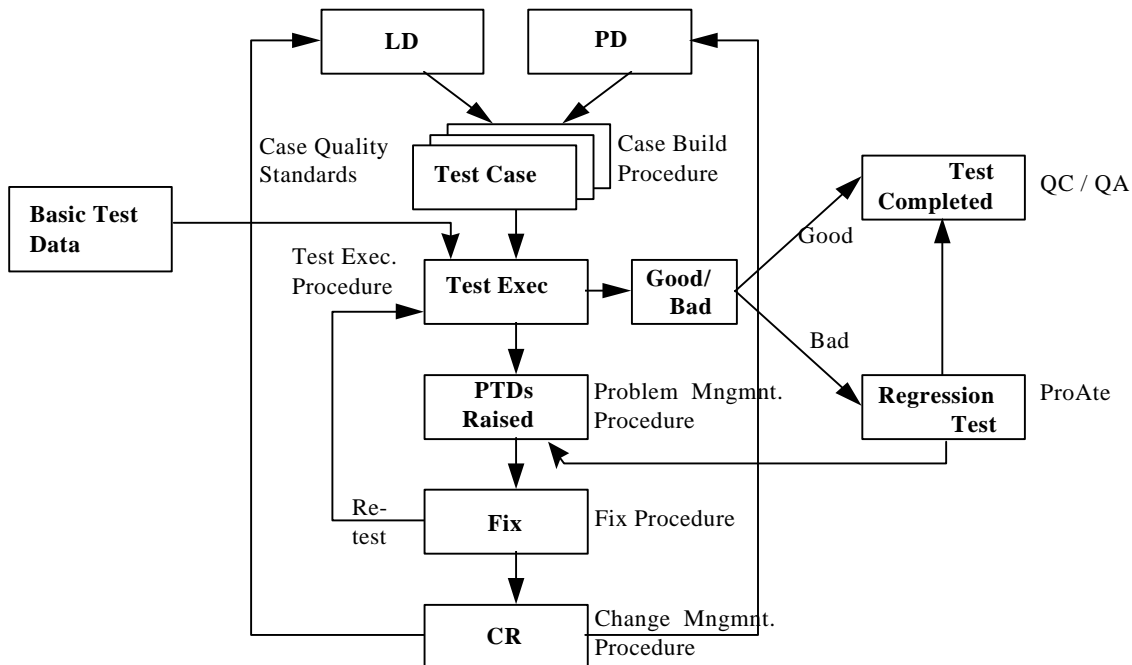


Figure 4: This figure shows the work flow of the System Test Process, from design documents to test case, from test case to test execution and then a problem might occur. When the problem is fixed and re-tested it might be accepted by the QC/QA procedures. After the testing is completed, the function might go through regression testing, depending on number of faults detected. For on-line the regression testing is performed in ProAte, i.e. The vendor AutoTester Environment (see footnote 4).

The risk based approach was based on the assumption that the tester himself prepared the test script including test data and that he documented the test result. This activity was important to the success of the risk based approach since the approach puts more responsibility on the tester than a traditional approach.

7.2 Documentation Structure and Deliverables

Prior to test execution the following documents were prepared:

1. **The System Test Plan**, including scope, roles and responsibilities, deliverables with level of detail, procedures overview and test schedule.
2. **The System Test Specification**, giving a system break down for testing, i.e. identifying all functions being tested.
3. **The System Test Cases**, for each function in the test specification the test case would identify **what** functionality to test by listing each validation rule to verify, all combinations of input data to verify etc. The test case would typically state "Test a future date in the Posting Date field," not including the actual date to be used in the test.

During the on-line tests the tester executed test shots and documented them in test scripts with input data used, expected result, and test result. A test tool provided the tester with logging capability to document the input data being used, logging the expected result and logging the actual result. The tools forced the tester to log the expected result prior to logging input data and actual result. This made the testers prepare their test shots carefully. Again, this methodology of testing and documenting and the tool were important for the risk based approach success, i.e. the methodology forced the tester to be prepared and the tool supported in the documentation process.

The idea was to use the same structure for batch testing, in which we did not succeed. Due to the fact that a batch run takes more time to execute and therefore the number of retries is limited, the batch test scripts including test data and expected result, had to be completed before the processing of a batch cycle.

The test structure is visualised in the figure below.

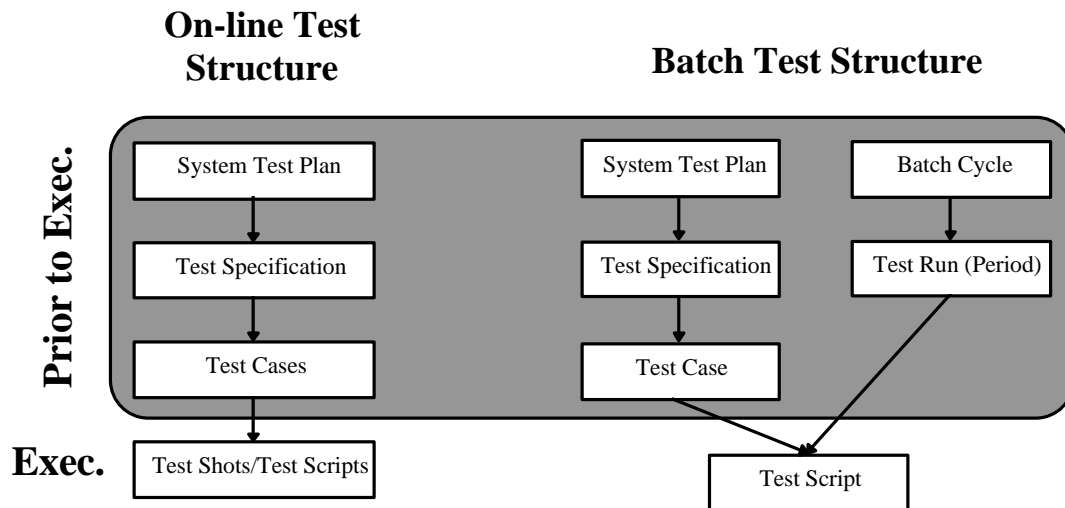


Figure 5: System Test Documentation Structure, on-line and batch, as implemented according to the risk based approach. The plan, specification and test cases were developed prior to test execution but the test scripts / shots were developed and documented during test execution. In the batch test the test scripts were more complete prior to test execution than in on-line. Batch preparation included also design of test runs (single processing days) and batch cycles (a series of test runs, e.g. daily run, month end and year end).

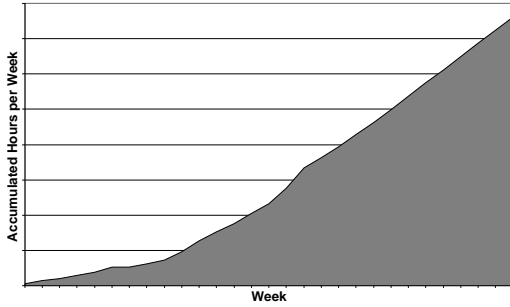
This test structure allowed for a delayed resources profile by utilising highly qualified testers during test execution and a limited number of qualified persons in test preparation. This was absolutely necessary since there were only limited resources available until end of construction.

7.3 The Organisation and the Test Planning

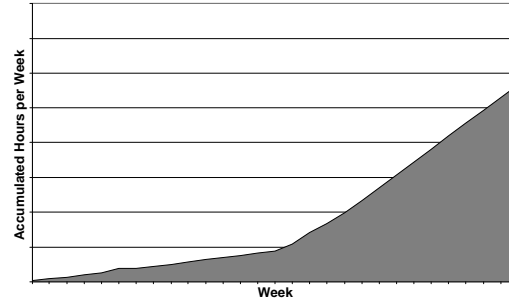
The resource profiles below show accumulated resource profiles for:

1. Estimated resource requirements for a traditional approach (**Original Estimate**)
2. Estimated resource requirements for a risk based approach (**Risk Based Estimate**)
3. Actual resource usage, based on the selected Risk Based Approach (**Actual**)

Original Estimate



Risk Based Estimate



Actual

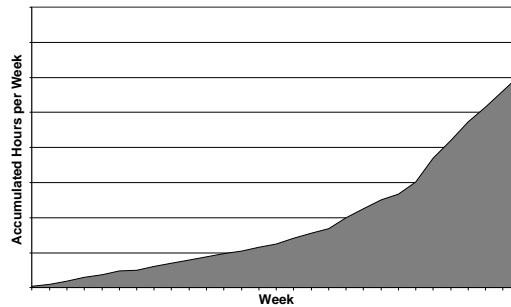


Figure 6: Resource profiles for Original Estimate (i.e. Traditional Approach), Risk Based Estimate (Risk Based Approach) and Actual (i.e. actual accumulated number of hours spent).

The graphs show that the risk based approach consumed less resources relative to the original estimate based on a traditional test approach.

The risk based test approach is highly dependent on using qualified testers, i.e. testers with experience within the application area and preferable with experience within the test environment. The reason is that the tester himself will build the actual test scripts during test execution, including test data. It is obvious that inexperienced testers will need a lot of training to be productive with this approach. However, training must be executed to make even the qualified testers familiar with the new test approach. The vendor executed several pilot projects, both in the on-line and the batch area to verify the test methodology, the accuracy of procedures and to train the testers. This was done through the short test planning phase.

Another criteria for success for the risk based approach was an efficient, dynamic and flexible test organisation. Over time, it proved to be essential that the testers (i.e. test team leader) do prioritise. Whenever there were discussions about which faults to correct first, the decision had to be based on the risk analysis and what functions needed to be tested most, not which fault was most convenient for the programmers to fix.

The organisation shown below proved to be efficient at supporting the Risk Based Approach.

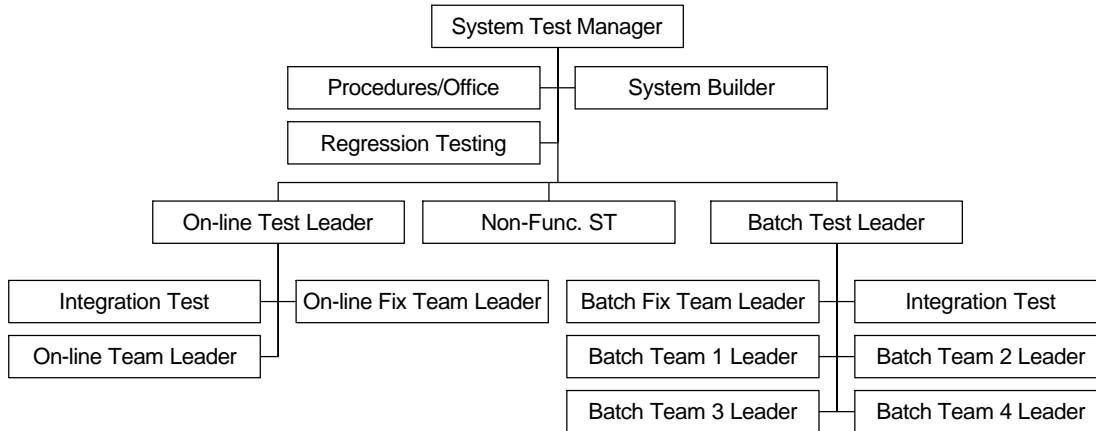


Figure 7: The Risk Based Approach to testing requires a flexible organisation, focused on fixing bugs related to critical functions.

What made this organisation efficient was the combination of a fixed high level structure combined with flexibility on the detailed level.

The fixed high level structure is represented by:

- **Centralised services.** The follow-up on procedures and production of test documentation was taken care of by the project office (Procedures / Office in the chart). Similar, the configuration control was taken care of by one role - the System Builder, and the automated regression testing was executed by one Regression Testing team.
- **Centralised Progress tracking.** Progress tracking was only done by the system test manager, although all individuals recorded their own progress every day.
- **Separate teams for on-line, batch and non-functional testing.** The documentation requirements and the test execution procedures were quite different for all three teams, and the organisation had benefits from not training the testers in more than one area.
- **Well defined responsibility.** The team leaders were responsible for the preparation of all test cases prior to test execution and also for reviewing the result after the test. In addition there was a quality control function to assure completeness and accuracy, and finally a quality assurance of the formalities.

The flexibility on the detailed level is represented by:

- **Shared resource pool for testers and fixers.** This is not represented in the graph above, but was very flexible. E.g. instead of the test team waiting for fixes to be implemented, they would participate in the fix process, and vice versa. This would also improve the testers knowledge about the system, and finally improve the testing quality.
- **Shared team management,** i.e. the on-line team leader would be the same person as the on-line test leader, and the batch team leader would be the same as the batch test team leader. This is done to assure that the focus is on testing (not on fixing), and the prioritisation of which modules to fix is done based on the risk analysis and not on which modules are easiest to fix."

7.4 The Control Procedures

The project implemented separate procedures for control issues (i.e. changes to scope, process or schedule not related to system design) and change requests (i.e. changes related to system design and implementation). All projects will be affected by issues and change requests, but the risk based approach made this project even more vulnerable to changes. The reason is that the planning process had been limited and the detail test scripts were not prepared until test execution. Usually several faults are identified during the test planning phase, but by the risk based approach to testing, the planning phase is part of the test execution.

All change requests had to be accepted by the product manager outside the system test team.

7.5 Progress Tracking and Progress Indicators

(Risk Reporting and Risk Prediction)

The introduction of the risk based approach made it very important to document the test progress to the customer as well as the quality of the test. The quality of the test was documented as part of the output documentation from the test execution, including the complete listing of the test log from the test tool.

The progress tracking was critical to have the customer believe in the product and to believe in the end date. The reporting included basically two elements:

1. The number of tests started and completed according to plan.
2. Indicators to show the load of faults found and corrected.

7.5.1 On-line Progress Tracking

The following graphs show the on-line tests started and completed. Because of the limited material prepared prior to test execution, the quality control of the test documentation prior to execution was very limited. This made the quality control (QC) and quality assurance (QA) processes during test execution even more critical. Therefore, the curve most interesting to the customer was tests actually completed from QA in the graph On-line Test Cases Completed.

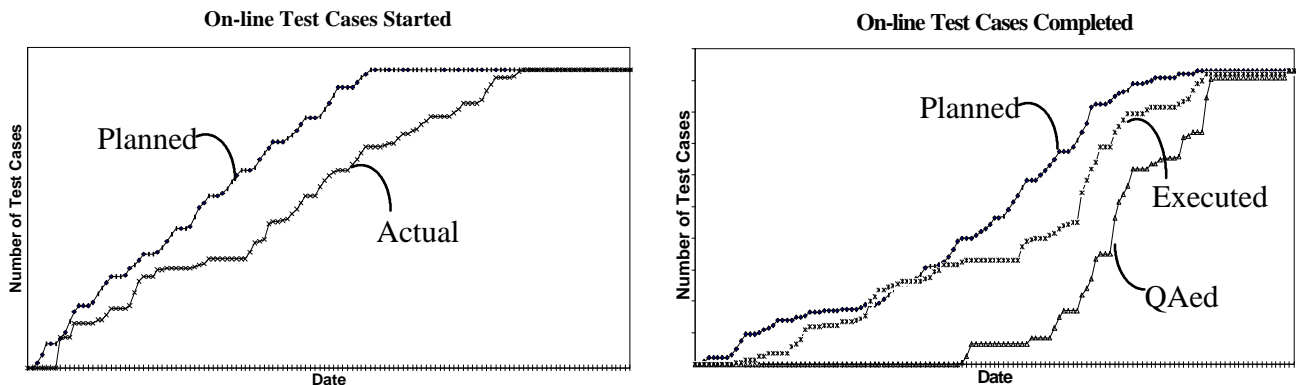


Figure 8: Progress Tracking. To the left is a graph showing planned and actual of test cases started. The graph to the right visualises test cases completed, showing planned complete, actually completed by tester (executed) and actually completed from QA.

7.5.2 Batch Progress Tracking

The batch process of ProDeposits is very complex. Approximately 300 batch programs constitute a daily batch run. A traditional approach to batch testing for The vendor would have been to set up one batch system and process day by day, fixing problems as they occurred.

The approach was to run as many test runs as possible as early as possible to identify problem areas (i.e. areas of high risk) and to focus the test in those areas.

The consequence was that 3 sub-systems were set up, each with a batch cycle consisting of 12-15 batch runs, i.e. processing 12-15 periods. A period (point of time) is a single day, a week end, a month end, a year end etc. Each batch cycle would be processed at least 3 times for all three systems during the system test period.

If possible, the complete batch cycle was completed, not waiting for fixes of faults identified in one batch run before continuing. The result was an early detection of problem areas and the possibility of focusing the test.

Because of the planned strategy to run all batch cycles three times per system, the number of tests started did not make any sense in batch testing as it did in on-line. The result was that the plan for batch testing was calculated as the total number of batch tests (i.e. number of verifies⁵ to be executed), evenly spread over the total number of days for batch testing. As a result of this, the progress did not look good in the beginning when a lot of outstanding integration tests were executed. However, after a while the progress improved, and during the last few weeks the graphs showed a steady slope.

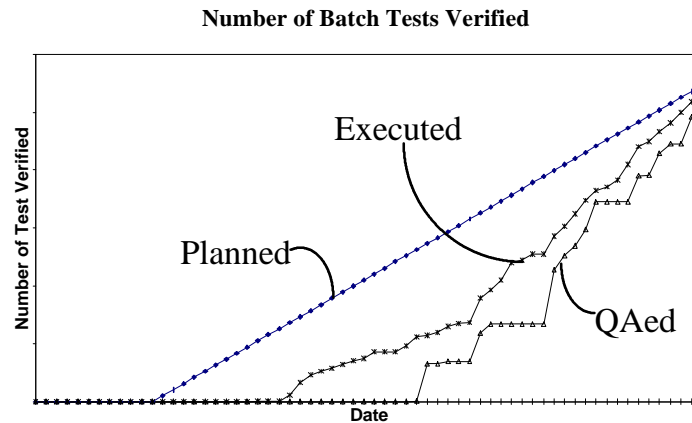


Figure 9: Progress Tracking Batch. The plan was calculated based on number of verifies to be executed and number of days of testing. The graph also shows number of verifies actually executed and number of tests QAed by date.

The graph above gave the customer a snap shot of the current situation. In addition we needed some indicators that provided the customer with good visibility of, and therefore confidence in, the test process.

7.5.3 Progress Indicators

We used two indicators, one related to the test process and one related to the fix process. The first one showed number of faults reported to the fix team and number of faults fixed (i.e. reported back to the test team). The second indicator showed number of faults reported back to the test team for re-test from the fixers and number of faults re-tested. The second indicator also included a graph showing number of fixes from the fix team being rejected by the testers as part of the re-test.

⁵ A verify document is a document with the expected result of a particular batch test. The document will list all results to look for after a test run. The number of expected results can vary depending of type of test. To execute a verify's to compare the actual result with the expected result after a test run.

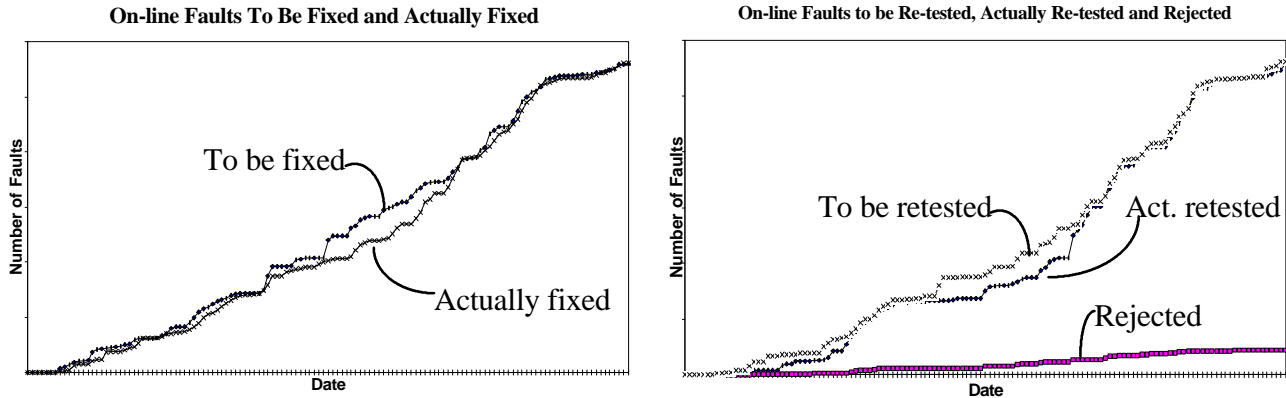


Figure 10: Progress Indicators. The left graph shows the number of faults delivered to the fix team and number of faults fixed. The graph at the right shows the number of reported faults that have been fixed and returned to re-test (to be re-tested) and the number of faults actually re-tested. The lower curve is the number of fixes being rejected in re-test.

Similar graphs to the above were developed for batch faults.

7.5.4 Estimated To Complete (ETC)

To calculation of ETC for test projects is always complex, and even more complicated when the preparation work is as limited as in this project. Again, the need for indicators to predict the number of resources needed to meet the end date, was essential to the test approach chosen.

We closely monitored the number of hours spent in testing and in fixing related to the number of faults identified and fixed. The following graphs were used for both, on-line and batch.

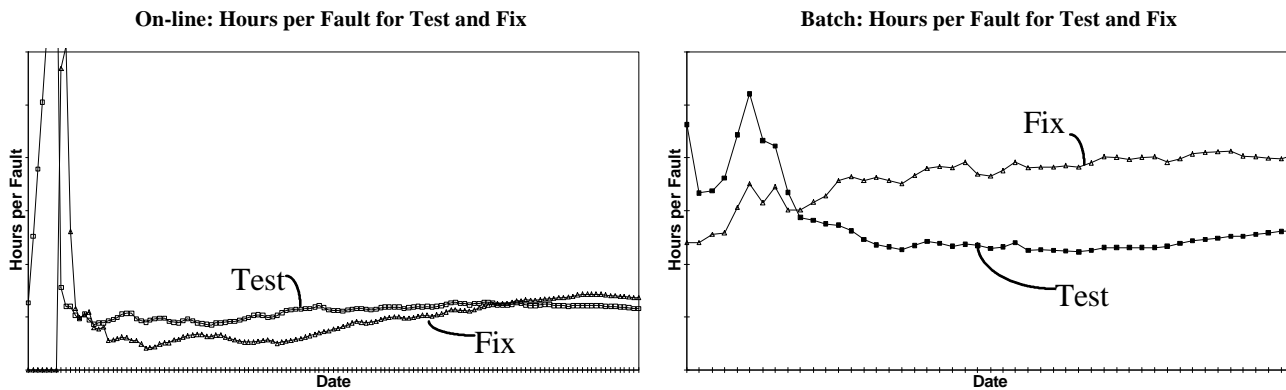


Figure 11: Estimated to Complete. The number of hours testing per fault found and number of hours analysis / programming per fault fixed were used as indicators to calculate ETC.

In addition to the number of hours per fault the following numbers were used for calculating the ETC for on-line:

1. Number of faults found per on-line transaction (i.e. per on-line test case)
2. Number of fixes being rejected (i.e. generating a new fault to be sent to fixing and re-test)
3. Number of remaining on-line test cases

By combining 1, 2 and 3 above, the remaining number of faults could be estimated, and by using the numbers from figure above, the total resource requirements could be estimated.

For batch the calculation method was somewhat different. In addition to the number of hours per fault found, the numbers used were:

1. Number of faults found per Verify document.
2. Number of fixes being rejected (i.e. generating a new fault to be sent to fixing and re-test)
3. Number of verify documents being accepted out of total number of Verify documents reviewed
4. Number of verify documents still to be verified.

The result graphs for on-line and batch are shown in the following figure. The rising curve is the accumulated hours spent and the falling is the calculated ETC over time. It took some weeks before the ETC-calculations were reliable, but they proved to be very accurate during the last few weeks. If more historical data could have gone in to the ETC calculation, a reliable result could have been provided at an earlier stage.

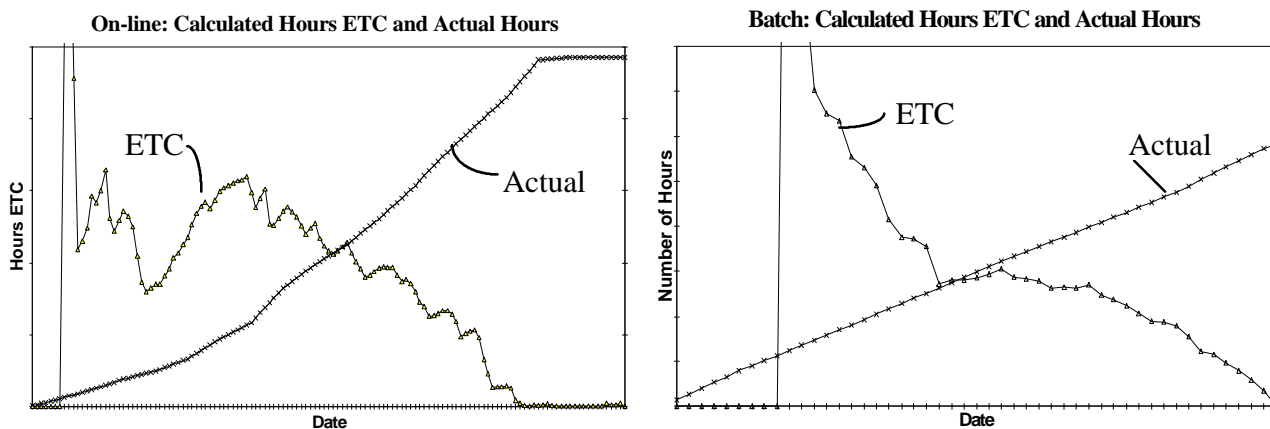


Figure 12: Calculated ETC and Actual hours spent for on-line and batch.

8. Automated Testing

(Risk Strategy and Risk Mitigation)

The project was committed to use automated regression testing by utilising the tool AutoTester from AutoTester Inc. This proved to be a commitment very hard to fulfill. Also, originally the intention was to develop all AutoTester test scripts prior to test execution.

Due to the changed test approach, the information required to develop AutoTester scripts was not available, i.e. the test data and the scripts would be provided by the tester during test execution.

The Risk Based approach was based on each tester using AutoTester to log all test shots and to record test scripts for automated regression testing. This proved to be very complicated because:

1. The tester had to think of regression testing all through test execution. This included to plan the test data, the sequence of transactions etc.
2. All testers shared the same database. They could easily damage each others test data if not paying attention.

The project used 25% of the total test resources for on-line, in automated regression testing. The regression test team managed to regression test 15% of all on-line transactions, and found 2.5% of all faults.

The recommendation to the next project will be:

1. Let the manual testers focus on doing manual tests, using a tool for documentation / recording without thinking automation. The result should be readable, not re-playable. The tester should be able to set up his own test data within his limits.
2. Set up a separate database for automated regression testing.
3. Select the worst transactions for automated regression testing.
4. Identify a separate test team to focus on automated testing.
5. Do not start recording for automated regression testing until the function is stable, i.e. most faults have been identified and fixed.
6. Over time develop a lifetime test script for all transactions, i.e. an automated test script to be used as an

9. Further Research

As a test the McCabe complexity, see McCabe [1976], was checked for a random list of the 15 on-line transactions with the highest number of faults identified and 15 on-line transactions with the lowest number of faults identified. The result showed that the McCabe complexity in average is 100% higher for those with a high number of faults than for those with a low number of faults.

This material however, needs more investigation. Particularly interesting is the analysis of the function's logical design to be able to identify functions with a potential of a large number of faults, at an early stage.

A process improvement project has been started by the vendor to continue some of the ideas discussed in this paper.

10. References

- | No | Reference |
|----|--|
| 1 | AutoTester, AutoTester Inc., Software Recording Corporation of America, 8150 North Central Expressway, Suite 1300, Dallas, Texas 75206, Tel: + 1 800 328 1196 |
| 2 | Boris Beizer, Software Testing Techniques Second Edition, Van Nostrand Reinhold, 1990. |
| 3 | McCabe, Initial paper on cyclomatic complexity definition, McCabe, T.J. 1976, A Complexity Measure, IEEE Trans. On SW Eng., Vol2, No. 4, Dec. 1976 |
| 4 | Systems Engineering, LBMS Europe, Evelyn House, 62 Oxford Street, London W1N 9LF, Tel: + 0171 636 4213 |
| 5 | Østedal, E. and Ståne, Tor A goal oriented approach to software testing, Reliability Engineering and System Safety. © 1992 Elsevier Science Publishers Ltd., England |
| 6 | Dale Walter Karolak, Software Engineering Risk Management, IEEE Computer Society Press, 1996. |

Slide 1



The IT implications of Y2k and the Euro

A presentation to QWE'98

Graham Titterington
9 - 13 November 1998

E-mail • gct@ovum.com
Web • <http://www.ovum.com>

Tel • +44 171 255 2670
Fax • +44 171 255 1995

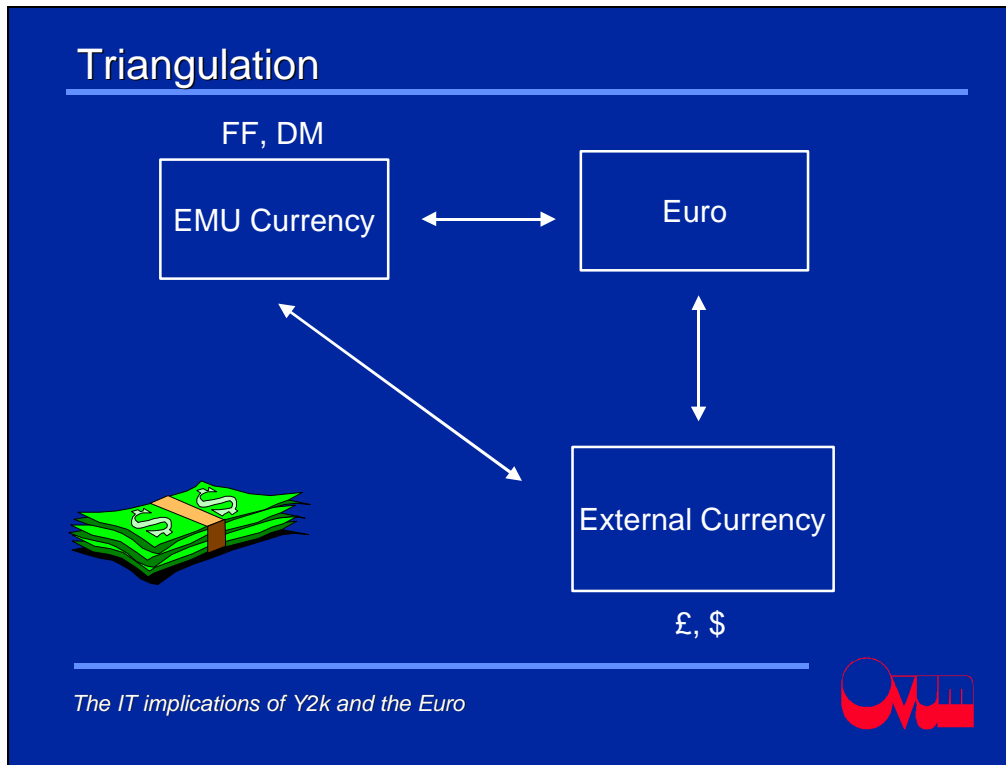
Slide 2

EMU timetable

- 1 January 1999 – the Euro is introduced; national currencies are locked to a fixed exchange rate; used by wholesale financial markets
- by 1 January 2002 – Euro notes and coins introduced; dual currency arrangements introduced for retail trade
- by 30 June 2002 – national currencies withdrawn.

The IT implications of Y2k and the Euro





- ## Differences: Issue (1)
- Euro is a business issue with IT implications
 - Y2k is an IT issue requiring a lot of business support
- EMU offers scope for building your business, but also brings risks of losing business if your competitors use it more effectively than you do.*
- The IT implications of Y2k and the Euro
-

Differences: Issue (2)

- The Euro project is about achieving business benefit
- Any business benefit from the Y2k project is accidental

Although Y2k projects should seek ways of benefiting the business, this is not the reason for doing them



Differences: Issue (3)

- Many Euro related changes will merge into business related development
- Y2k work is clearly distinct, although it blurs with development where systems are replaced to avoid remediation, or handled by routine maintenance

IT costs for the Euro will be largely obscured



Differences: Issue (4)

- EMU requirements are uncertain, for example dual accounting and accounting harmonisation
- Y2k requirements known for centuries

The uncertain features are mainly extensions to existing applications, and require more work than the mechanical conversion aspects.



Differences: Issue (5)

- You need to concurrently process new and old currencies, and store historic data
- There is no need to store 2 digit dates, once these have been widened to 4 digits

Y2k projects which adopt a date windowing solution are permanently restricting their ability to store historic data



Differences: Issue (6)

- EMU is being introduced in 3 phases
- Year 2000 comes in instantaneously

An alternative view is that the need for year 2000 processing is emerging continuously as applications process their first post 2000 plan or order



Differences: IT applications (7)

- Most Euro related changes are extensions to applications
- Y2k projects do not extend the functionality of applications

The mechanical changes are limited to exchange calculations, and modifying field sizes.



Differences: IT applications (8)

- Many new applications will need to be written
- Only new tools are needed for Y2k

Exploiting the benefits of the Euro will require completely new applications



Differences: IT applications (9)

- Euro changes are limited to the application layer
- Y2K changes permeate all layers of an application

However Euro-related changes to databases are more substantial



Differences: IT applications (10)

- Euro changes are logically more complicated than Y2k changes
- Y2k changes are more numerous affecting more programs, e.g. real-time and embedded applications



Time is a fundamental part of the working of computers.



Similarities (1)

- Happening in the same time frame
- False temptation to do both changes simultaneously
- Would issues be linked if Euro had been introduced ten years ago?



Similarities (2)

- Require changes to large numbers of programs, databases and screens
- Some elements of repetitive change, suitable for automation of search and code modification



Similarities (3)

- Require a similar project management process and environment
- Can re-use inventory, configuration management and testing tools and skills
- Happen in parallel with routine business development changes



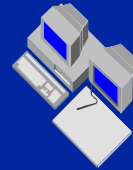
Testing

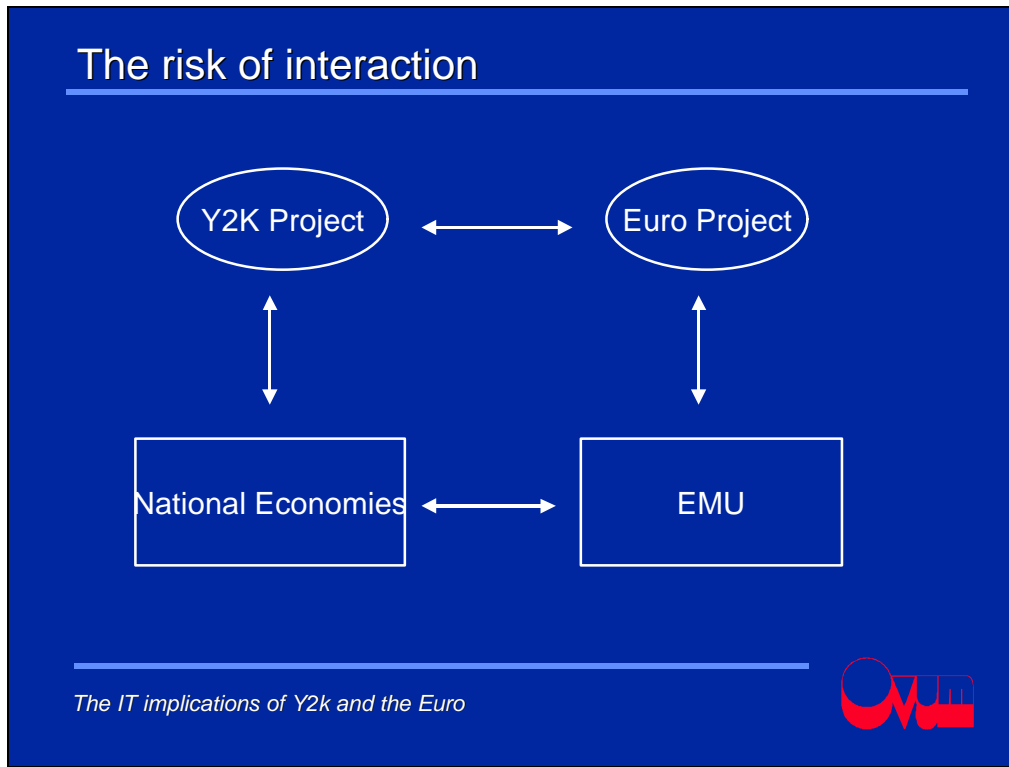
- Year 2000 testing has many unique requirements
- Euro projects split into extensions and conversion aspects
- Euro conversion aspects are amenable to testing using extensions to capture/replay tools



Opportunities for IT

- Tool vendors can leverage their R&D
- IT departments can use both projects as a vehicle for improved tools and processes
- Companies can use both projects to bring in new applications to improve overall efficiency





- ## Conclusion
- The similarity of these projects is largely coincidental
 - Nevertheless both require a similar approach
 - Most Euro related changes won't be recognised as mechanical conversion
 - The Euro project is more flexible
- The IT implications of Y2k and the Euro
-

A comparison of the IT implications of the Y2k and the Euro issues

Presentation to the 2nd International Quality Week (QWE'98)

Brussels, 9 - 13 November 1998

Graham Titterington

Ovum Ltd

1 Mortimer Street

London

W1N 7RH

England

Tel: +44 171 255 2670

Fax: +44 171 255 1995

e-mail gct@ovum.com

Abstract

The IT implications of these two issues are significantly different. The year 2000 issue requires us to convert our applications which use a representation of dates which has become obsolete, to one which will meet our needs for several years to come. The Euro issue is mainly about developing and extending applications to meet a new business environment. Nevertheless the coincidental timing of the two projects, together with the fact that both require audits of existing applications and modifications to large numbers of them, has focussed attention on what they have in common. It has even caused confusion about their nature, particularly with regard to the Euro project.

Change is imminent

In just 2 months time Europe will see the introduction of European Monetary Union (EMU) and a single currency called the *Euro*. Although it will be another 3 years before it becomes a full currency, with notes and coins in circulation on the streets, it will have an immediate impact on business that will extend to the IT operations of most organisations. From the IT point of view it is unfortunate that this is happening at the same time that the year 2000 problem reaches its climax, since these two issues are competing for the same limited resources. Many commentators are linking the two issues closely but Ovum believes that there are more differences than similarities between them.

In this paper I look at the introduction of the Euro from an IT perspective, and compare it with the year 2000 remediation project.

The repercussions of European Monetary Union will be felt by all organisations that trade in, or with, any country that enters into the EMU. Thus, organisations will not escape the need to change their financial systems, even if they base their operations outside the European Union (EU) or in an EU country not participating in the EMU.

Timetable

The outline programme is:

- 1 January 1999 – the Euro will be introduced; participating national currencies will be locked to a specific exchange rate with it; and wholesale financial markets will start to trade using it
- by 1 January 2002 – Euro notes and coins will be introduced; dual currency arrangements will be introduced for retail trade in each participating state
- by 30 June 2002 – national currencies will be withdrawn.

It is worth noting that the latter two dates can be brought forward, but cannot be delayed without renegotiation of the governing treaty. Since January 1st is not an auspicious time to introduce new currency, with most businesses closed down and January sales in a state of hectic excitement, it is likely that a date in the autumn of 2001 will eventually be chosen. Also many people in the retail sector estimate that 50% of the cost of changing over to the Euro will be specifically incurred during the 6 month period of dual currencies scheduled for 2002, and would like to see this period substantially reduced. Ideally they would like to see it replaced by a period when old currencies will be accepted by banks but will not re-issued, so that the old currencies will disappear from circulation very rapidly. This would be similar to the process which is used when a country replaces some or all of its notes or coins by a new design, or when Britain introduced decimal currency in 1971. If this strategy is chosen the entire process could be completed before the end of 2001, and businesses will have to complete their preparations more quickly. However for the purposes of this paper I will assume that the timetable above will be adhered to.

IT resources

As far as IT departments are concerned, the Euro and year 2000 projects are competing for the same resources of people, development infrastructure and testing facilities. In addition to year 2000 and the Euro, businesses have other urgent needs to develop new systems to provide new services or adapt to the changing requirements in their industries. Leaving a business to stagnate for four years is almost as suicidal as failing to cope with these two global issues.

However we strongly recommend that the activities should be separated as far as possible, and against modifying the same programs for different reasons concurrently.

To minimise conflicts, we recommend postponing as much Euro related IT work as possible until the year 2000 renovation is complete. An absolute requirement for trading in post-EMU Europe is to survive the millennium.

Fortunately some of the work needed in connection with year 2000 provides a head start in preparing for the Euro. This includes:

- producing an inventory of IT systems and programs
- providing a heavy duty project management and configuration control system
- setting up a test and development environment
- sharing many of the test cases.

Characteristics of the projects

The Euro

The introduction of the Euro is primarily a business issue. Beyond a relatively basic requirement to enable financial transactions to occur, the purpose of a Euro project is to deliver business benefit. The role of IT is to support business needs.

However, it is likely that about half the total business costs associated with adopting the Euro will be spent in the IT departments. IT changes for the Euro can be categorised into four groups:

- changes needed to enable businesses to trade using Euro
- extensions needed to maintain historical records in old currencies
- extensions to enable systems to handle both Euro and national currencies during the transitional period
- new systems and extensions needed to support new business methods.

From the IT perspective, the “temporary” features needed in the third group dwarf the “permanent” changes in the first two groups.

Changes needed to enable businesses to trade using Euro

These comprise changes to:

- enlarge the size of fields holding monetary values
- add a decimal point in countries where there is no sub-unit of currency in common current use, because the main unit of currency is of relatively low value (for example Italy, Belgium, Portugal, Spain)
- add a currency symbol to avoid ambiguity where this is not already present
- redesign forms and schemas to accommodate these changes, if necessary
- modify the ranges of values that are allowed in currency fields and variables.

Many financial processes are triggered by financial values (for example credit checks) and so clearly these trigger levels will need to be re-calculated for the Euro. However we would hope that these values are not hard-coded in an application!

Note that there are no permanent changes affecting foreign exchange calculations. Once the transitional period is complete the Euro will be a

normal international currency with a very large multi-national economy behind it.

Changes needed to maintain historical records in old currencies

There will be a need to store data about transactions which were conducted in old currencies, and to make calculations using this data. This need is partially derived from legal requirements to keep accounts and partially for planning purposes. This extension to the functionality of applications will not be accomplished by mechanical means.

Extensions to enable systems to handle both Euro and national currencies during the transitional period

The greatest challenge of the Euro for IT departments is in coping with the transitional arrangements over the next four years. It is not yet clear how much parallel reporting will be necessary to present financial data in both Euro and national currencies. Retail outlets will have to accept two currencies for up to six months – this will clearly have substantial implications for point-of-sale terminals. Probably the simplest solution will be to have two such terminals at each point of sale, but even this solution will require substantial modifications to the central processors supporting them, and the suppliers of such equipment will have a difficult task in meeting these demands.

If it is decided that invoices and receipts have to give prices in both currencies, substantial redesign of their formats will be needed.

For the transitional period, participating national currencies will only be convertible into and out of Euro, and using six significant digits for calculating the conversion to avoid guaranteed gains for currency dealers in the fixed rate environment. (Traditionally only four significant digits are used in foreign exchange calculations.) The exchange rate between the Euro and other currencies will fluctuate according to the world financial markets. There will no longer be a quoted direct exchange rate between participating currencies and external currencies (for example, between Francs and US Dollars). These conversions will have to be calculated as a two-stage calculation that determines the value in Euro as its intermediate step. This procedure is governed by a treaty regulation (*Article 235*) and has the status of a European law. Inverse calculations must either divide by this same, six significant digit conversion factor, or use a procedure that gives exactly the same result. The allowable limits for rounding the results are also specified.

Article 235 only applies to the conversion of European currencies. It does not affect other transactions, such as interest calculations.

These requirement means that the Euro cannot be treated as just another foreign currency during the transitional period, although the rules do allow flexibility in deciding how to implement the calculations.

There will be a need to report in both Euro and national currency during the transitional period. The extent of this is not yet clear – within the first three years of the transition, organisations within participating countries can choose their own date on which to convert their internal systems. However from the beginning of this period, a business will have to be able to accept invoices in Euro. Those organisations with operations in several participating countries will gain more benefits from adopting the Euro at an early date. In practice pressure from commercial partners will dictate the conversion schedule for many businesses. Despite these pressures to move to the Euro, no organisation can change over all of its systems to the Euro before 2002, because its employees and pensioners will want to be paid in a currency that they can spend and it also has to be able to pay invoices in old currencies.

Small enterprises that do not already have experience of multi-currency trading will have the greatest problems in setting up dual currency arrangements.

New systems and extensions needed to support new business methods.

The need for these changes are not clear yet, and will emerge over the transition period, and beyond. We can expect to see new financial services, and more international trade requiring expansion of supporting services.

There may be harmonisation of accounting practices across Europe to facilitate international commerce. However these are not part of the existing treaty commitments and are certainly not going to happen in 1999. This area may eventually become the most significant work area in terms of updating IT systems to operate in the post-EMU Europe. For example, it has been suggested that interest calculations will have to be based on the *change of day* principle. These changes may also apply in EU countries that do not participate in the EMU. However, this is an area where the requirements are still unclear.

Year 2000

The year 2000 project on the other hand is a very large mechanical conversion task. The purpose of the project is to enable the organisation to continue to operate, ideally as well as it does at present. Although we do gain some benefits from year 2000 projects, such as improved IT methods and practices and a better organised portfolio of applications, any benefits are incidental. The project is justified by necessity! Some extension may be necessary to correct programming bugs in the original application, such as to recognise year 2000 as a leap year. The objectives of the year 2000 project are well understood and limited. However because of the scale of the task and the totally rigid timescale we are likely to have to prioritise tasks within the project in a way which has no parallel on the Euro project. We can live with minor date problems, such as the incorrect century appearing on printed reports, while we can not accept financial statements in the wrong currency.

It is a simplification to regard all the year 2000 remediation activity as a homogeneous task. The problem is so pervasive that we need to adopt significantly different approaches to remediating different types of application and different types of modules.

Political uncertainty

The Euro is man-made and is subject to political will. Many details are still undecided and we have to be ready to make changes at short notice. For example, the timetable is not yet firm; at the moment it is more likely to be brought forward than to slip backwards, as this would require a treaty modification.

Other uncertainties include:

- which countries will join later, and when such an opportunity will arise
- whether there will be a requirement to publish statutory reports in Euro, or in both Euro and national currencies, during the transitional period
- how financial markets will function on days which are public holidays in one of the participating countries
- some details about how EMU will operate.

A concern is that the political masters may not understand the timescale needed to implement the IT changes. Since IT systems will have to conform to

legal and political dictates, this uncertainty is an added burden for Euro projects.

Conversely, year 2000 is a natural certainty that will happen on a specific day.

Scope of the software problem

The number of programs affected by the EMU is much smaller than the number affected by year 2000.

The Euro is concerned with money; year 2000 is concerned with time. While very many programs are concerned with manipulating money, calculating costs and recording and reporting financial events, the good news is that money is not as pervasive as time in the computing world. For example, in the Euro project we will *not* need to worry about:

- how to simulate or manipulate the hardware clock
- software licences that expire when the clock is reset
- back-up data that is deleted when the clock is changed
- hardware, BIOS or operating systems that are not Euro-compliant
- the word processor
- embedded systems that drive vital life-support systems and which stop when the date ticks to zero
- the lift ceasing to operate (provided that we have managed to pay for our electricity!).

Outside the financial services industry, and until the time when the Euro currency comes into circulation (1st January 2002), the repercussions of the Euro will be limited to data processing applications. From this time onwards it will be necessary to update all cash handling equipment, including electronic point-of-sale terminals.

Although the scope of the problem is more limited, where changes to programs are necessary, they are more logically complex than the typical date related changes. The date problem is mainly about data representation, whereas the Euro changes involve altering the logic of applications and developing some substantial new applications.

Similarities between the Euro and year 2000 projects

The Euro project can be divided into:

- semi-mechanical modifications in currency conversion calculations and presentation of financial data
- functional extensions and enhancements.

The first of these two groups has some similarities with the year 2000 project:

- start by creating an inventory of systems, identifying which have dependencies on currency, and determining priorities
- planning, management and configuration control are the keys to a successful project
- search the code to locate the parts that are dependent on currency, and then determine the impact of any changes
- co-ordinate with software suppliers to determine whether, and when, Euro-compatible versions of their products will be available

- thorough testing is needed to ensure both that the changes are accurate, and that the unchanged functions are unaffected.

Project structure

The overall project structure for the early stages of the Euro project is very similar to the one needed on year 2000 projects and needs to include the following stages:

- business/IT strategic plan
- feasibility study
- external/internal design
- programming
- testing
- implementation and roll-out of the new systems.

In order to provide the major extensions and new functionality to support the Euro, an IT project needs to follow the same procedure as any other new development. Year 2000 does not have a parallel for this type of development.

Testing systems

These two projects share a need for good test management capability to manage the scale of the testing activity and maintain the test results along with configuration information about the application under test.

The Euro

Euro-related development comprises new (or radically altered) systems, and updated systems.

New systems

New Euro systems should be tested as comprehensively as any other new system. Since many of these systems will provide on-line services, their performance and load-handling properties are particularly important.

Modified systems

As usual in code maintenance projects, the modules that have been altered must be thoroughly tested. After this, regression testing focusing on the functionality of the application is the main concern. Automated capture/replay tools can be used to check the unchanged aspects of applications. The currency fields can be masked and checked manually. However, a better solution is for automated tools to provide rules for checking whether these fields have been updated correctly. Tool suppliers are starting to take up this challenge, for example by providing bolt on modules to capture/replay tools.

An important aspect of the regression testing is to check that the interfaces between system components, including interfaces where data is transferred through a database, are totally consistent and that the whole system continues to function as a whole.

Performance and load testing are not major concerns for those aspects of the Euro-updated systems which are susceptible to mechanical conversion, because the nature of the Euro-related changes will not significantly alter either the processing requirement or the volume of data moving around the system. However, one aspect that should be checked is the amount of extra storage needed to retain dual accounting records. And of course any new or

extended applications may be critically impacted by performance considerations.

Test data generation for updated systems

For the mechanical conversion aspects of the Euro project, test data can be derived from the existing systems. It can come either from capturing the output of the applications, or their existing test data can be used if this is available. In both cases, however, financial data will need to be scaled to provide values in Euro. Where a test is designed to explore the limits in the range of values that a program can handle, the new limits will need to be substituted. In a country where the national currency does not include a sub-unit field, or is of a different order of magnitude from the Euro, it is essential to modify all money values in the tests.

Year 2000

Year 2000 projects have significant and complex special needs for testing. It is efficient to unit test changes areas of code, as well as to test remediated systems. Typical year 2000 projects need to spend about 60% of their resources on testing.

A system clock simulation tool is virtually essential for these projects as many year 2000 projects work with mainframe-based applications and there are several reasons which make it difficult to change the system clock for a test run, such as triggering the expiry of software licences and impacting other applications running on the computer at the same time.

Test data generation at the system level normally proceeds by capturing actual input and output from the existing version of the application. Test data for the next century can be constructed from this by a technique known as date-ageing which advances all dates in the test scripts and test data by a specified period. Many tools are available to do this and they exhibit a range of sophistication. Many applications require dates to observe business rules, for example that transactions do not fall on a Sunday. Simply advancing all dates will violate these rules. However tools which adjust the new dates to satisfy the rules can create problems where the application calculates the interval between two dates, such as for an interest calculation. These complications contribute to the scale of the year 2000 problem!

A comprehensive set of test cases for the remediated code is needed. It should cover all feasible combinations of 20th and 21st century dates as well as exercising any special functionality associated with particular dates, such as leap years or the end of the financial year. Tests should also include cases where each of these dates enters the application through its interfaces, through its database or filestore, and through its system clock. Special tools are needed to age data within a test database as it has to take account of the structure and syntax of the database tables.

Bridging tools are sometimes needed to transfer data between a remediated application and a data store which has not yet been converted, or vice-versa.

Roll-out of

We need to co-ordinate the roll-out of Euro-compliant systems to accommodate their needs to communicate with each other, as with the year 2000 project. This communication can be direct, or through shared databases and files. The problems can be minimised by selecting groups of programs, so that communication between updated and non-updated systems are minimised. The remaining communication problems can be resolved by using bridging programs to support data transfer, or by producing intermediate versions of programs that use a mixture of old and new interface formats.

Databases will have to be updated and reformatted along with programs. Tools, may help with this task.

Opportunities

Both projects provide business opportunities for tool vendors to meet the needs of customers who have both an urgent requirements and budgets! The year 2000 project provides the bigger opportunity because of:

- its size throughout the world
- the need for several different kinds of support.

By comparison, the needs of the Euro project largely overlap with standard application development. There is the danger of vendors using the Euro as a repository for solutions looking for a problem to solve! However vendors should gain benefit indirectly by gaining permanent customers from organisations which have used their products on year 2000 projects.

IT departments have the opportunity to use both projects as vehicles for improving their methods and processes.

Organisations have the opportunity to acquire new and improved applications. In a few cases these projects have allowed them to replace applications which would otherwise have remained unchanged for many years.

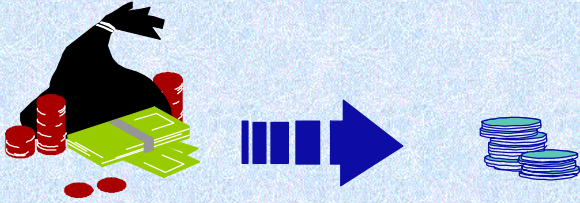
Conclusions

The similarity between the year 2000 project and the Euro project are largely coincidental. A similar approach and a similar remediation environment are required for the mechanical aspects of both projects. It follows that the same staff and skills are going to find employment in both projects. We strongly recommend that the two projects should run sequentially rather than in parallel.

However most of the changes that will eventually be required for the Euro project aren't mechanical in nature and require a more creative approach. These will be driven by business needs. The timetable for the Euro allows organisations flexibility, even if the length of the transitional periods is going to be reduced. Indeed many of the new business needs will not necessarily become apparent before the full introduction of the Euro. The Euro project is complicated by remaining uncertainties in its requirements. In particular the curtailment of the second transitional period, when both new and old currency are planned to circulate alongside each other, will simplify the project significantly. Also changes to accounting procedures will not adhere to the same timetable as the introduction of the Euro itself.

Cost of Quality

The Real Bottom Line



presented by
L.Daniel Crowley, CCP, CSTE
QA Manager, IDX Seattle, USA



Quality Week Europe Nov 12, 1998

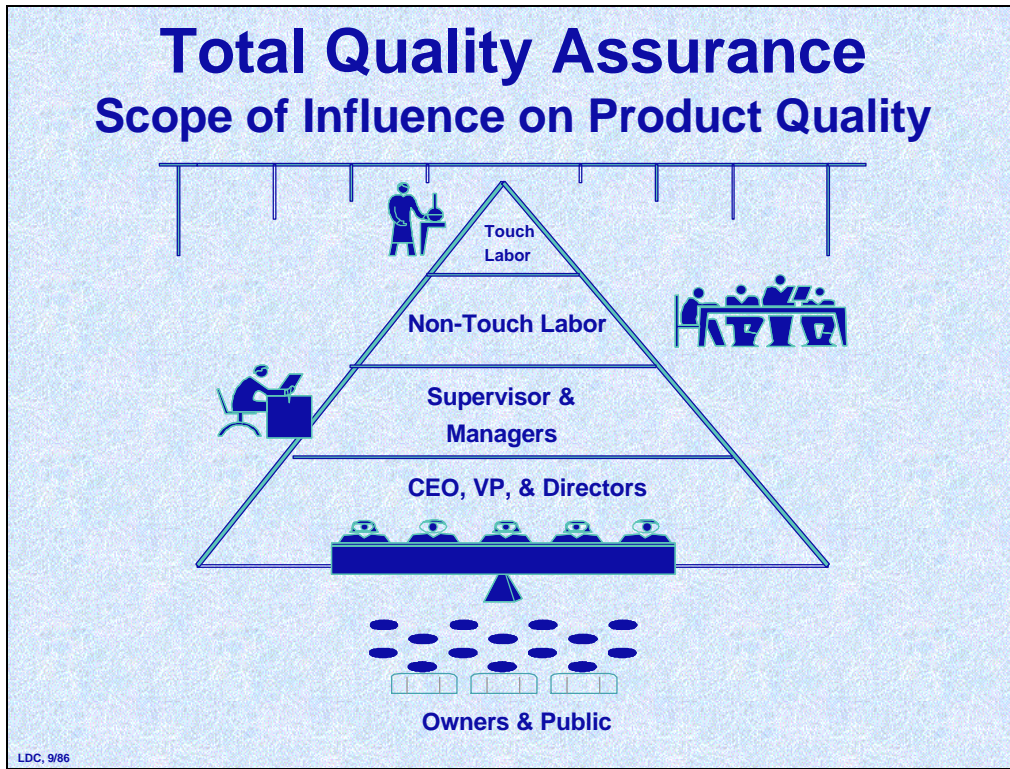
© 1998

Cost of Quality

Schedule

- Introduction
- Principles of CQI (TQA) and Cost of Quality
- Strategy for COQ
- COQ Categories
- Implementation of COQ

Slide 3



Slide 4

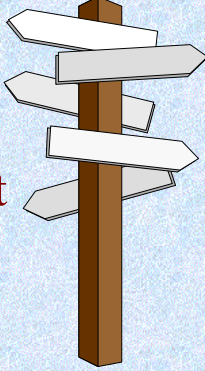


Cost of Quality

Running a company by profit alone is like driving a car by looking in the rearview mirror.

It tells you where you've been, not where you are going!


Dr. E. Deming



Cost of Quality Goal

- The goal of a Cost of Quality system is to:

Facilitate Quality Improvements that leads to operating cost reduction Opportunities.





ASQC, "Principles of Quality Costs", 1986

Cost of Quality

Definition

- The cost difference between present operation and the possible operation of a business with all systems and employees at 100% performance.

 or 

- The difference between actual revenues and what revenues could be if all customers were always satisfied, that is , No Unhappy Customers.

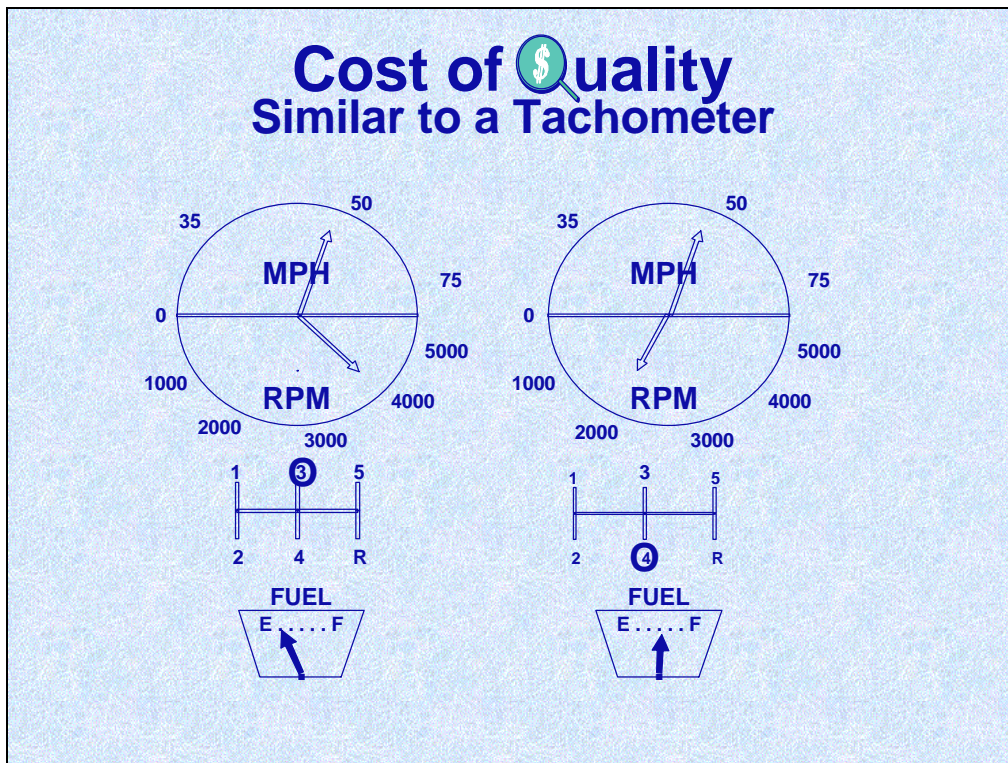
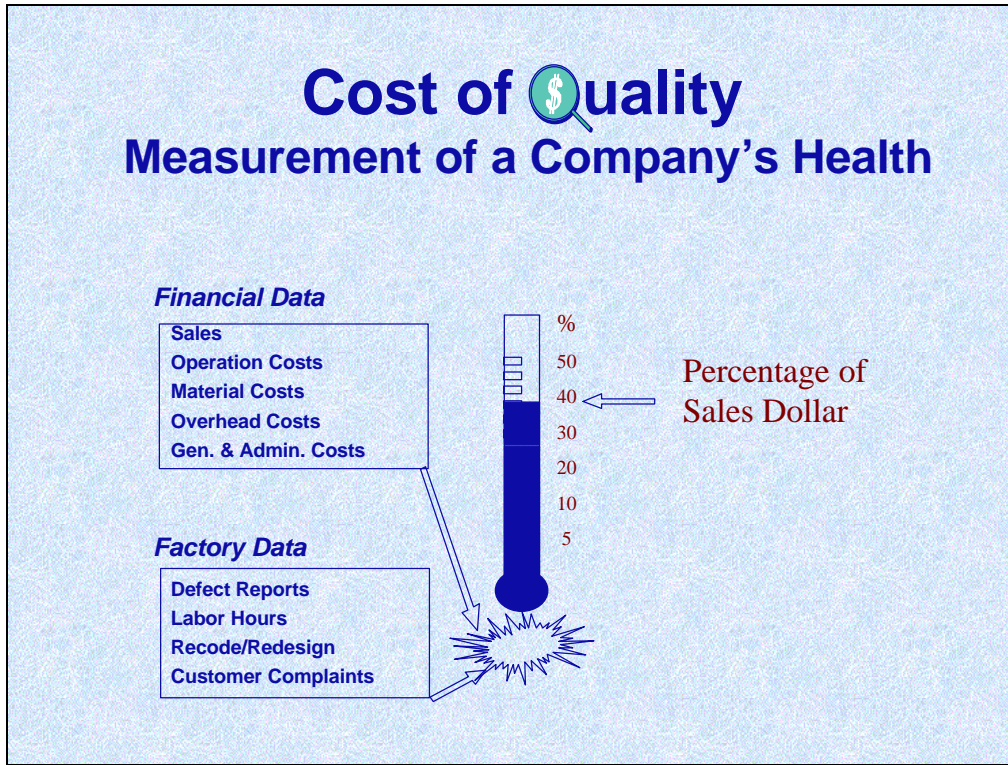
LDC 10/88

Cost of Quality

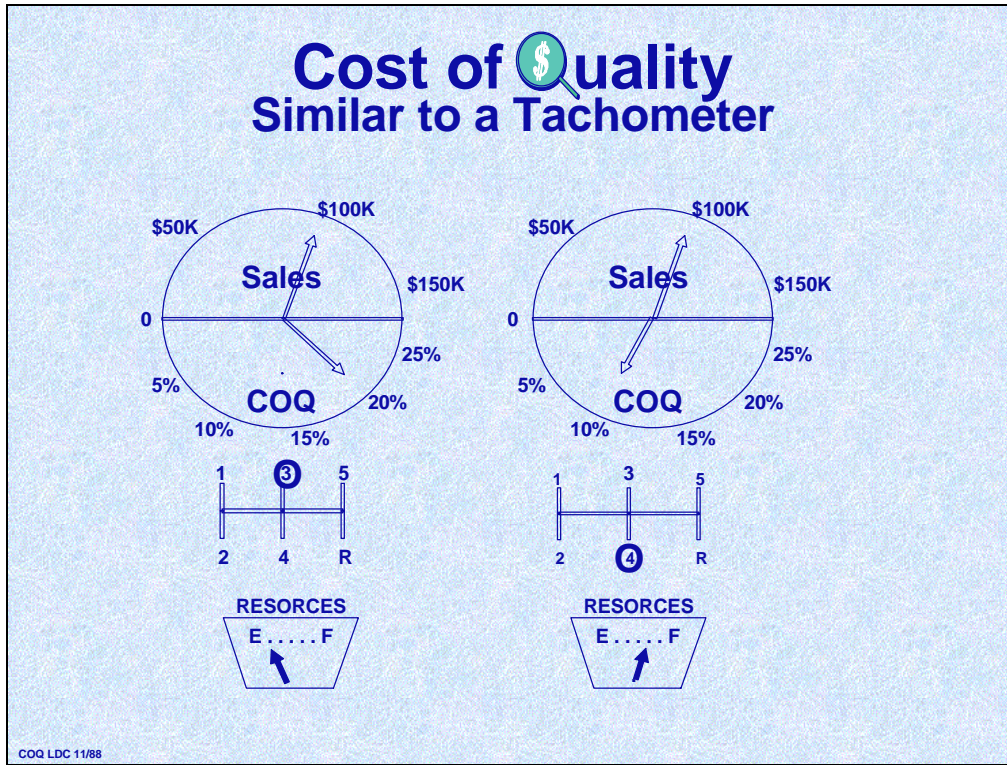
Cost of Quality is Not:

An Exact Cost. 

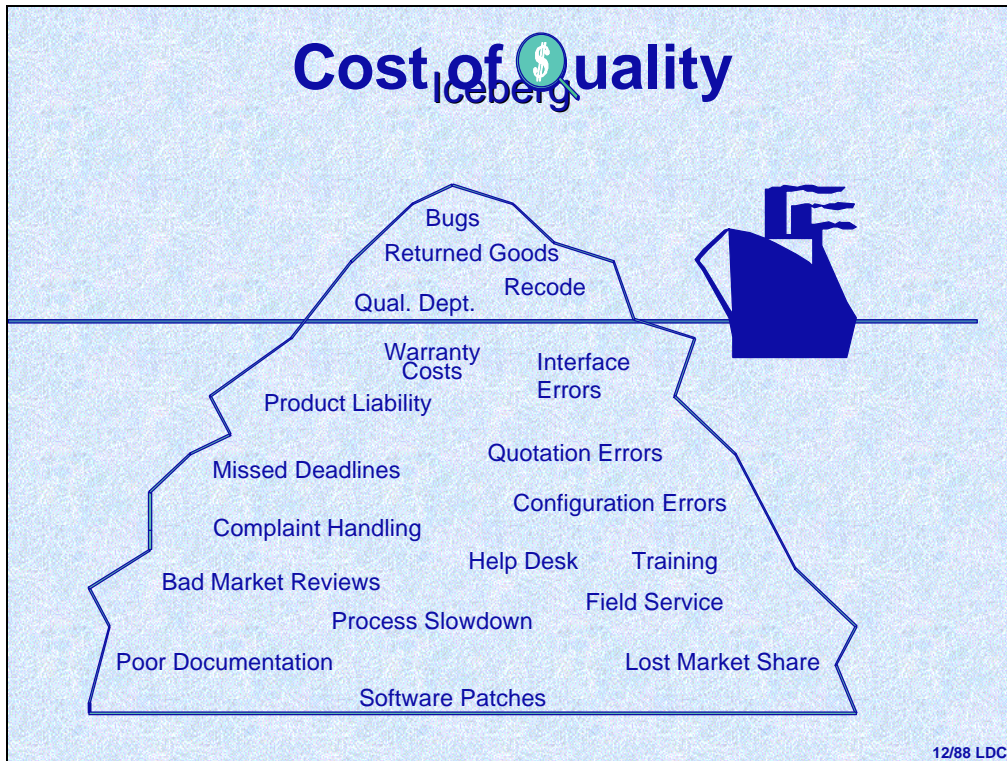
 A Performance Measurement.

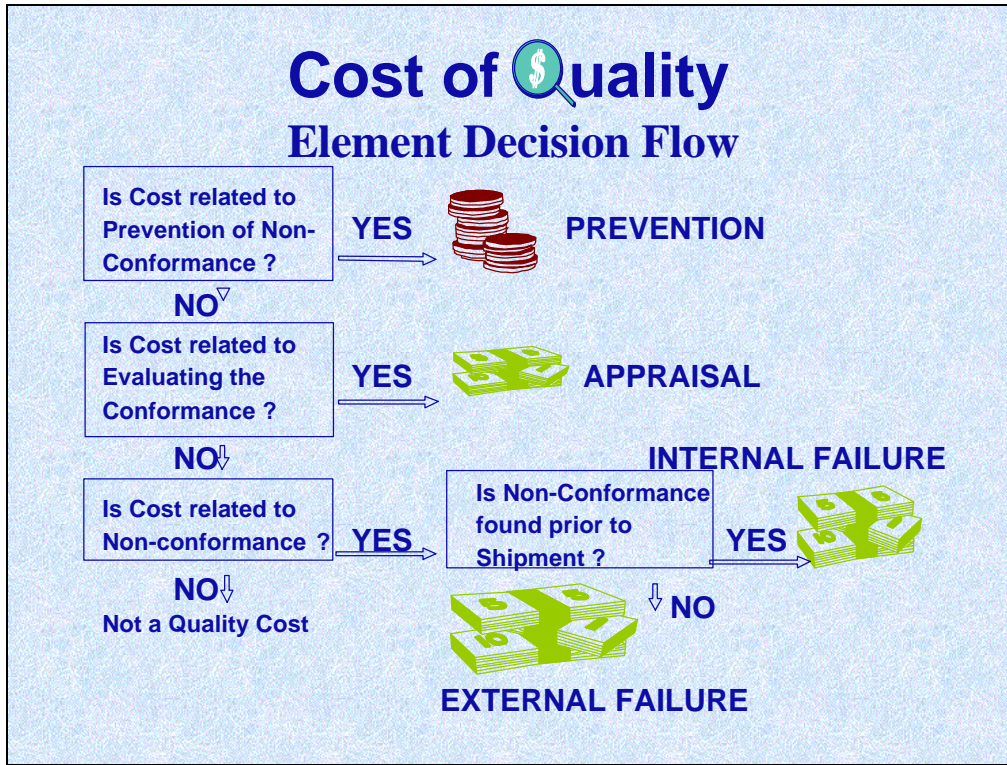


Slide 11



Slide 12





- ## Cost of Quality Examples of Elements
- | | |
|---|---|
| <ul style="list-style-type: none"> ✓ <u>PREVENTION</u> Design Quality Progress Reviews Requirements Documentation SQA Training Cleanroom Software Engineering | <ul style="list-style-type: none"> ✓ <u>APPRAISAL</u> Unit Testing Regression Testing Automated Test Tools User Interface Reviews |
| <ul style="list-style-type: none"> ✓ <u>INTERNAL FAILURE</u> Recode/Repair Labor Defect Tracking & Reports Requirement Changes Down Hardware | <ul style="list-style-type: none"> ✓ <u>EXTERNAL FAILURE</u> Returned Goods Liability Costs Help Desk Lost Sales/Market Share |

Cost of Quality Corrective Action vs Failure

- Corrective Action is paid for Once,



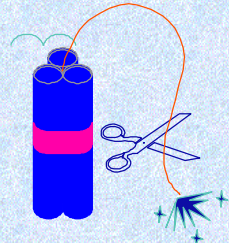
- Whereas Failure to take corrective action may be paid for over and over again.



Cost of Quality Strategy Premise

The Strategy is based on the premise that:

- For each failure there is a root cause.
- Causes are preventable.
- Prevention is always cheaper.



ACQC, Principles of Quality Costs, 1986

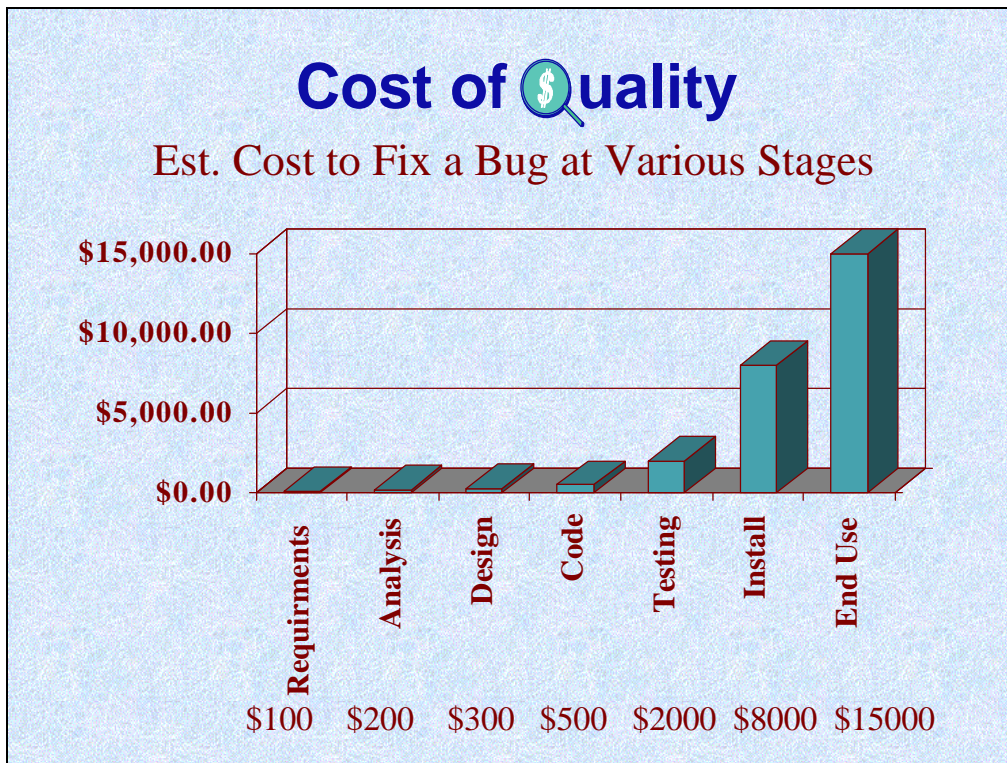
Cost of Quality

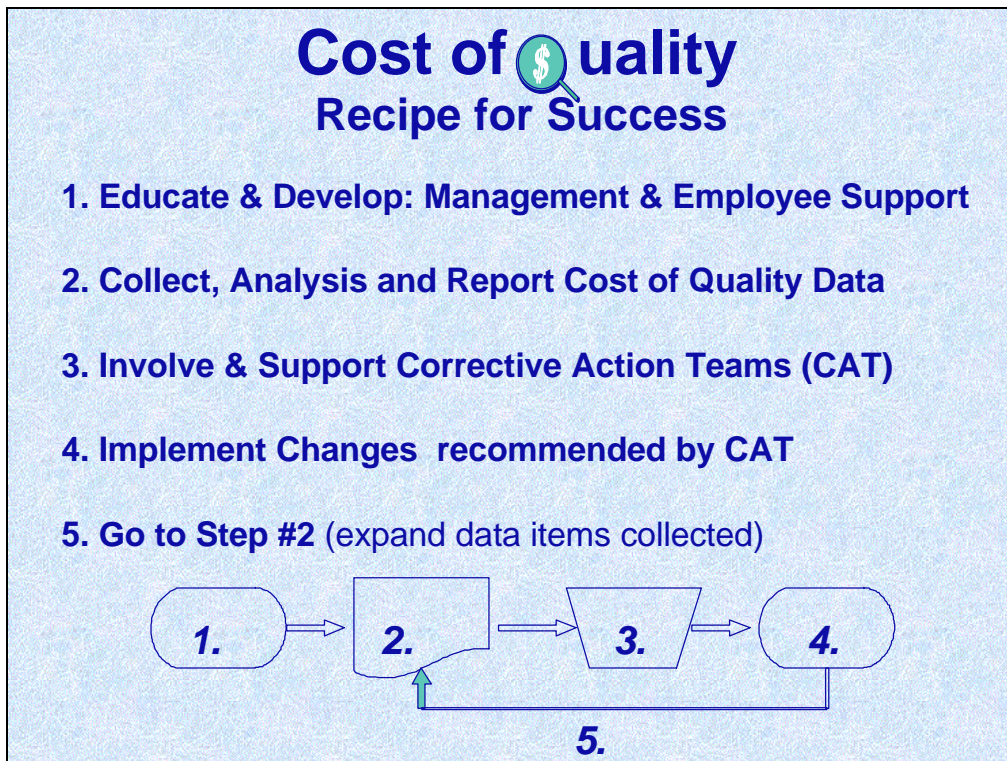
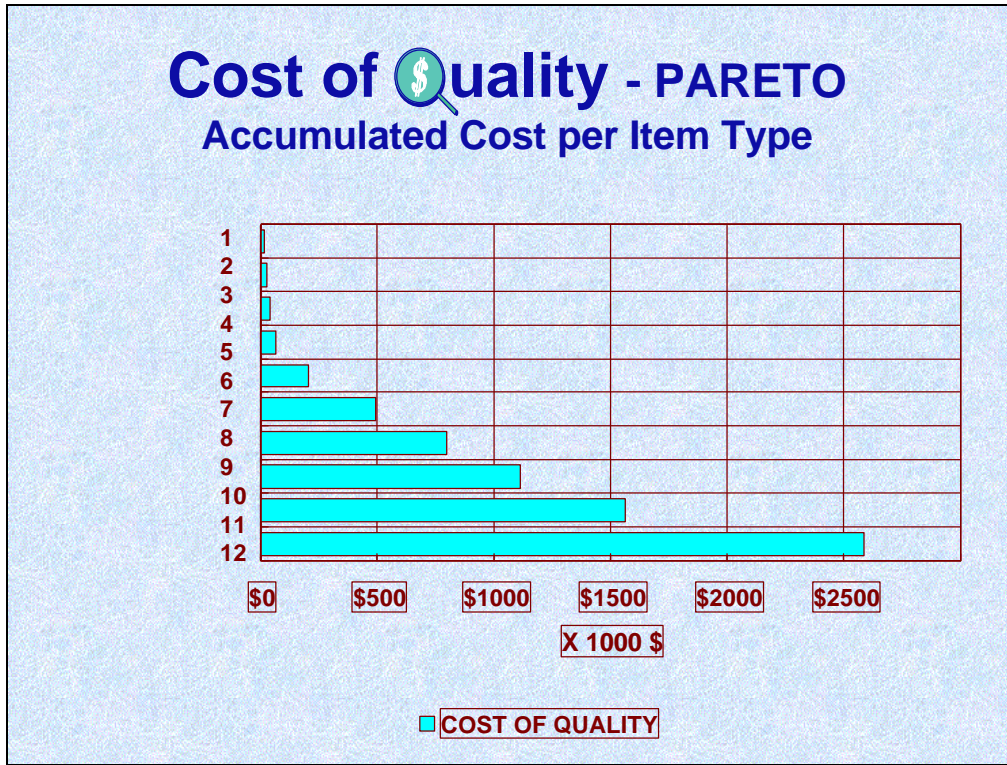
Strategy for using Quality Costs

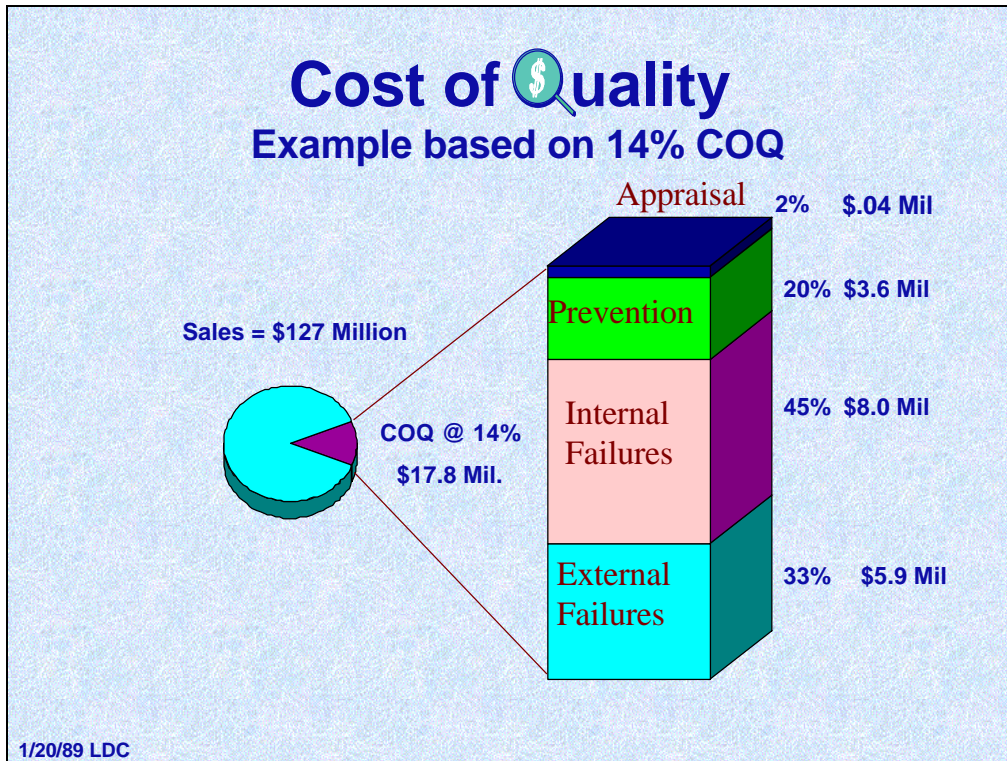
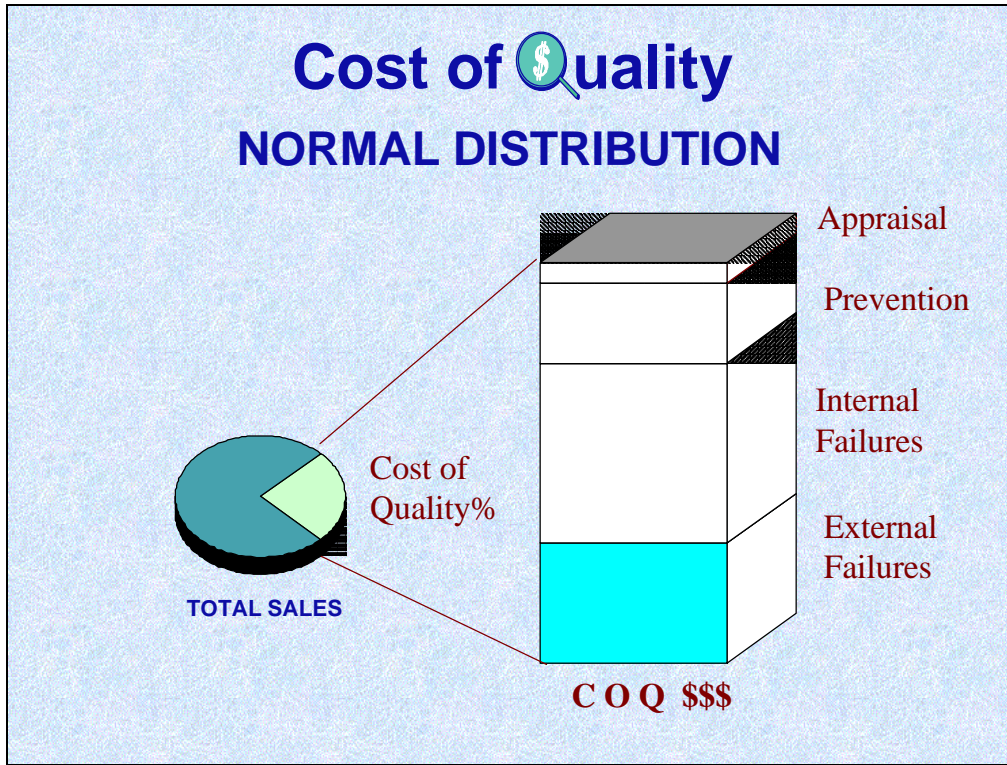
- Take direct attack on FAILURE costs, try to drive to zero \$.
- Invest in the "right" PREVENTION activities to bring about improvements.
- Reduce APPRAISAL costs according to results achieved.
- Continuously evaluate and redirect PREVENTION efforts to gain further improvement.



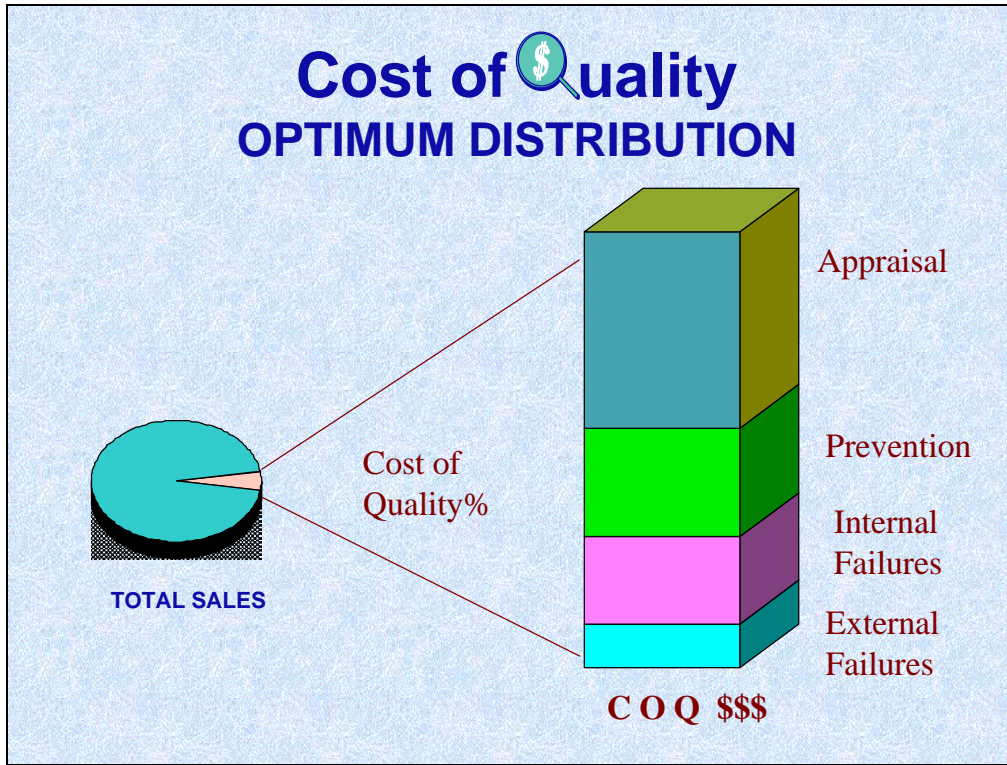
ASQC, Principles of Quality Costs, 1986



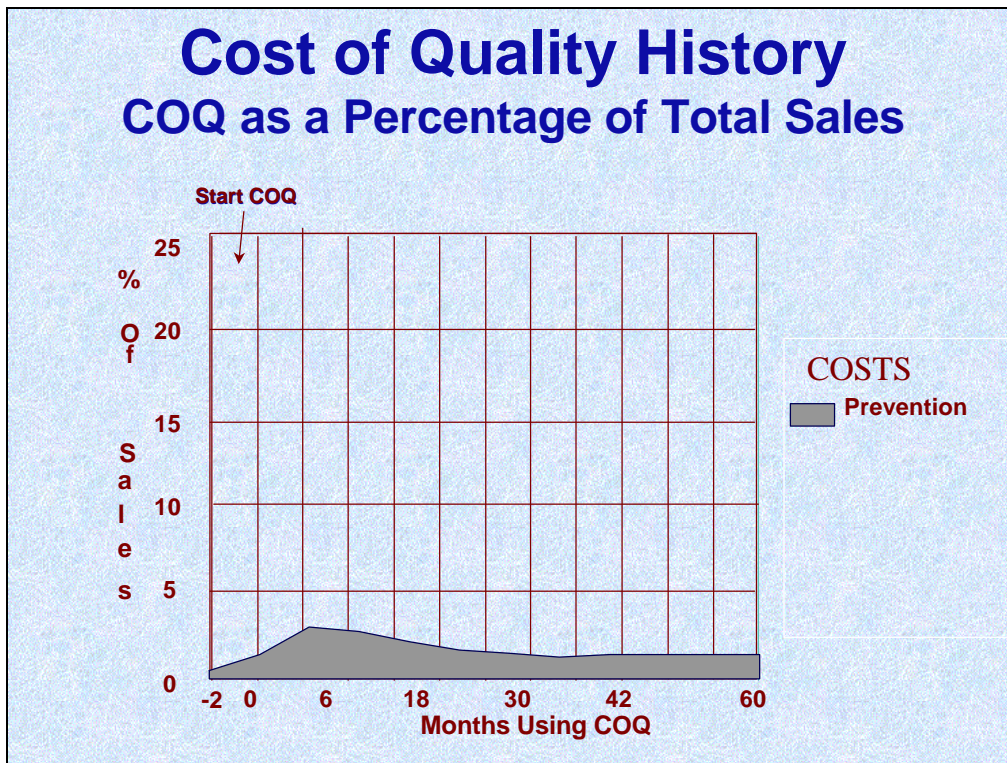


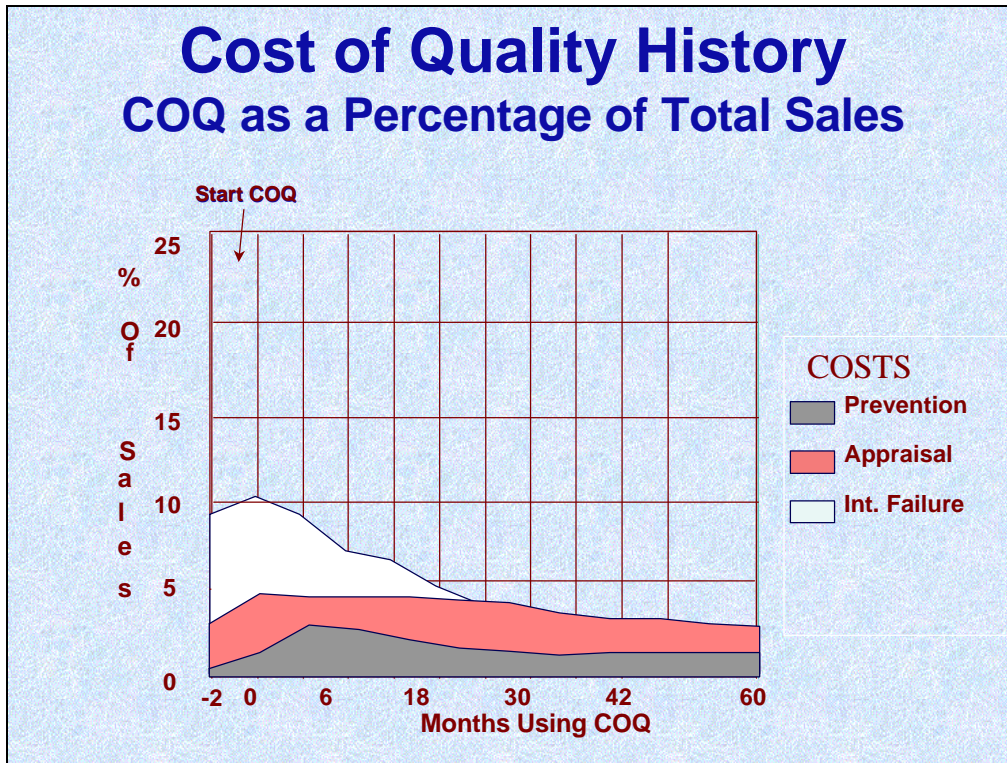
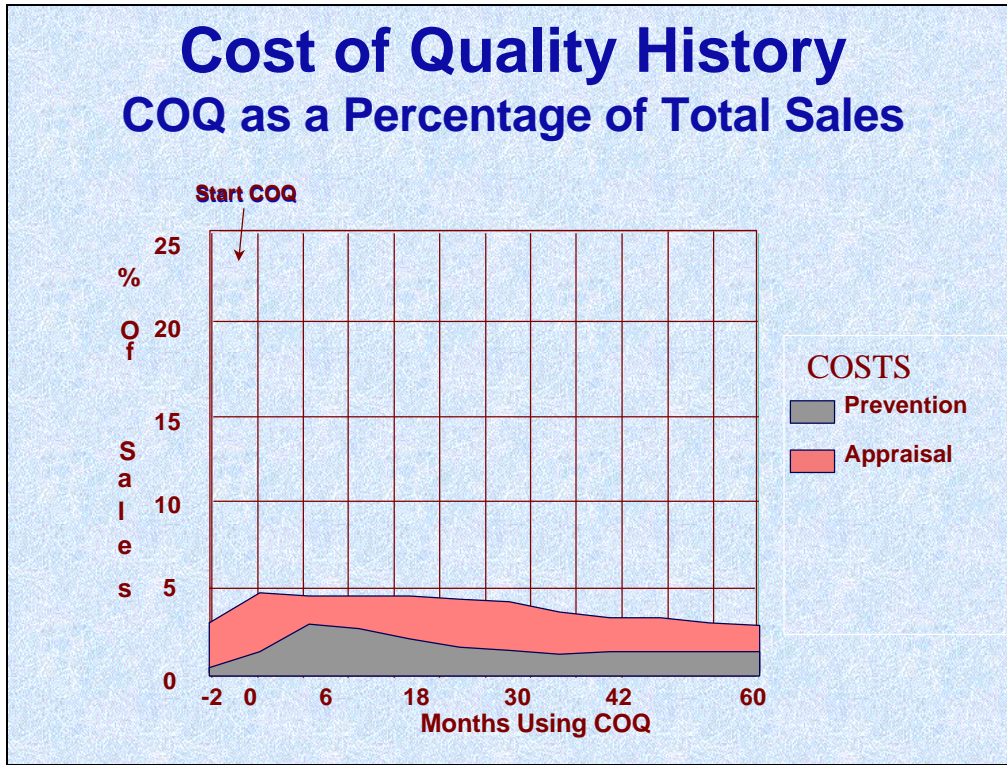


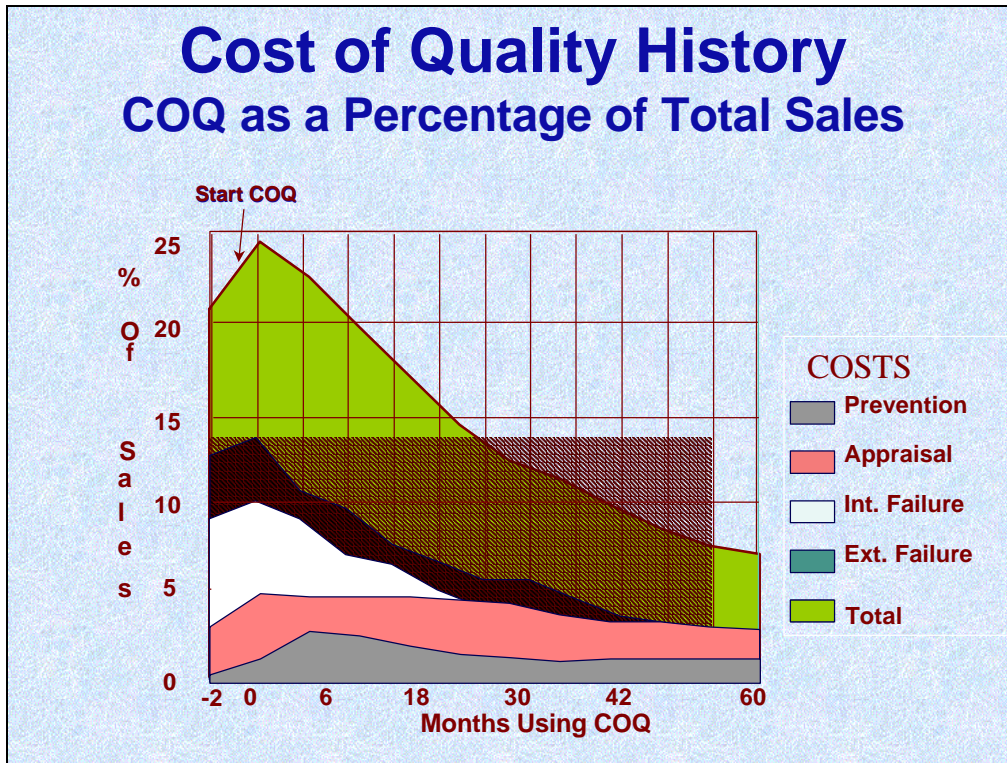
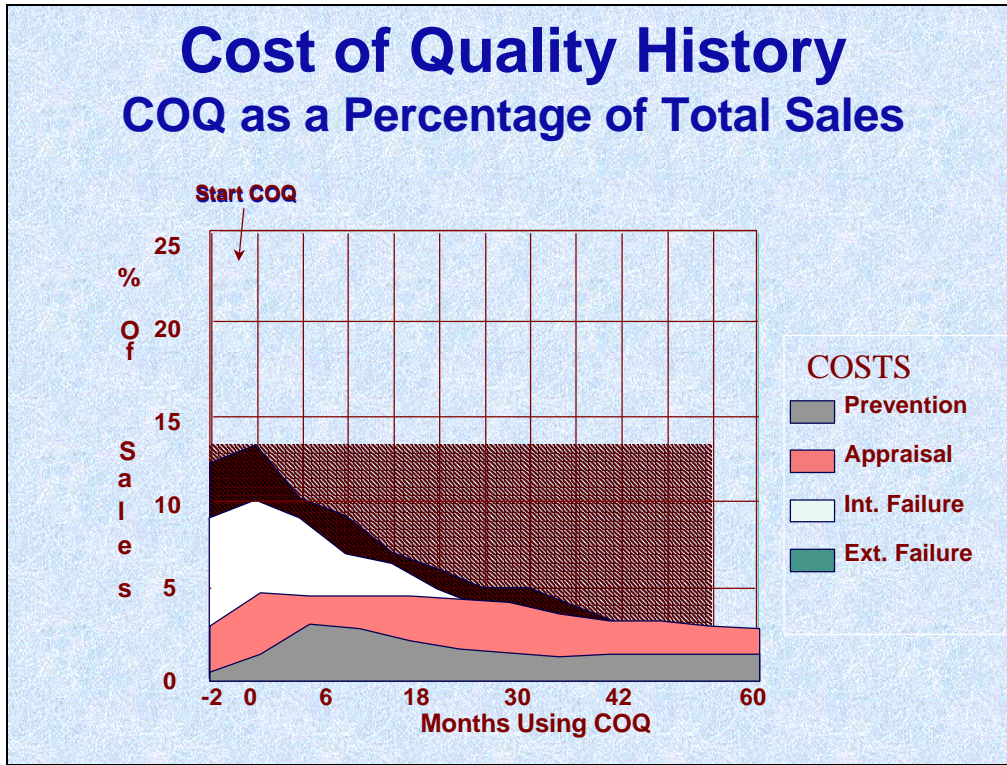
Slide 23

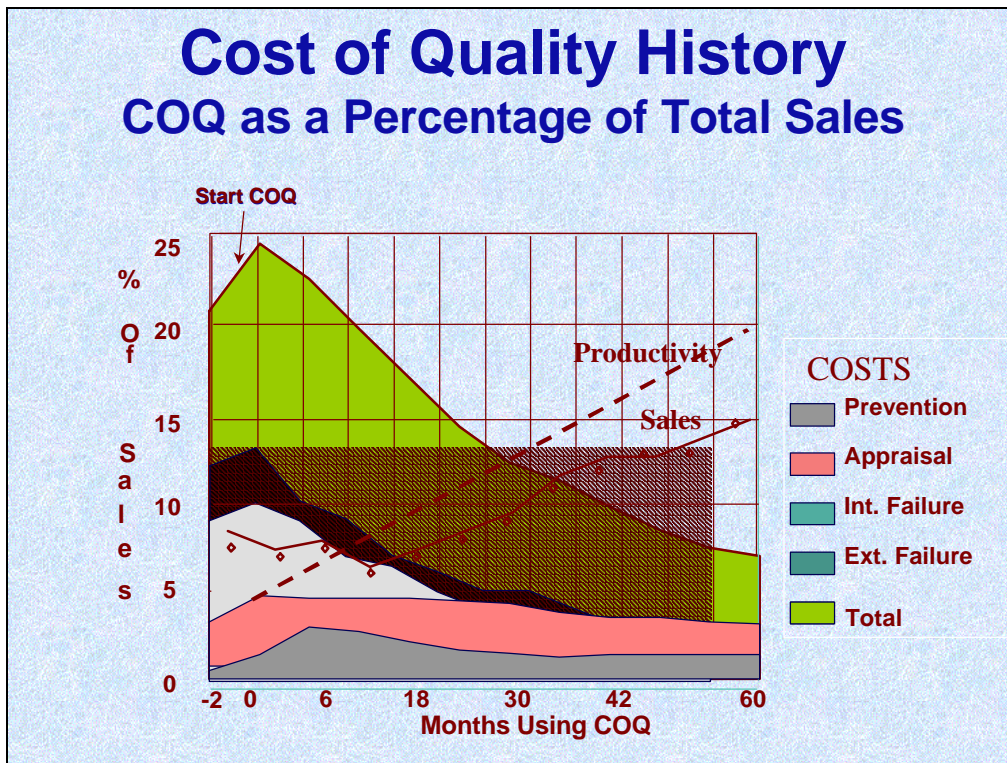
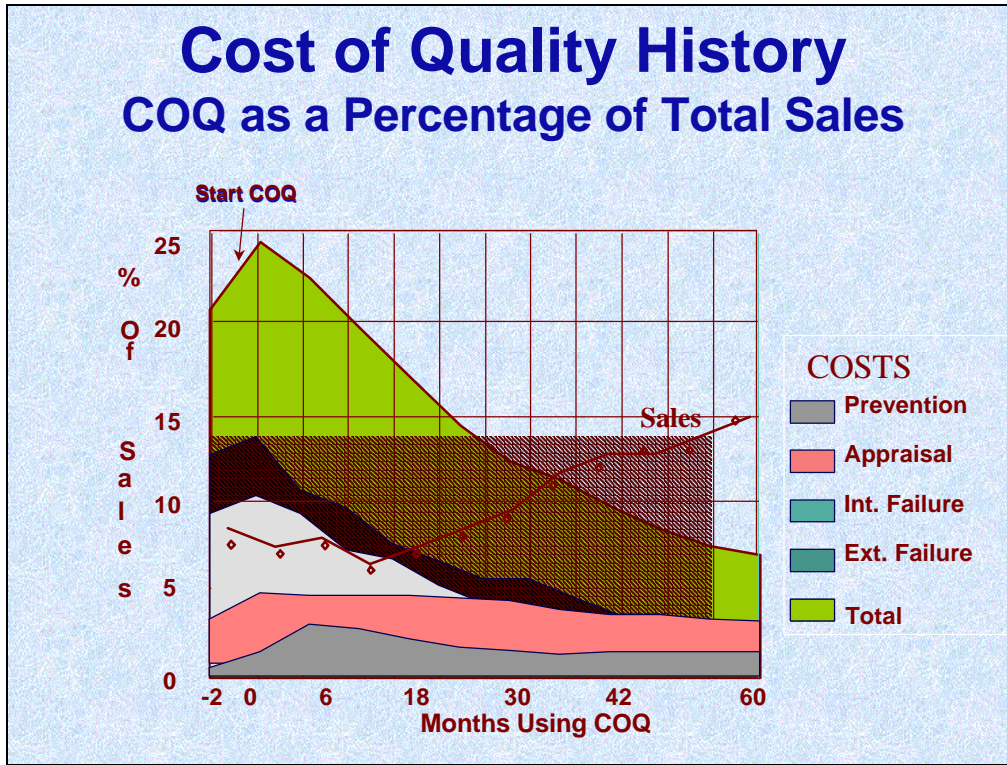


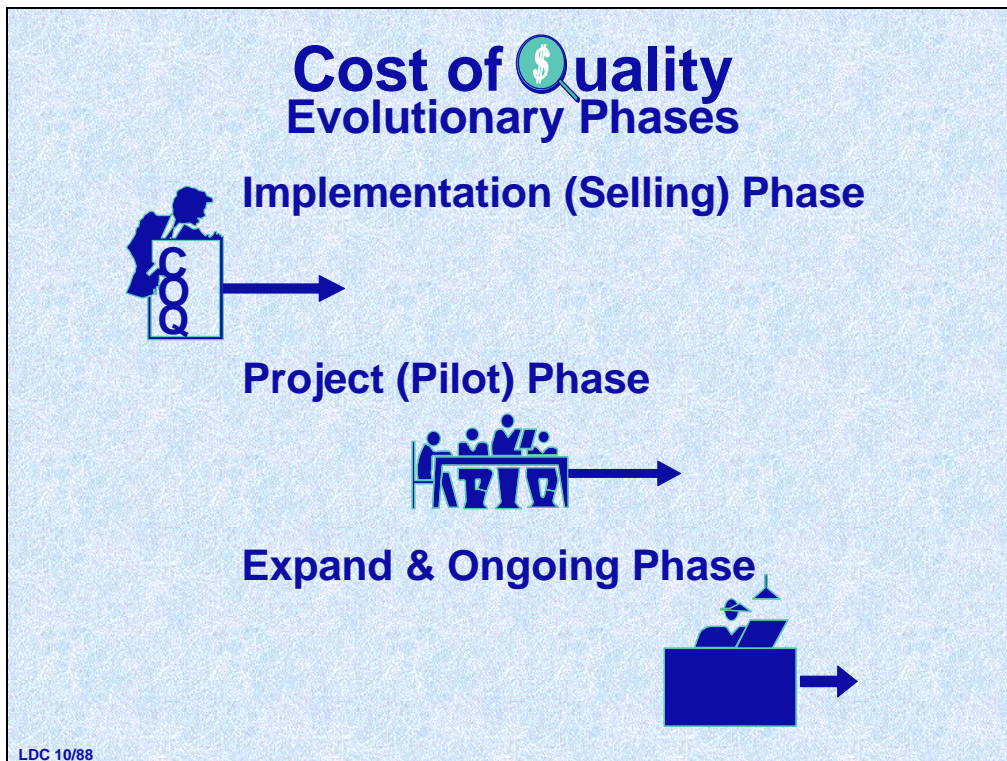
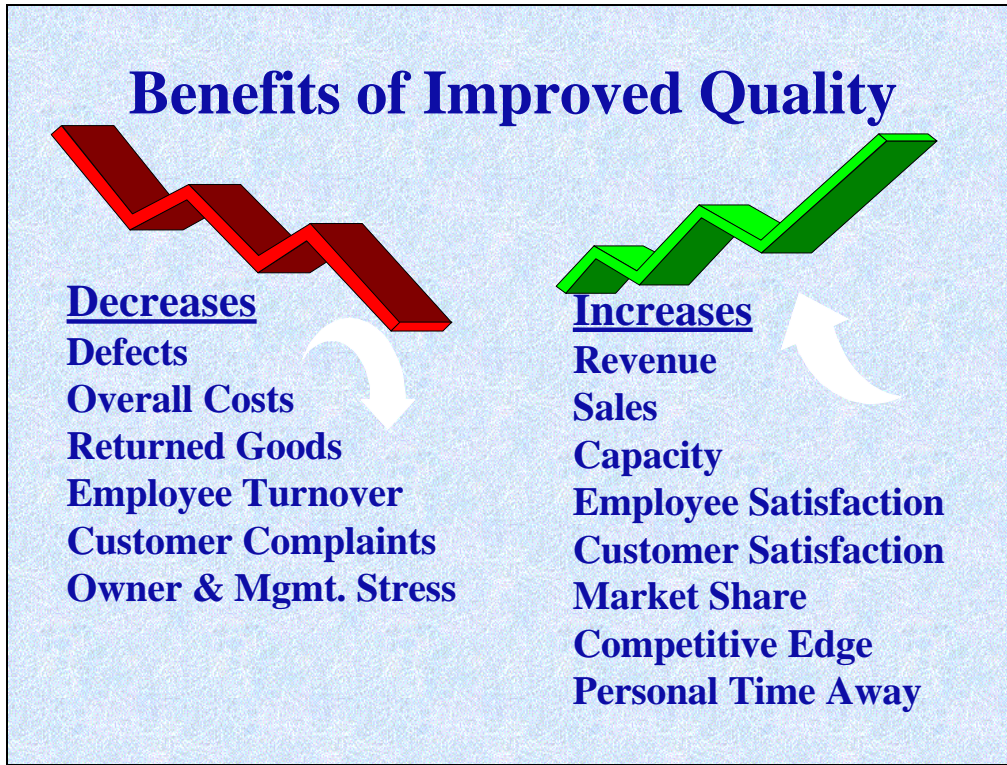
Slide 24







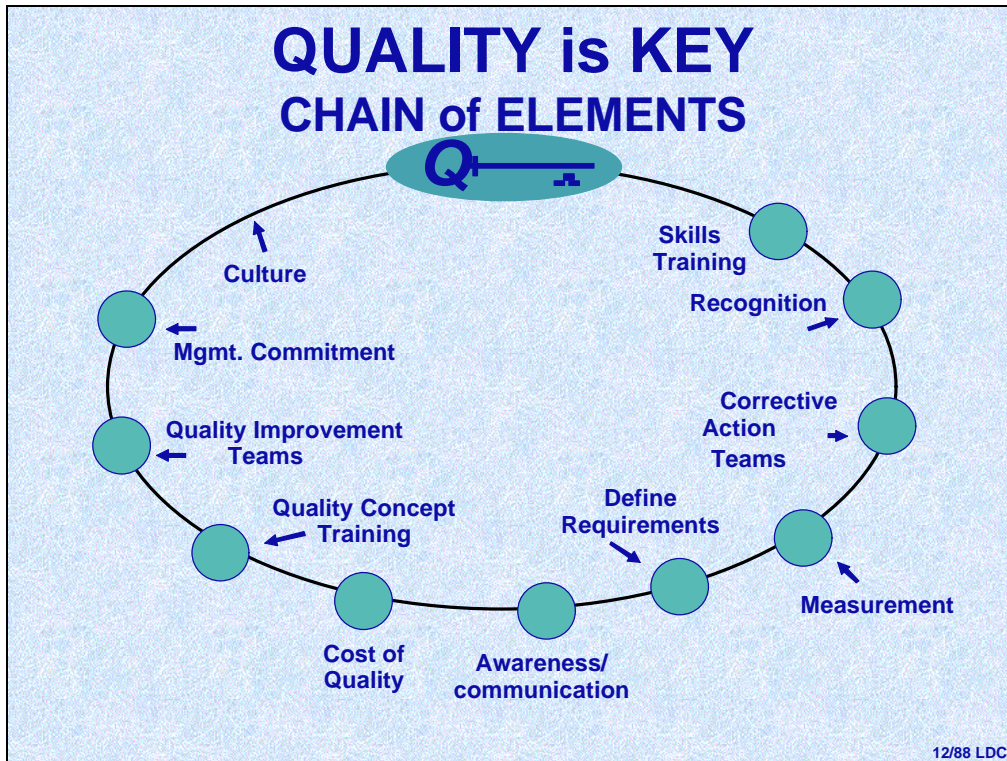





Average Industry Cost Of Quality %

- Based on Industry Standards -
Cost of Quality % of Total Sales \$\$\$

• Software Companies	30-60 % COQ
• Information / Service	20-35 % COQ
• Manufacturing	15-25 % COQ






Thank You for Listening

References:

- Principles of Quality Costs (ASQ)
- Guide for Reducing Quality Costs (ASQ)
- Quality without Tears, Crosby
- Quality Cost Analysis: Benefits & Risks, Kaner
- Quality Control Handbook, Juran

To Contact Daniel Crowley
• 206-689-1352
• Email Daniel_Crowley@IDX.com



Slide 1



Questionnaire based usability testing

European Software Quality Week '98

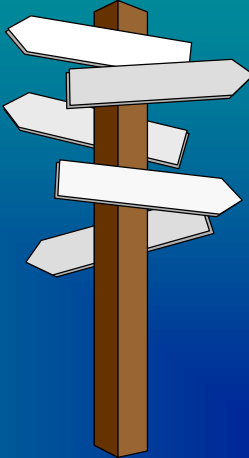
Erik van Veenendaal
(eve@tm.tue.nl)



Slide 2

Topics

- A closer look at usability
- Questionnaire based testing (SUMI)
- Practical examples
- Cost / Benefits



Usability in action....

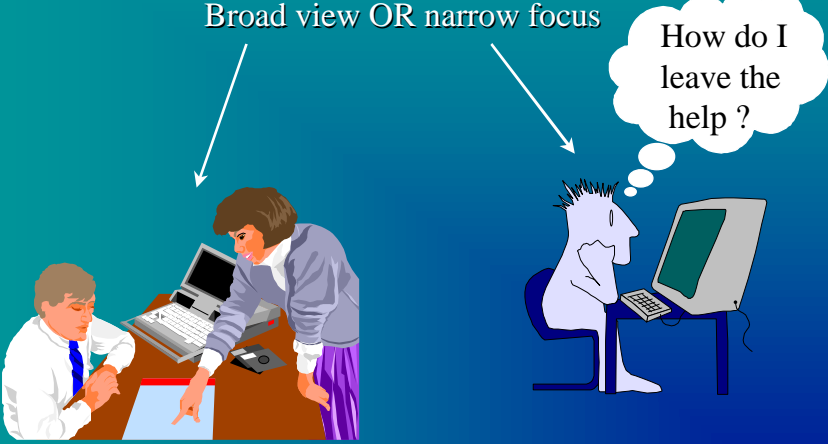
- ATM “...only usable on an overcast day..”
- Video recorder
- Microsoft office: tool bars
 - Microsoft has 17 usability labs
- Not *just* examples: 25% of IT project fail
 - usability is a critical success factor
- traditional communication (attitude) problem
- more often part of testing strategy
- “...usability is subjective and vague....”

Usability definitions

- the **effectiveness, efficiency and satisfaction** with which specified *users* can achieve specified *goals* in particular *environments* (ISO 9241)
- *the capability of the software* to be **understood, learned, used and liked** by the user, when used under specified conditions (ISO 9126)
- not just the property of the user-interface, but a *multiple component* issue associated with the attributes of **learnability, efficiency, memorability, errors and satisfaction** (Nielsen, 1994)

Focus

Broad view OR narrow focus



How do I leave the help ?


“Can users carry out their tasks ?”

5

The slide features a teal background with a dark blue border. At the top left, the word 'Focus' is written in a yellow, serif font. Below it, the text 'Broad view OR narrow focus' is centered. Two white arrows point from this text to two illustrations. The left illustration shows a woman in a purple top leaning over a desk to help a man in a white shirt with a laptop. The right illustration shows a man in a white shirt sitting at a desk with a computer, looking thoughtful. A thought bubble above him contains the text 'How do I leave the help?'. Below the illustrations, the question '“Can users carry out their tasks ?”' is written in a white, italicized font. In the bottom right corner, the number '5' is displayed.

Techniques


- Expert review, cognitive walkthrough, checklist, metrics, process cycle test, use cases, laboratory test
- Problems
 - time, expertise, limited view, user involvement
- Hard to get started
 - is there a problem ?
 - selling usability (testing)



The slide features a teal background with a dark blue border. At the top left, the word 'Techniques' is written in a yellow, serif font. Below it, there is a bulleted list of techniques and problems. The first bullet point is 'Expert review, cognitive walkthrough, checklist, metrics, process cycle test, use cases, laboratory test'. The second bullet point is 'Problems', followed by a sub-bullet '– time, expertise, limited view, user involvement'. The third bullet point is 'Hard to get started', followed by two sub-bullets: '– is there a problem ?' and '– selling usability (testing)'. In the bottom right corner, there is an illustration of a man in a grey suit and red tie, standing with his arms outstretched and a surprised expression.

A possible solution

- Questionnaire based testing
 - good cost / benefit ratio
 - *easy to start with !!*
- Software Usability Measurement Inventory (SUMI)



SUMI

- Broad focus: user satisfaction
- *Well founded* questionnaire: based on practical research (MUSiC)
- Referred to in ISO 9126 and ISO 9241
- The user scores are standardised by using a reference database
- Quantitative, *objective* information of users' *subjective attitude* to six usability aspects

SUMI - metrics

- Affect - user's feeling about interacting with the product
- Efficiency - user's perception of efficiency of task performance
- Helpfulness - view of how communicative the product is (e.g. help, error messages, warnings)
- Control - user's feeling about how the product responds in a normal and consistent manner
- Learnability - how quickly does the user become familiar with the product and the quality of the documentation

9

SUMI - 2

- Intended for use by users
 - with little or no experience of computers
 - doing representative tasks
 - minimum sample of 10 !!
- Running version of software required
 - prototype, test release, operational version
- Supported by an analysis tool
- Specialised version for MultiMedia and Internet type applications

The questionnaire

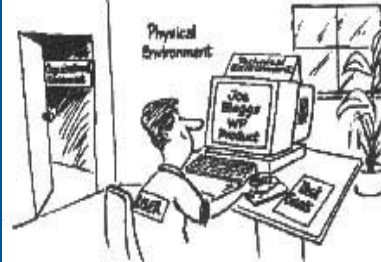
50 questions

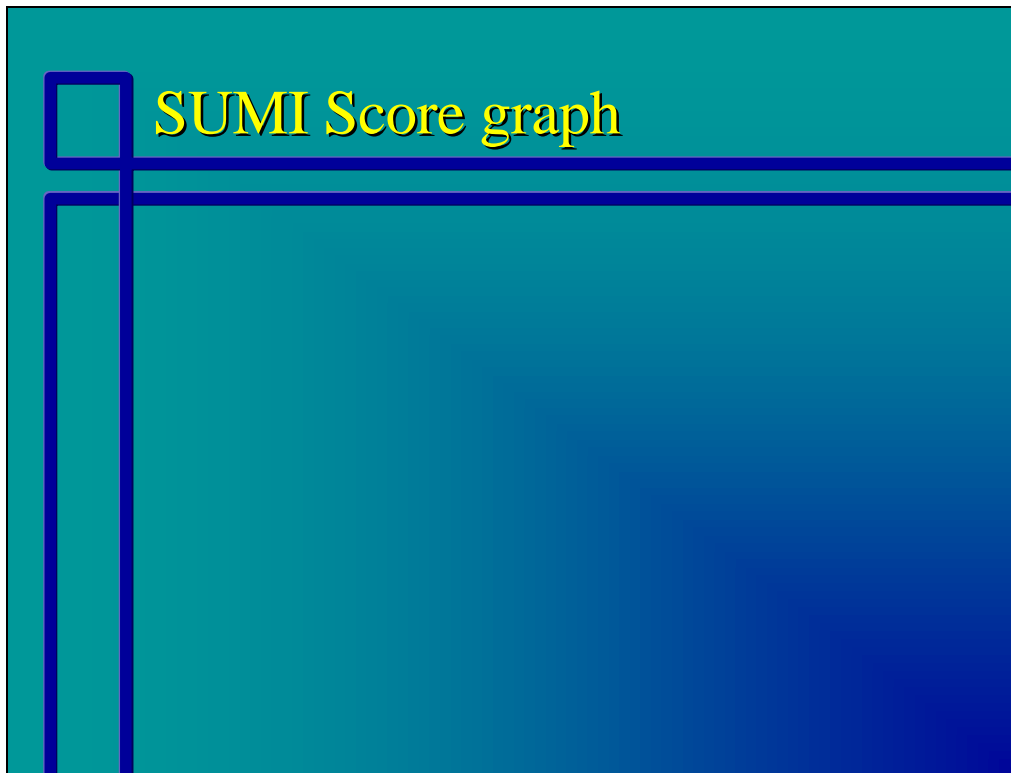
	agree	undecided	disagree
I sometimes don't know what to do next with this software	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It is easy to make the software exactly do what you want	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
It takes too long to learn the software commands	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
The software has a very attractive presentation	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Available in English, German, French, Dutch, Spanish, Italian, Greek and Swedish

Context analysis: User groups


- The usability of a product is affected not only by the features of the product itself but also by its “Context of Use”
- Context is characterized by :
 - the users of the product
 - the task they carry out
 - the working environment
 - ...
 - tool : CoU checklist MUSiC





Output

- Small sample: consider the *entire distribution* and not just the mean value
- *Outliers* are identified
 - user level
 - question level
 - *context analysis !!*
- Score tables
 - per users, per question
 - for analysis purposes



Project 1 : PDM system

- PDM: Product Data Management
- Implementation company wide
- Limited resources, attitude problem
 - usability discussions
- Customer trail project
 - various user groups
- SUMI to compare releases
- Result: (very) low score
 - global : 36 !!

Project 1 continued

- Analysis
 - task efficiency (too many and too difficult steps)
 - messages and help-feature unclear
- Follow-up
 - detailed analysis with users (*time available*)
 - » specialised user-interface tool
 - increase priority outstanding CR's
 - improved information service (*Affect*)
 - re-evaluation planned
 - usability is a steering committee issue now !!


Project 2 : Intranet site

- Intranet site Test services department
- Large Dutch bank
- Usability lab available, experience on SUMI
 - MUMMS applicability test, open questions added
- Users
 - varying in age and background
 - users with / without internet experience
- Results: moderately high all round
 - lot of divergence in users opinion

Project 2 continued

- Analysis
 - not many MM features (attractiveness)
 - efficiency showed difficulty in the structure for some users
- Follow-up
 - change structure of set (more user oriented)
 - add MM features
 - control issues such as menu bar showing where you are
 - re-evaluation planned
 - MUMMS evaluation now part of their services


Cost/benefits process



- Easy-to-use, knowledge:
 - instructions and constraints for using SUMI
 - statistical concepts
- Objective indicator
 - testing tends to focus on defects
- Low costs (3 days)
 - small initial investment needed
- Late in life cycle
- Minimum of 10 user doing the “same” tasks

Cost/benefits results

- *Fast well-founded results !!*
- (limited) detailed analysis possible
 - SUMI metrics, improvement directions, further testing
- Score alone does not tell *what causes* the usability problems, add open questions
- Comparison of products (different versions)
- A great way to start and more
 - *beware any method can be used both well and badly*



Questionnaire based usability testing

Drs. Erik P.W.M. van Veenendaal CISA
(eve@tm.tue.nl)

Usability is an important aspect of software products. However, in practice not much attention is given to this issue during testing. Testers often do not have the knowledge, instruments and/or time available to handle usability. This paper introduces the Software Usability Measurement Inventory (SUMI) testing technique as a possible solution to these problems. SUMI is a rigorously tested and validated method to measure software quality from a user perspective. Using SUMI the usability of a software product or prototype can be evaluated in a consistent and objective manner. The technique is supported by an extensive reference database and embedded in an effective analysis and reporting tool.

SUMI has been applied in practice in a great number of projects. This paper also deals with some practical applications. The results, usability improvements, cost and benefits are described. Conclusions are drawn regarding the applicability and the limitations of SUMI for usability testing.

A closer look at usability

Several investigations have shown that in addition to functionality and reliability, usability is a very important success factor (Moolenaar and Van Veenendaal,1997). Sometimes it is possible to test the software extensively in a usability lab environment. However, in most other situations a usability test has to be carried out with minimum resources.

The usability of a product can be tested from different perspectives. Quite often the scope is limited to "ease-of-use". The "ease-of-use" is mainly determined by characteristics of the software product itself, such as the user-interface. Within this type of scope usability is part of product quality characteristics. The usability definition of ISO 9126 is an example of this type of perspective:

<p><i>Usability</i> the capability of the software to be understood, learned, used and liked by the user, when used under specified condition (ISO 9126-1,1998)</p>

Two techniques that can be carried out at reasonable costs evaluating the usability product quality, are expert reviews and checklists. However, these techniques have the disadvantage that the real stakeholder, e.g. the user, isn't involved.

In a broader scope usability is being determined by using the product in its (operational) environment. The type of users, the tasks to be carried out, physical and social aspects that can be related to the usage of the software products are taken into account. Usability is being defined as "quality-in-use". The usability definition of ISO 9241 is an example of this type of perspective:

<p><i>Usability</i> the extent to which a product can be used by specified users to achieve goals with effectiveness, efficiency and satisfaction in a specified context of use (ISO 9241-11,1996)</p>
--

Clearly these two perspective of usability are not independent. Achieving “quality-in-use” is dependent on meeting criteria for product quality. The interrelationship is shown in figure 1.

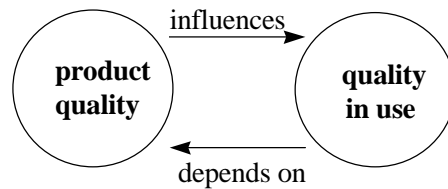


Figure 1 : Relationship between different types of usability

Establishing test scenarios, for instance based on use cases (Jacobson,1992), can be applied to test usability in accordance with ISO 9241. However, usability testing with specified test cases / scenarios is a big step for most organization and often not even necessary. From a situation where usability is not tested at all one wants a technique that involves users, is reliable but still requires limited resources.

Within the European ESPRIT project MUSiC [ESPRIT 5429] a method has been developed that serves to determine the quality of a software product from a user’ perspective. Software Usability Measurement Inventory (SUMI) is a questionnaire based method that can be designed for cost effective usage .

What is SUMI?

Software Usability Measurement Inventory (SUMI) is a solution to the recurring problem of measuring users' perception of the usability of software. It provides a valid and reliable method for the comparison of (competing) products and differing versions of the same product, as well as providing diagnostic information for future developments. It consists of a 50-item questionnaire devised in accordance with psychometric practice. Each of the questions is answered with "agree", "undecided" or "disagree". The following sample shows the kind of questions that are asked:

- This software responds too slowly to inputs
- I would recommend this software to my colleagues
- The instructions and prompts are helpful
- I sometimes wonder if I am using the right command
- Working with this software is satisfactory
- The way that system information is presented is clear and understandable
- I think this software is consistent.

The SUMI questionnaire is available in English (UK and US), French, German, Dutch, Spanish, Italian, Greek and Swedish.

SUMI is intended to be administered to a sample of users who have had some experience of using the software to be evaluated. In order to use SUMI effectively a minimum of ten users is recommended. Based on the answers given and statistical concepts the usability scores are being calculated. Of course SUMI needs a working version of the software before SUMI can be measured. This working version can also be a prototype or a test release.

One of the most important aspects of SUMI has been the development of the standardization database, which now consists of usability profiles of over 2000 different kinds of applications.

Basically any kind of application can be evaluated using SUMI as long as it has user input through keyboard or pointing device, display on screen, and some input and output between secondary memory and peripheral devices. When evaluating a product or series of products using SUMI, one may either do a product-against-product comparison, or compare each product against the standardization database, to see how the product that is being rated compares against an average state-of-the-market profile.

SUMI gives a global usability figure and then readings on five subscales:

- *Affect*: how much the product captures the user's emotional responses
- *Control*: degree to which the user feels he, and not the product, is setting the pace
- *Efficiency*: degree to which the user can achieve the goals of his interaction with the product
- *Helpfulness*: extent to which the product seems to assist the user
- *Learnability*: ease with which a user can get started and learn new features of the product.

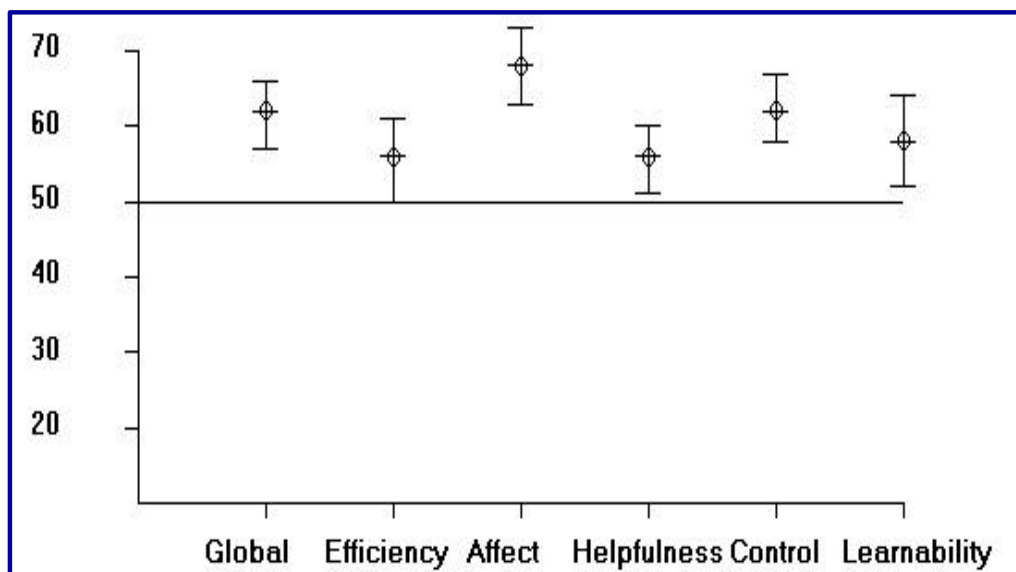


Figure 2: a sample profile showing SUMI scales

Figure 2 shows an example of SUMI output; it shows the scores of a test and the spreading of these scores (measured by the standard deviation) against the average score of the reference database, reflected by the value 50. Consequently the usability scores shown in the sample profile are positive, e.g. more than state-of-the-art, with a reasonable level of spreading.

SUMI is the only available questionnaire for the assessment of usability of software, which has been developed, validated and standardized on a European wide basis. The SUMI subscales are being referenced in international ISO standards on usability (ISO 9241-10,1994) and software product quality (ISO 9126-2,1997). Product evaluation with SUMI provides a clear and objective measurement of users' view of the suitability of software for their tasks.

Recently a specialized version of SUMI to be used for multimedia products has been developed: MUMMS (Measuring Usability of Multi Media Systems).

Any SUMI test must be carried out by asking people that perform realistic, representative tasks. Employing a method such as usability context analysis (NPL,1995) helps identify and specify in a systematic way the characteristics of the users, the tasks they will carry out, and the circumstances of use. Based on the results the various user groups can be described and

used to define how these user groups can be represented in the test.

Practical Applications

Project 1: Project Management Package

Approach

Subject to the usability evaluation by means of SUMI was a software package offering project administration and control functionality. The software package is positioned as a multi-project system for controlling the project time, e.g. in terms of scheduling and tracking, and managing the productivity of projects, e.g. in terms of effort and deliverables. The package has developed by a Dutch software house that specializes in the development of standard software packages.

The SUMI test was part of an acceptance test carried out on behalf of a potential customer. Due to the very high number of users, a number of different user groups, their inexperience with project management software and the great variety of information needs, usability was an important characteristic. It was even looked upon as the critical success factor during implementation. Two main user group were distinguished. One user group was mainly involved in input processing of effort and time spent. For this user group especially operability and efficiency is of great importance. Another user group was characterized as output users. Especially receiving the right management information is important for the output users. Per user group a SUMI test has been carried out.

Regarding the usage of the SUMI technique for the usability evaluation a specific acceptance criteria was applied. SUMI provides quantitative values relating to a number of characteristics that lead to a better understanding of usability. As part of the acceptance test, the SUMI scale was used that provides an overall judgement of usability, the so-called "global scale". Based on the data in the SUMI database, it can be stated that the global score has an average value of 50 in a normal distribution. This means that by definition for a value exceeding 50 the user satisfaction is higher than average. In the test of the project management package the acceptance criteria applied that for each user group the global scale and the lower limit of the 95% confidence interval must both exceed the value of 50.

Results

The "global scale" regarding both user groups was below the desired 50. For the input user group the score was even a mere 33. The output user group showed a slightly better score. Not only the "global scale" but also most other subscales were scoring below 50.

Because the results did not meet the acceptance criteria that were set a number of usability improvement measures needed to taken. Examples of measures that were taken based on the results of the SUMI test are:

- extension and adaptation of the user training
- optimization of efficiency for important input functions
- implementation of specific report generation tools for the output user with a clear and understandable user-interface.

Project 2: Intranet site

Approach

By means of MUMMS, the specialized multimedia version of SUMI, the usability of an intranet site prototype of a large bank was evaluated. The intranet site was set up by the test services department to get well-known and to present themselves to potential customers. The

fact that during the test only a prototype version of the intranet site was available meant that some pages were not yet accessible. For MUMMS a special subscale has been introduced, with the objective to measure the users' multimedia “feeling”:

- *Excitement*: extent to which end-users feel that they are “drawn into” the world of the multimedia application.

In total ten users (testers) were involved in the MUMMS evaluation. The set of users can be characterized by:

- not having been involved during the development of the intranet site
- potential customers
- four users with internet experience
- six users without internet experience
- varying by age and background (job title).

Results

The table below shows the overall scores for the various MUMMS subscales:

	Affect	Control	Efficiency	Helpfulness	Learnability	Excitement
average score	69	74	62	67	67	68
median	71	77	67	69	67	72
standard deviation	9	12	11	8	6	12

Table 1: Overall MUMMS score table

The various scores were moderately high all round. However, there seems to be a divergence of opinion on the control and excitement scales. Some low scores are pulling down the control and efficiency scales (see next table). Two users from the sample were giving exceptionally low average scores. They were analyzed in detail but no explanation was found.

	A	C	E	H	L	E	Average
User 1	71	81	67	71	74	77	73
User 2	74	74	74	71	67	71	72
User 3	81	84	67	67	74	74	74
User 4	54	51	54	57	64	44	54
User 5	71	74	43	58	55	76	63
User 6	64	84	67	81	67	69	72
User 7	51	81	74	54	74	64	66
User 8	71	81	64	74	71	81	73
User 9	77	81	76	84	77	74	78
User 10	64	47	51	57	57	44	53

Table 2: MUMMS scores per user

Results

As stated the usability of the Intranet site was rated moderately high from the users’ perspective, although there seemed to be a lot of divergence in the various user opinions. Some more detailed conclusion were:

- *Attractiveness*
The attractiveness score is high (almost 70%). However some users (4, 7 and 10) have a

relatively low score. Especially the questions “this MM system is entertaining and fun to use” and “using this MM system is exiting” are answered in different ways. It seems some additional MM features should be added to further improve the attractiveness for all users.

- *Control*
A very high score for control in general. Again two users can be identified as outlayers (4 and 10) scoring only around 50%, the other scores are around 80%. Problems, if any, in this area could be traced back to the structure of the site.
- *Efficiency*
The average score on efficiency is the lowest, although still above average. Users need a more time than expected to carry out their task, e.g. find the right information.

On the basis of the MUMMS evaluation it was decided to improve the structure of the internet site and to add a number of features before releasing the site to the users. Currently the update of the intranet site is being carried out. A MUMMS re-evaluation has been planned to quantify the impact of the improvement regarding usability.

Applicability of SUMI

On the basis of the test carried out in practice, a number of conclusions have been drawn regarding the applicability of SUMI and MUMMS:

- it is easy to use; not many costs are involved. This applies both to the evaluator and the customer. On average a SUMI test can be carried in approximately 2 to 3 days; this includes the time necessary for the mini context analysis and reporting;
- during testing the emphasis is on finding defects, this often results in a negative quality indications. SUMI however, provides an objective opinion;
- the usability score is split into various aspects, making a thorough more detailed evaluation possible (using the various output data);
- MUMMS provides, after detailed analysis and discussion, directions for improvement and directions for further investigation. SUMI can also be used to determine whether a more detailed usability test, e.g. laboratory test, is necessary.

However, also some disadvantages can be distinguished:

- a running version of the system needs to be available; this implies SUMI can only be carried at a relatively late stage of the project;
- the high (minimum of ten) number of users with the same background, that need to fill out the questionnaire. Quite often the implementation or test doesn't involve ten or more users belonging to the same user group;
- the accuracy and level of detail of the findings is limited (this can partly be solved by adding a number of open question to the SUMI questionnaire);

Conclusions

It has been said that a system's end users are *the* experts in using the system to achieve goals and that their voices should be listened to when that system is being evaluated. SUMI does precisely that: it allows quantification of the end users' experience with the software and it encourages the tester to focus in on issues that the end users have difficulty with. Evaluation by experts is also important, but it inevitably considers the system as a collection of software entities.

A questionnaire such as SUMI represents the end result of a lot of effort. The tester get the result of this effort instantly when SUMI is used: the high validity and reliability rates reported for SUMI are due to a large measure to the rigorous and systematic approach adopted in

constructing the questionnaire and to the emphasis on industry-based testing during development. However, as with all tools, it is possible to use SUMI both well and badly. Care taken over establishing the context of use, characterizing the end user population, and understanding the tasks for which the system will be used supports sensitive testing and yields valid and useful results in the end.

Literature

- Bevan, N. (1997), Quality and usability: a new framework, in: E. van Veenendaal and J. McMullan (eds.), *Achieving Software Product Quality*, Tutein Nolthenius, 's Hertogenbosch, The Netherlands
- Bos, R. and E.P.W.M. van Veenendaal (1998), For quality of Multimedia systems: The MultiSpace approach (in Dutch), in: *Information Management*, May 1998
- ISO/IEC FCD 9126-1 (1998), *Information technology - Software product quality - Part 1 : Quality model*, International Organization of Standardization
- ISO/IEC PDTR 9126-2 (1997), *Information technology - Software quality characteristics and metrics - Part 2 : External metrics*, International Organization of Standardization
- ISO 9421-10 (1994), *Ergonomic Requirements for office work with visual display terminals (VDT's) - Part 10 : Dialogue principles*, International Organization of Standardization
- ISO 9241-11 (1995), *Ergonomic Requirements for office work with visual display terminals (VDT's) - Part 11 : Guidance on usability*, International Organization of Standardization
- Jacobson, I. (1992), *Object Oriented Software Engineering; A Use Case Driven Approach*, Addison Wesley, ISBN 0-201-54435-0
- Kirakowski, J., The Software Usability Measurement Inventory: Background and Usage, in: *Usability Evaluation in Industry*, Taylor and Francis
- Kirakowski, J. and M. Corbett (1993), SUMI: the Software Usability Measurement Inventory, in: *British Journal of Educational Technology*, Vol. 24 No. 3 1993
- Moolenaar, K.S. and E.P.W.M. van Veenendaal (1997), *Report on demand oriented survey*, MultiSpace project [ESPRIT 23066]
- National Physical Laboratory (NPL) (1995), *Usability Context Analysis: A Practical Guide*, version 4.0, NPL Usability Services, UK
- Preece, J. *et al*, *Human-Computer Interaction*, Addison-Wesley Publishing company
- Tienekens, J.J.M. and E.P.W.M. van Veenendaal (1997), *Software Quality from a Business Perspective*, Kluwer Bedrijfsinformatie, Deventer, The Netherlands

The Author

Drs. Erik P.W.M. van Veenendaal CISA has been working as a practitioner and manager within the area of software quality for a great number of years. Within this area he specializes in testing and is the author of several books, e.g. "Testing according to TMap" (in Dutch) and "Software Quality from a Business Perspective". He is a regular speaker both at national and international testing conferences and a leading international trainer in the field of software testing. Erik van Veenendaal is the founder and managing director of Improve Quality Services. Improve Quality Services provides services in the area of quality management, usability and testing.

At the Eindhoven University of Technology, Faculty of Technology Management, he is part-time involved in lecturing and research on information management, software quality and test management. He is on the Dutch standards institute committee for software quality.

Slide 1

ESI


B I G
Business-driven Improvement Guide

**STAGED MODEL FOR SPICE:
HOW TO REDUCE TIME TO MARKET**

QWE-98 © ESI 1998

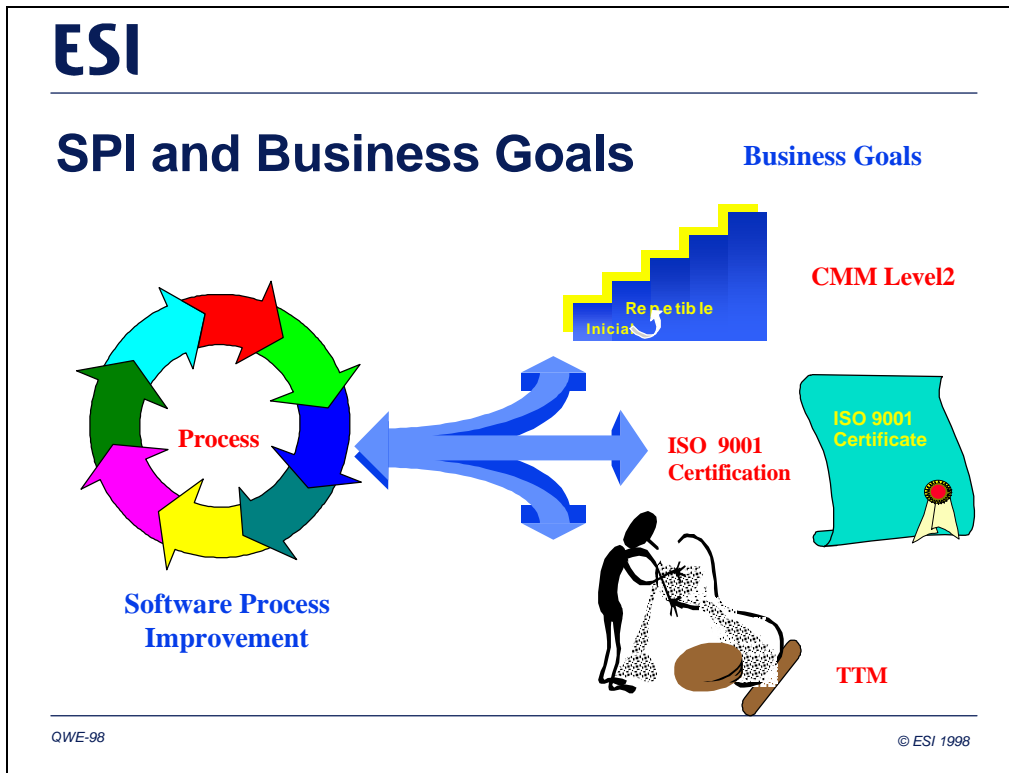
Slide 2

ESI

Objectives of the Paper

- Present a **new guide, BIG-TTM**, as a part of ESI's BIG series.
- Explain the development of BIG-TTM based on BIG series **development method**.
- Describe the most **relevant factors** for reducing **Time to Market** of software products from ESI's point of view.
- Introduce **IMPACT, general improvement cycle** used as a framework of the BIG guides

QWE-98 © ESI 1998



ESI

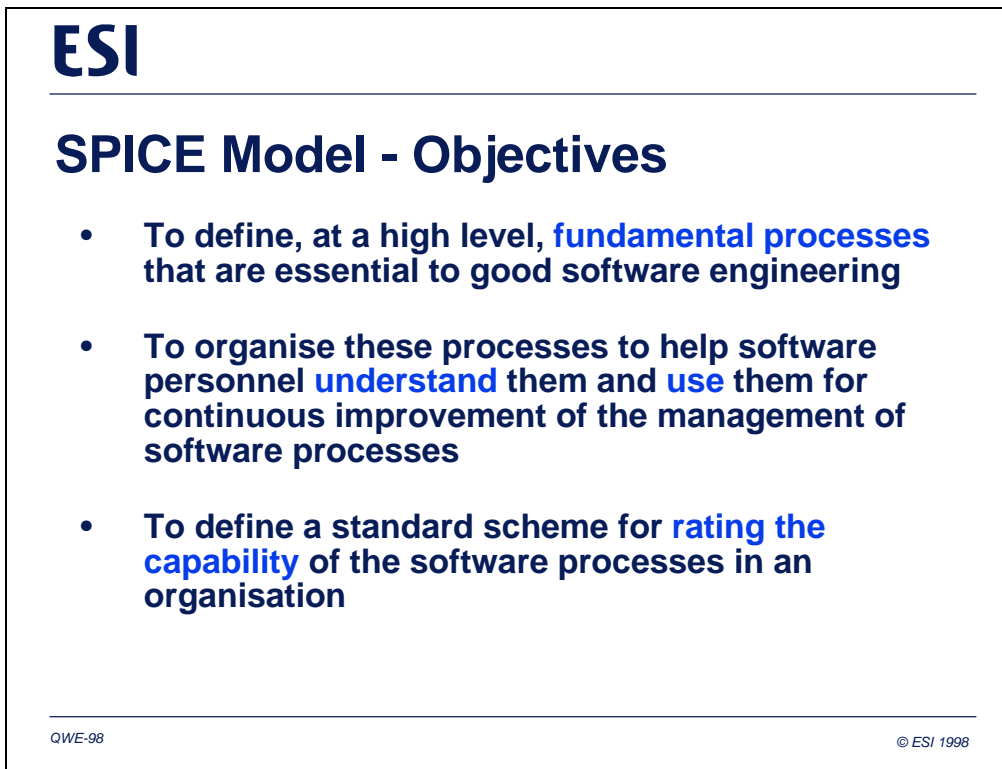
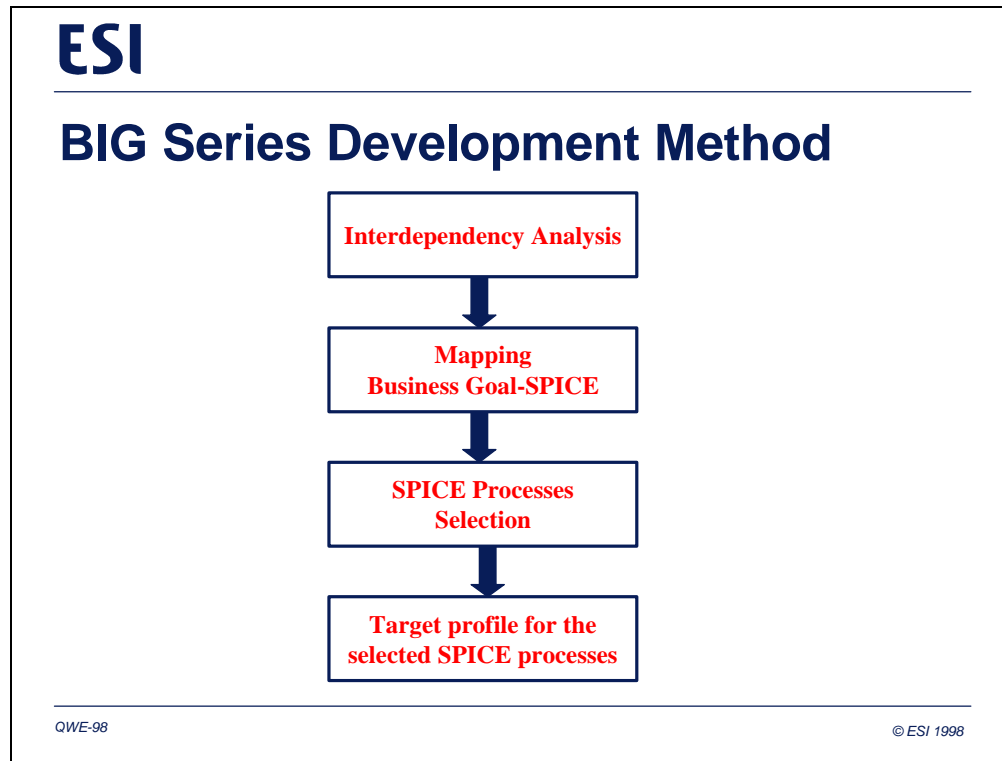
BIG Series

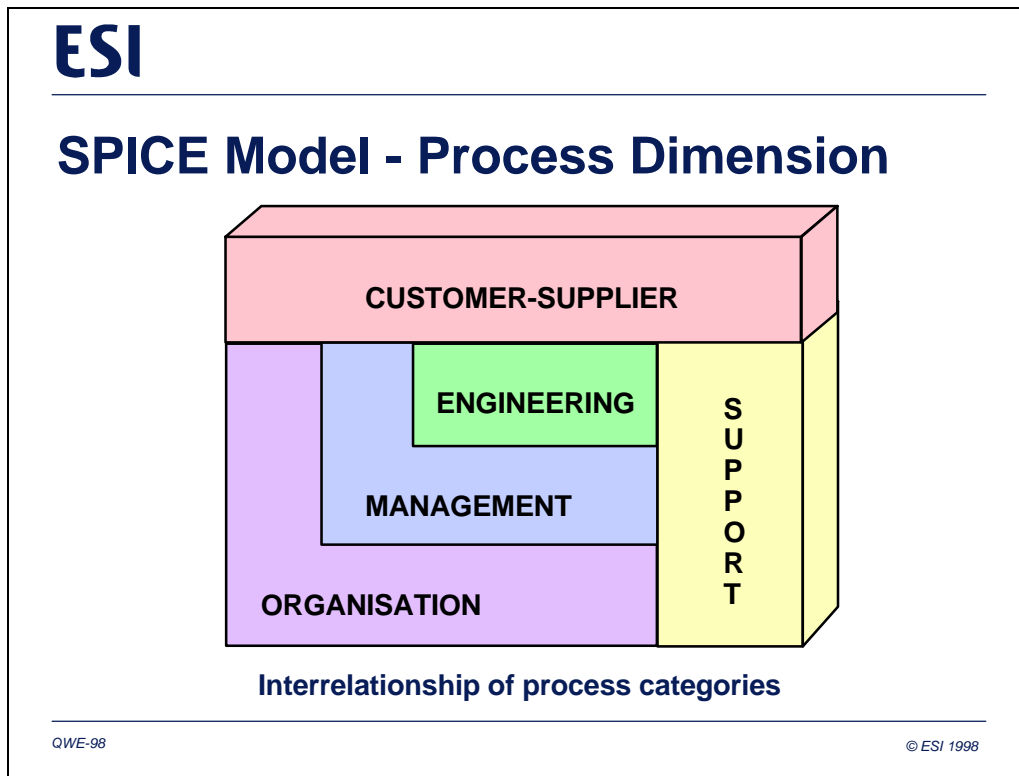
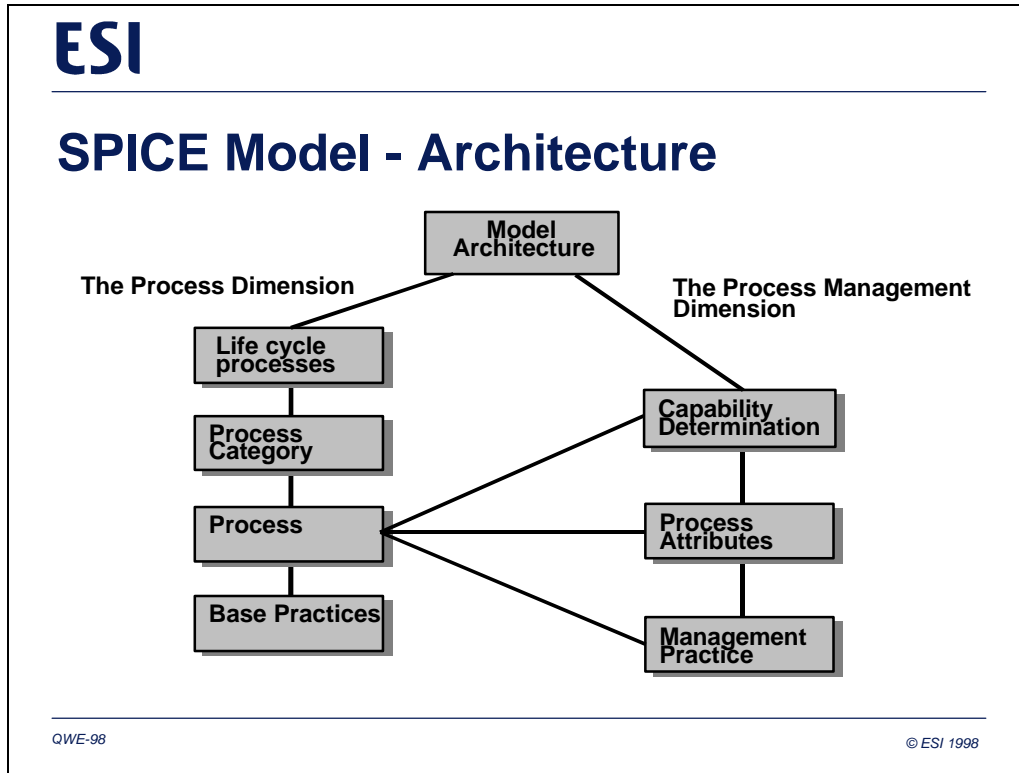
Common features of BIG products:

- Derive an **improvement plan** to achieve an explicit Business Goal.
- Software Process Improvement plan based on the **SPICE model**.

ESI's BIG Series	Business Goal
• BIG-ISO9001	ISO 9001 certification
• BIG-TTM	Reduce Time to Market
• BIG-CMM	Achieve CMM Level 2

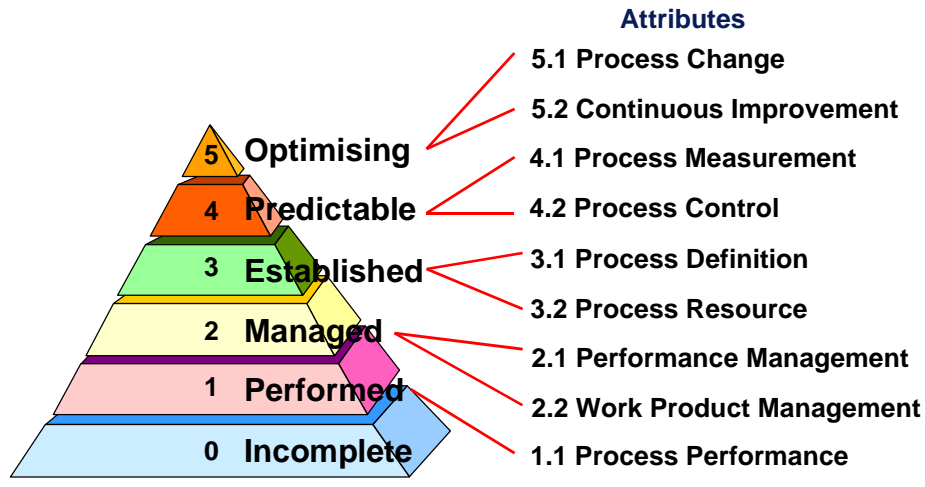
QWE-98 © ESI 1998





ESI

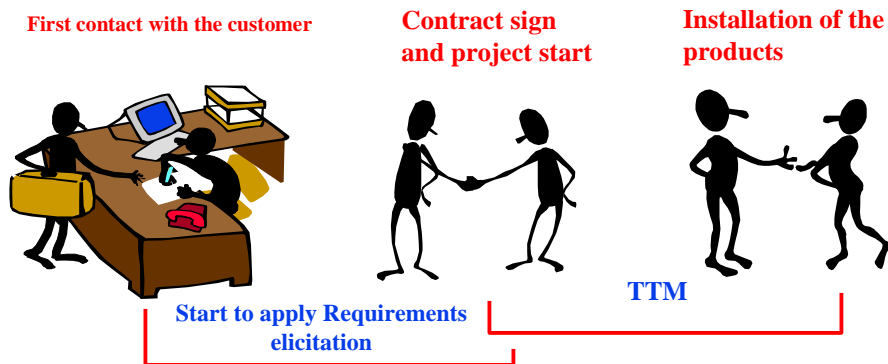
SPICE Model - Capability Dimension

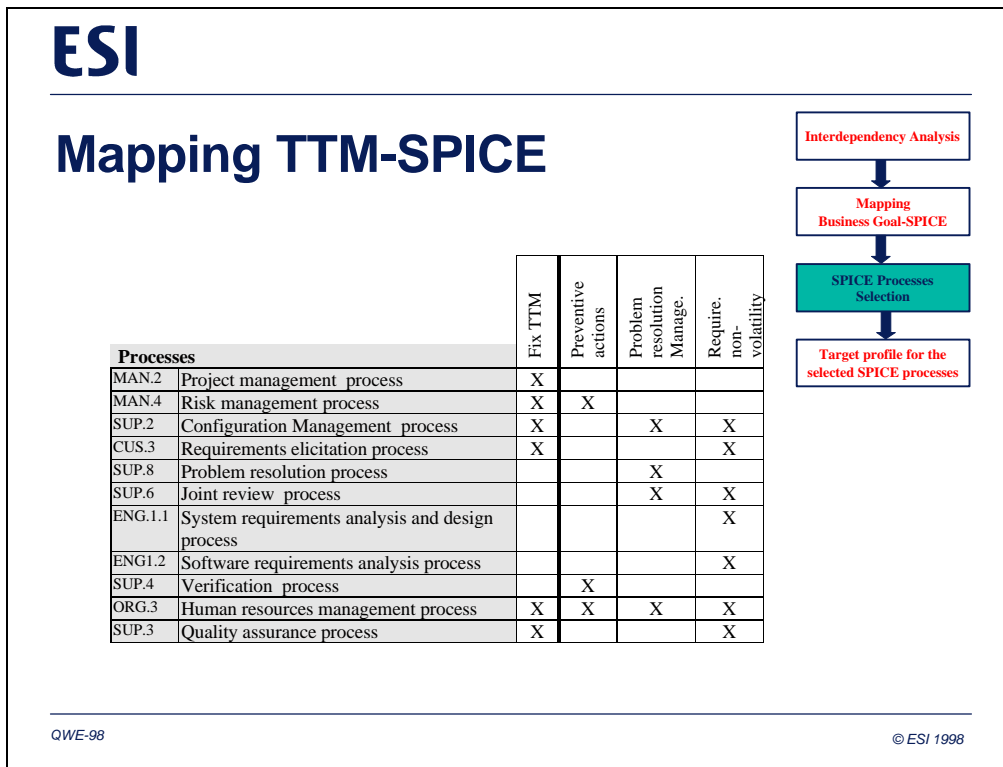
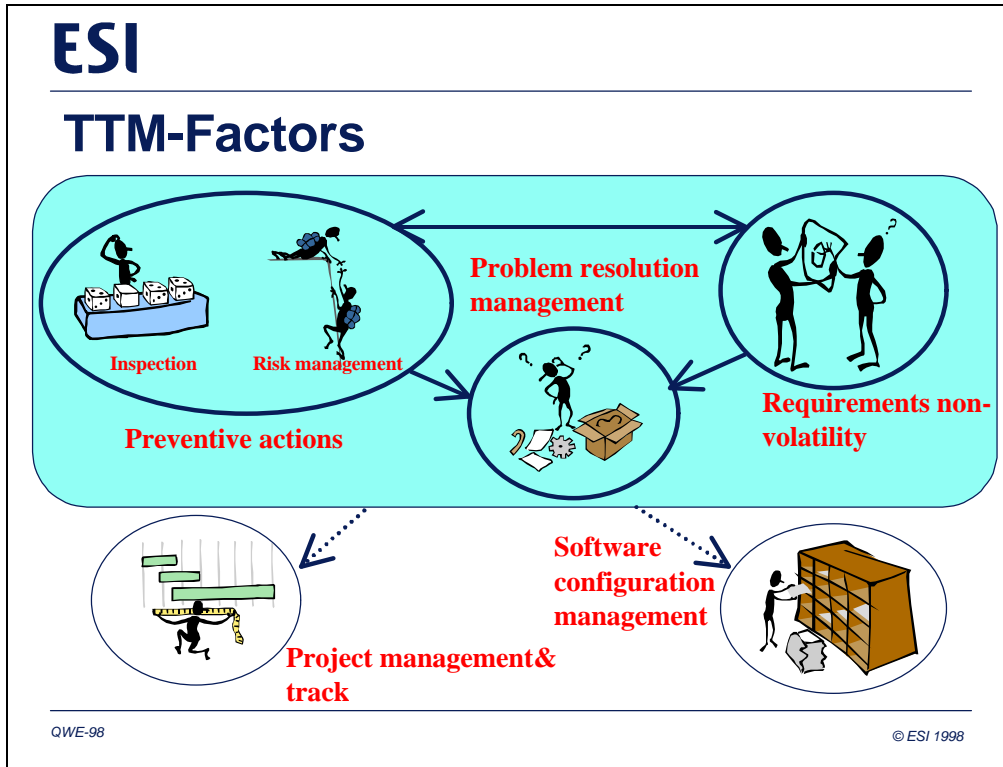


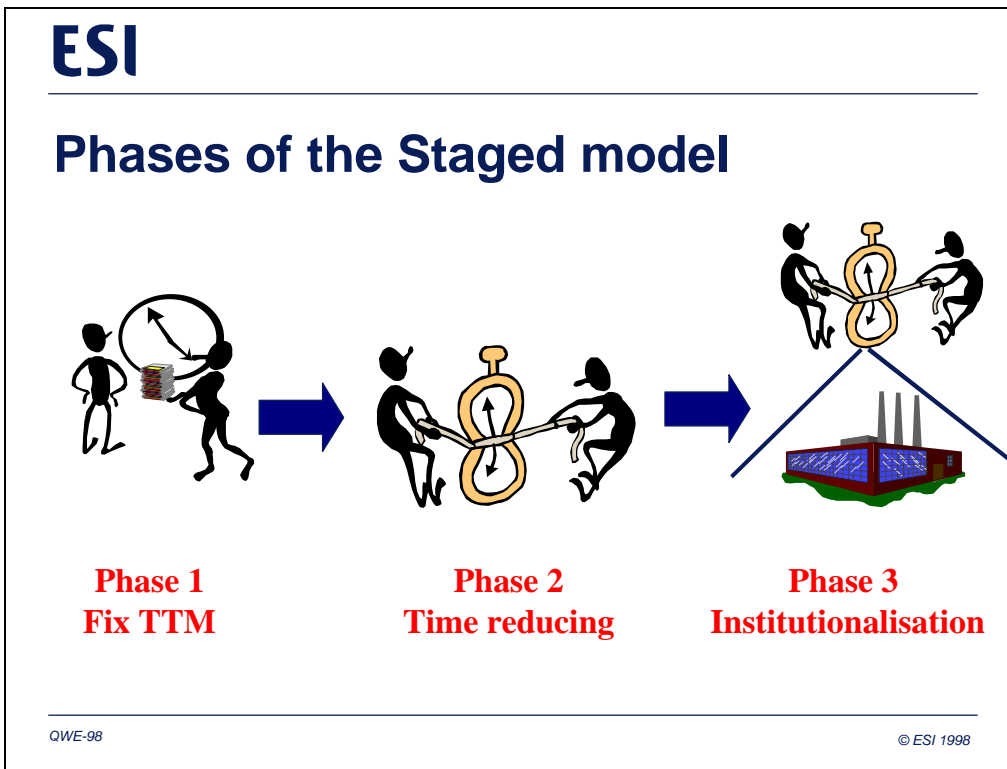
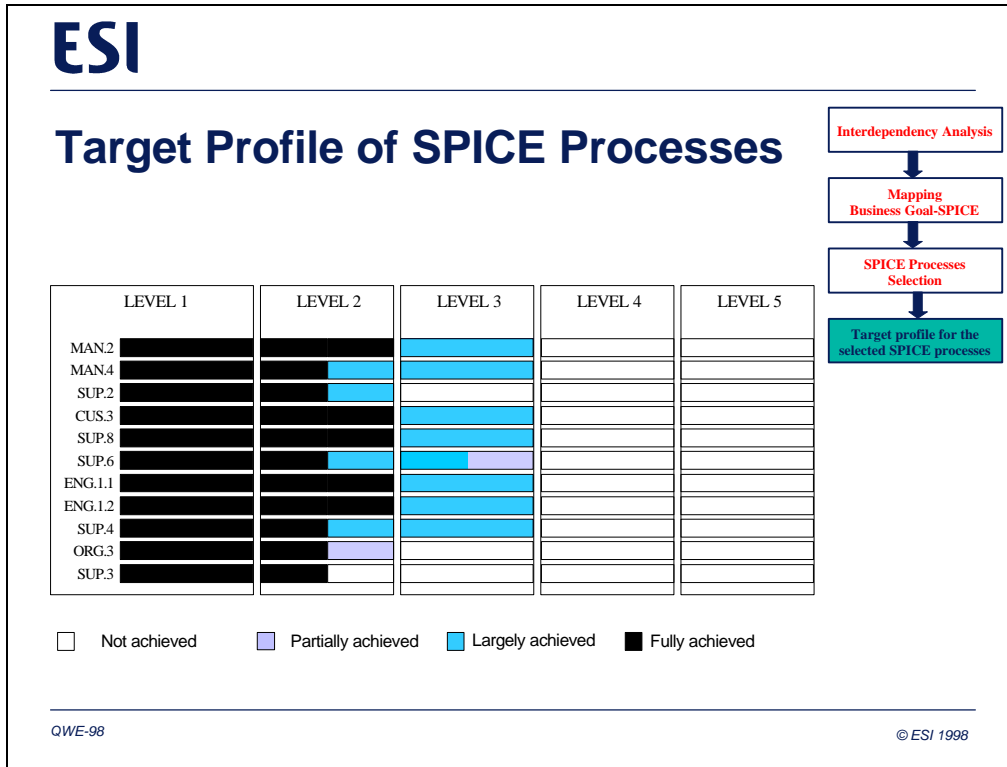
ESI

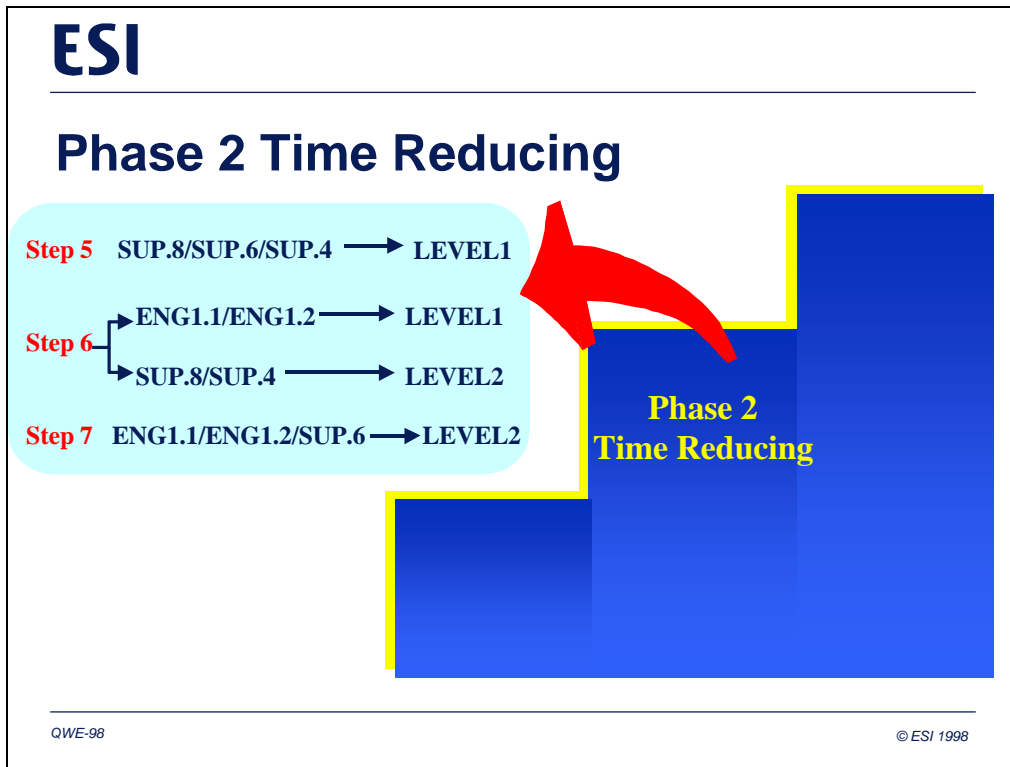
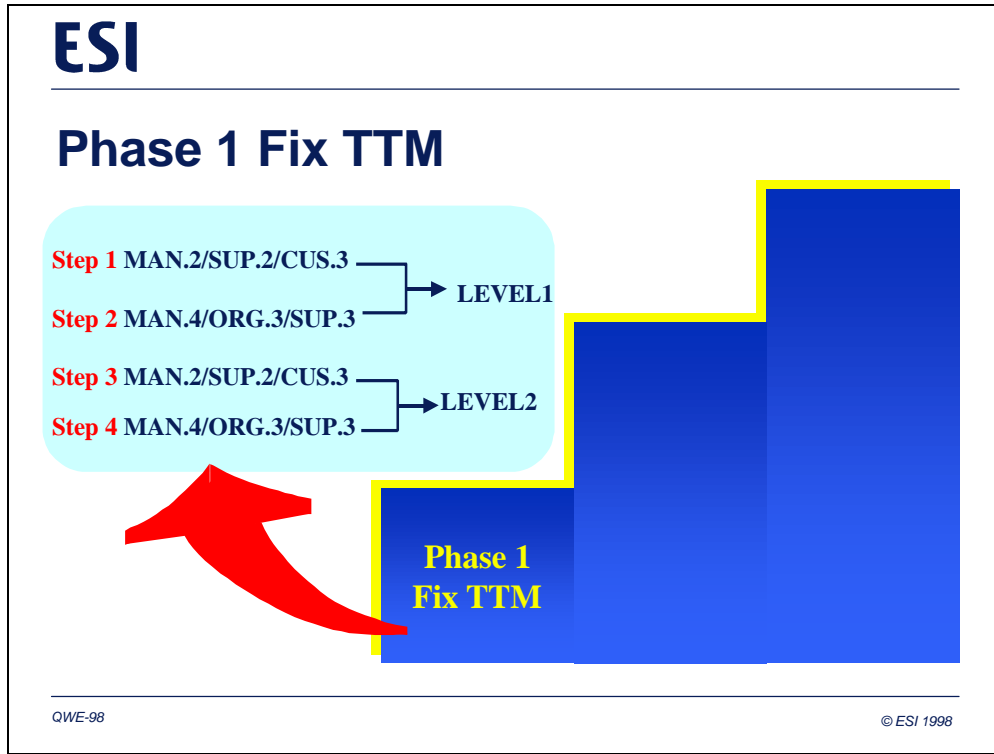
TTM - Definition

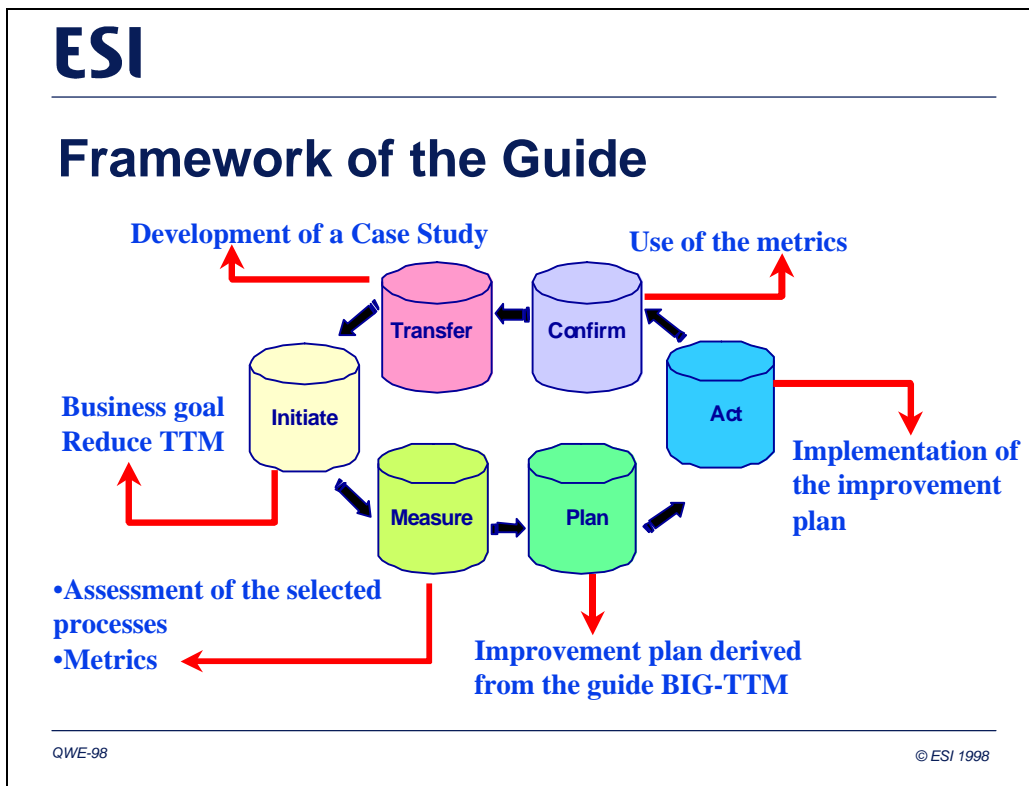
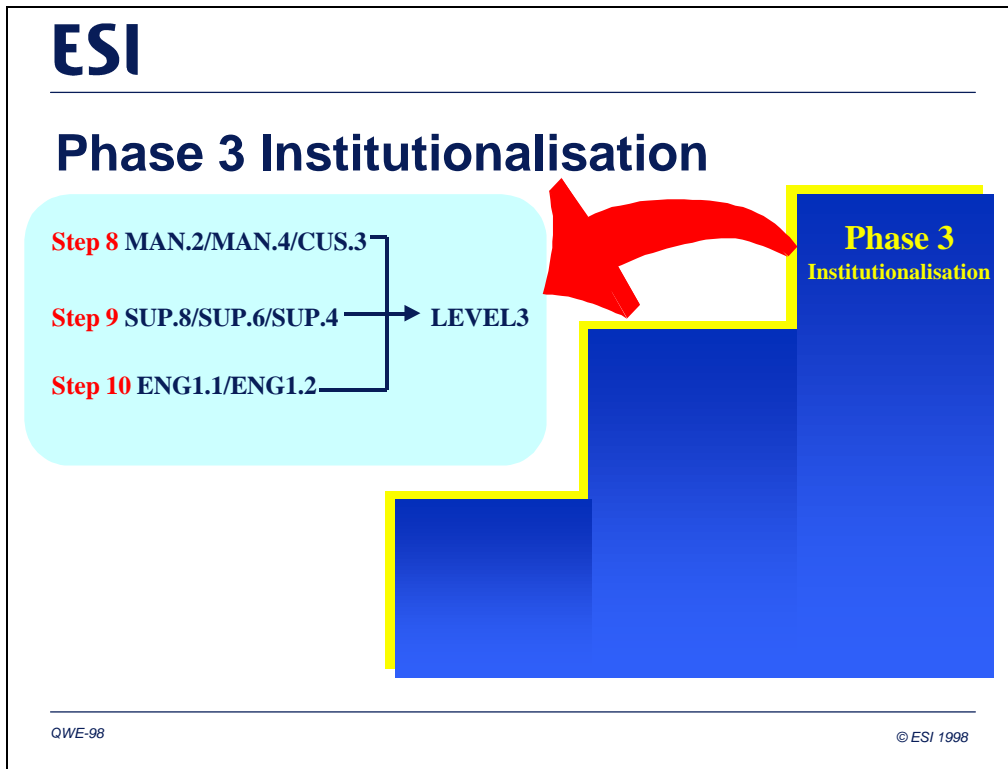
Time elapsed between the decision to **start a project** (contract award) and the **installation of the products** at the client premises











ESI

Summary

- **BIG-TTM guide as a product of ESI's BIG series**
- **TTM as explicit business goal for the BIG-TTM improvement plan and, factors relevant to TTM**
- **IMPACT as BIG-TTM plan framework**

Staged Model for SPICE: how to reduce Time to Market (TTM)

Gorka Benguria (gorka@esi.es), M^a Luisa Escalante (marisa@esi.es),
M^a Elisa Gallo (melisa@esi.es), Elixabete Ostolaza (elixabete@esi.es),
Mikel Vergara (mikel@esi.es), Ana Andrés

European Software Institute
Parque Tecnológico, Edificio 204
E-48170 Zamudio - Bizkaia SPAIN
Tel.: ++34-94 420 95 19; Fax: ++34-94 420 94 20

Staying competitive by shortening the development life cycle of software products while still guaranteeing product quality is one of the main concerns of software organisation units.

In recent months, the European Software Institute (ESI) has been working on a series of Business Improvement Guides (BIG- products) that clearly express the relationship between software process improvement and the achievement of specific business goals.

This paper describes the development of one of these improvement guides (BIG-TTM), based on the business goal 'time-to-market (TTM)' and, an improvement cycle where the improvement plan deduced from the guide will be embedded.

Keywords: time to market, software process improvement, SPICE, staged model, business goal

Introduction

The relationship between software process improvement and the achievement of business goals is not always obvious. When defining an improvement plan for a software organisation based on an improvement model such as SPICE, ESI's experience shows the need to focus attention on certain assumptions before deciding what activities to include in the plan and the sequence of those activities. To address this issue, ESI is developing a series of Business Improvement Guides (BIG-products).

The BIG products aim to facilitate the definition of improvement plans focused on the fulfilment of specific business goals. They are based on staged models and are mainly addressed to small to medium sized enterprises (SMEs). The improvement plans derived from the BIG guides are based on the SPICE standard process improvement model.

SPICE (ISO/IEC TR 15504) is a continuous model for process improvement that can be used as a basis for laying down simple, clear and sequential steps for an improvement plan. An added advantage is that SPICE is an ISO technical report, published as a fully operative standard in September 1998.

The first guide of the series was the BIG-ISO 9001 guide, focused on the achievement of ISO 9001 certification. The development method followed by the BIG guides is described in *figure 1*.

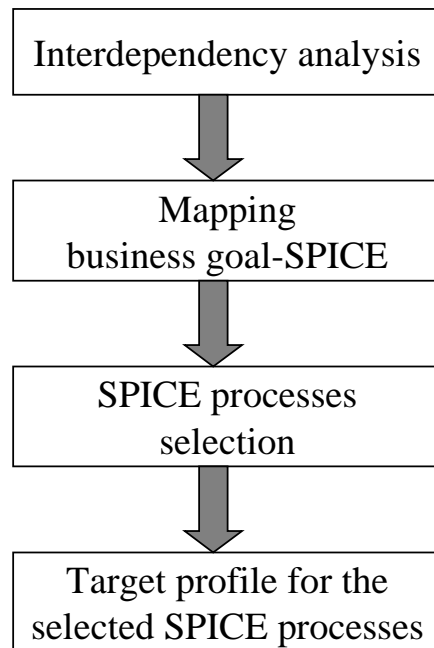


Figure 1 Development method

SPICE model

SPICE is a wide framework containing best practices for software engineering. It can be adapted and tailored to almost any shape. Its malleability makes it an excellent choice for SMEs. SPICE is a practical model that deals with technical, managerial and organisational issues in software industry.

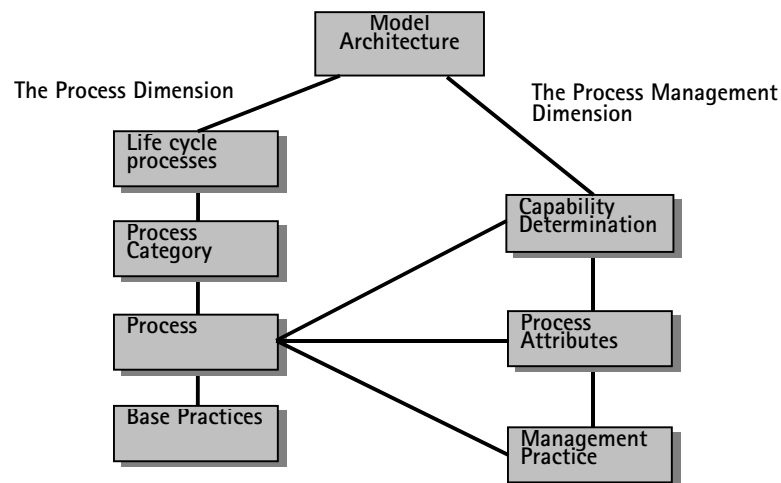


Figure 2 SPICE model architecture

Structure of the SPICE model

The SPICE model architecture is made up of two dimensions: process and process capability.

The process dimension.

The SPICE model groups the processes of this dimension into three life cycle process groupings, containing five process categories, according to the type of activity they address. The **primary life cycle processes** consist of the *customer-supplier and the engineering category*. The customer-supplier category includes processes that have a direct impact on the customer, covering development and transition of the software to the customer, and supporting the correct operation and use of the software product and/or service. The *engineering category* is made up of processes that directly specify, implement or maintain the software product, its relation to the system and its customer documentation. The **supporting life cycle processes** consist of *support category* processes, which may be employed by any of the other processes (including other supporting processes) at various points in the software life cycle. The **organisational life cycle processes** consist of the *management and organisation categories*. The management category involves processes which contain practices of a generic nature that may be used by anyone who manages any type of project or process within a software life cycle. The organisation category covers the processes that establish the business goals of the organisation and that develop process, product and resource assets to meet these goals.

Process capability dimension.

Evolving process capability is expressed in terms of process attributes grouped into capability levels. Process attributes are features of a process that can be evaluated on a scale of achievement, providing a measure of the capability of the process. They are applicable to all processes. Each process attribute describes a facet of the overall capability of managing and improving the effectiveness of a process in achieving its purpose and contributing to the business goals of the organisation.

A capability level is characterised by a set of attribute(s) that work together to provide a major enhancement in the capability to perform a process. There are six capability levels in the SPICE model ranging from 0 to 5, to indicate the level of the maturity of the processes according to their compliance with the attributes associated with levels. Each level adds a formality and rigour in the way activities are performed until level 5 is reached, when there is a continuous improvement of the processes.

Time to Market as a business goal

The business goal of BIG-TTM, new step in the series of BIG-products, is time-to-market (TTM). The definition of TTM in the context of this guide is the time elapsed between the decision to start the project (usually contract award) and the installation of the product at the client premises. Nevertheless, to be accepted a contract usually needs to attach a project schedule based on high level requirements. In order to avoid significant deviations between the contract offer and the first consistent schedule once a detailed requirements version is available, appropriate requirements elicitation techniques and methods must be used. This is why, the

processes involved in the improvement plan deduced from the BIG-TTM guide only affect projects once they have officially started, with the elicitation process as the only exception to the rule.

This specific business goal, TTM, has been chosen due to its criticality in the software industry since timely deliveries are the basis for customer satisfaction which affects financial results. The ability to deliver what was planned, when it was planned, is one of the challenges faced by software development companies. Reducing the time to delivery while maintaining the quality of software products, is a step forward which allow companies to be more competitive.

Time to Market most relevant factors

As with any complex problem, optimising TTM does not have a clean cut, one-size-fits-all solution. There are many reasons why software is late and many are interrelated. Over-optimistic planning, underestimation of project complexity or poor management are some of the reasons claimed. Studies [1] show that more than technology, what mainly affects time to delivery is organisational, managerial or human factors. Therefore it is important to shift the focus from technology oriented improvements to management and process improvement. This is what the BIG-TTM model provides, along with a guide for its implementation.

In the BIG-TTM guide, ESI has selected a set of the TTM most relevant factors based on:

1. The Insead survey [1]. Analysis by INSEAD, a leading European management school in Fontainebleau (France) of the results of a survey around 100 large European software intensive organisations. The survey asked respondents questions about management practices used in successfully completed software projects.
2. ESI Repository of Experience. Search of VASIE repository on experiences specifically related to TTM. This repository contains the results of Process Improvement Experiences conducted by European industry under the European Commission; funded ESSI program.
3. Results of MEPROS. MEPROS is an ESI project that defines a guide containing a set of indicators for measuring the level of achievement of business objectives based on Scorecard methodology. For the business goal TTM, the indicators identified by MEPROS measure the most relevant factors as outlined below.

Most relevant TTM factors

- Requirements non-volatility. The requirement analysis phase of the systems development life cycle is very time consuming. The objective of a software intensive organisation must be to reduce this time without sacrificing quality. By requirements non-volatility, we mean capturing good quality requirements as soon as possible in the development cycle in order to minimise changes later on. Techniques such as JAD (Joint Application Development) can help to achieve this issue and has been considered in the guide.

- Preventive actions. By preventive actions we refer to risk management and inspection issues. Inspections reduce the impact of problems. Inspections of work products allow problems to be detected as soon as possible in the development cycle. Early detection allows filtering the relevant problems, including those that impact on time, to solve them quicker, reducing the impact on other activities or development phases. Risk management is a way of anticipating possible problems and defining corrective actions in advance. This approach avoids or reduces the time spent solving a problem.
- Problem resolution management. When problems are detected, the management of its resolution, together with the user, is also a key issue to TTM. Not all the problems may need to be solved before the work product delivery. The decision on which problems to solve for the first delivery and which ones to postpone for following deliveries is critical for achieving the TTM business goal.
- Software configuration management – SCM. Requirements non-volatility and problem resolution management ask for a strong SCM. Project managers can keep the project under control by identifying configuration items to be baselined, maintaining the baseline description knowing the status of each item and managing change request.
- Project management & track (Fix TTM). Any company that needs to improve the factors that most affect TTM must, firstly, put the planning and the tracking of the projects under control. This is the reason why the other essential bases for improving TTM is PM & Track.

The interdependencies among these factors are shown in *Figure 3*.

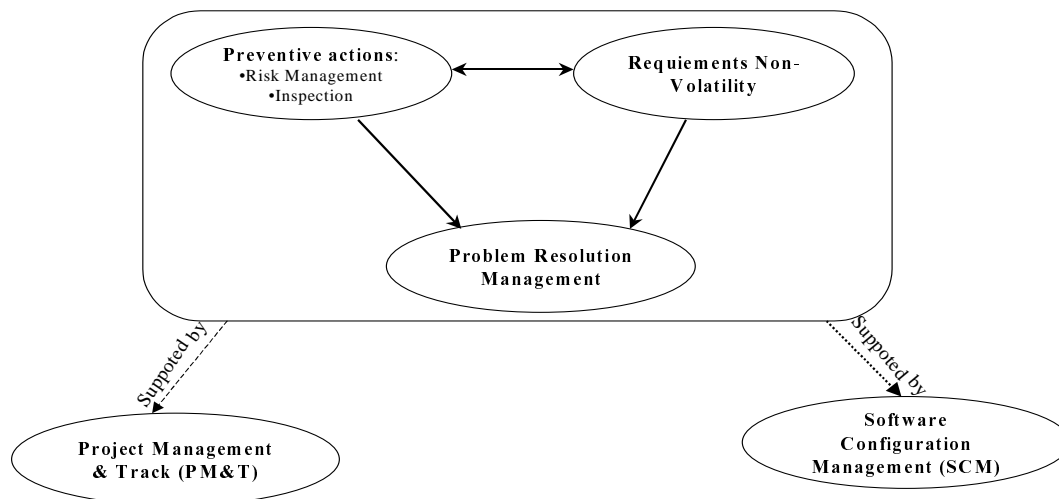


Figure 3 TTM Factors¹

¹ Dashed arrows show that SCM and PM&Track are an essential support for the other factors.

Double arrow indicates strong interaction (e.g. requirements changes can add a risk to the project, an inspection can derive new requirements needs).

Simple arrows show that problem resolution management inputs can come from preventive actions (inspections) or requirements non-volatility.

SPICE processes selected for TTM

The framework described above has to be translated into a set of concrete actions and these actions have to be prioritised to define the most appropriate improvement path. To achieve this, once the main factors affecting TTM are identified, a mapping to SPICE processes is carried out.

From the mapping, a set of processes emerge as the group that fulfils the requirements defined for optimising TTM, guaranteeing consistency of purpose and no undesirable side effects. This group comprises the basic processes assuring repetition of project results (fix TTM) and, the fundamental issues that have to be tackled in order to optimise TTM in an organisation.

Processes		Fix TTM	Preventive actions	Problem resolution Management	Require. non-volatility
MAN.2	Project management process	X			
MAN.4	Risk management process	X	X		
SUP.2	Configuration management process	X		X	X
CUS.3	Requirements elicitation process	X			X
SUP.8	Problem resolution process			X	
SUP.6	Joint review process			X	X
ENG.1.1	System requirements analysis and design process				X
ENG1.2	Software requirements analysis process				X
SUP.4	Verification process		X		
ORG.3	Human resources management process	X	X	X	X
SUP.3	Quality assurance process	X			X

Figure 4 Mapping TTM most relevant factors and SPICE

Definition of target profiles

Following the method of *figure 1*, the next step toward the improvement plan is the definition of the target profile. This profile is the graphical expression of the expected level of maturity of the processes. Given the factors relevant to TTM, it is necessary not only that the processes exist but also that they be performed with a degree of formality and efficiency. This is achieved by defining which capability SPICE level is appropriate for each of the processes. Levels are characterised by a bar diagram that shows the level that each process should achieve.

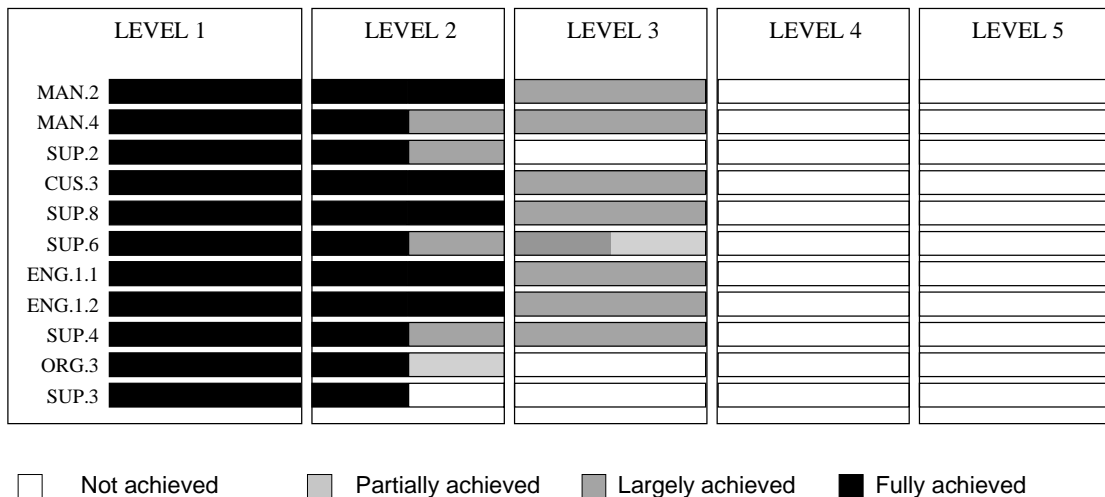


Figure 5 Target Profile

Improvement plan

The model provided by the set of chosen processes and their level of maturity forms the basis on which to build the improvement plan, as described in the guide development method of *figure 1*.

This improvement plan contains all the best practices that the SPICE model assigns to the chosen processes and management practices that are associated with the attributes of levels 2 and 3. This gives an idea of the complexity and extent of this improvement plan, outlined in *Figure 6*. The figure shows the division of the plan into three phases and 10 steps:

1. Fix TTM phase.

The main objective of this phase is to fix the delivery time of projects. This phase covers the necessary steps to assure a certain capability level of processes. The level is set such that, when applying the processes to projects similar to those developed in the past, the organisation should be able to predict time and cost with accuracy while still providing the required quality. The processes affected are: project management process, risk management process, configuration management process, human resources management process, quality assurance process and requirements elicitation process.

2. Time reducing phase.

The objective of the time reducing phase is to **optimise** the software delivery **time** while maintaining the quality of the work products. The effectiveness of the plan can be measured with a set of metrics provided by the BIG-TTM guide.

The main processes that contribute to time reduction are: verification process, joint review process, problem resolution process, customer elicitation process, risk management process, system analysis and design process and software requirements analysis process.

3. Institutionalisation phase.

The aim of the institutionalisation phase is to define processes common and standard across the organisation that can be applied and tailored to each and every software project of the organisation, if necessary. Without this phase, repetition of successful behaviours can not be guaranteed in the organisation.

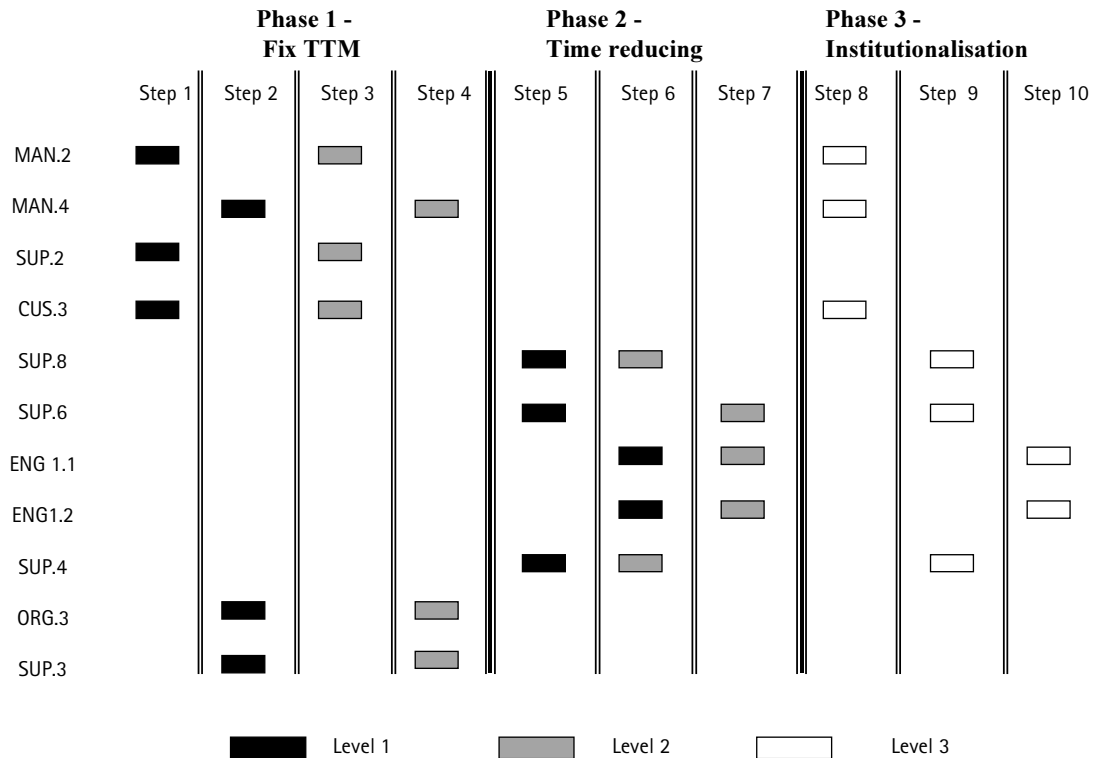


Figure 6 Improvement plan phases

Framework of the BIG-TTM guide

The final task of BIG-TTM is to provide an improvement plan which when implemented will lead organisations to achieve the time to market business goal.

For its series of BIG products, ESI works with a framework called IMPACT - Initiate Measure Plan Act Confirm Transfer -. The IMPACT improvement life cycle –see Figure 7- provides a common base for implementing an improvement plan derived from a BIG guide. The general improvement life cycle will be tailored according to the peculiarities of each guide.

There are a number of criteria that must be fulfilled before initiating an improvement life cycle:

- An organisation starting an improvement program should be aware of the extra cost and effort initially involved to implement it.
- The commitment of all parties across the organisation is not only necessary; it is vital to success. A software process improvement (SPI) plan requires an

overall change of behaviour in the organisation that is not possible without everybody's involvement.

- The importance of commitment from senior management must be emphasised. Without it no program will succeed. Companies that have been successful in creating a culture of quality have done it under the direction of senior management with knowledgeable assistance.

Some requirement before implementing the BIG-TTM plan:

- Someone with significant responsibility in the organisation should sponsor the improvement plan. It can be the manager of the software organisation unit or someone of higher authority.
- A separate improvement team should be set up. The priority for developers is to produce systems, and therefore process improvement activities might not receive as much attention as they deserve or need.

Instantiation of IMPACT improvement life cycle for BIG-TTM:

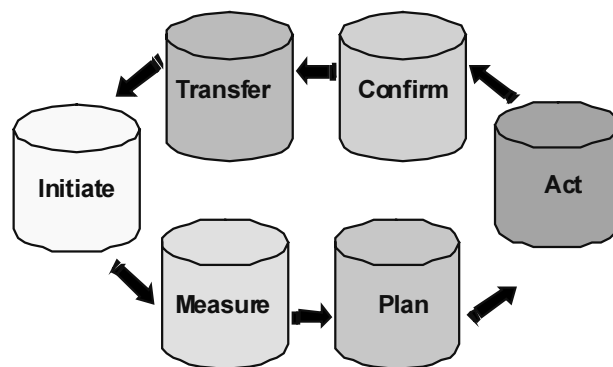


Figure 7 IMPACT Improvement life cycle

Initiate.

The initiate step of IMPACT enables organisations to identify their key requirements for process improvement and tailor the subsequent steps of IMPACT effectively.

BIG-TTM considers that the organisation has as key requirement to improve its processes in order to reduce the development life cycle of its projects. The goal of the improvement plan derived from the guide, reduces time to market, is therefore fix from the beginning.

Measure the current situation

The measure step of IMPACT enables organisations to use an approach that is right for their organisation depending on factors such as organisation size; current status of their process improvement project; and funding available.

The BIG-TTM guide measures the initial baseline of an organisation through:

- A SPICE assessment of the selected SPICE processes.
- A set of time metrics that measures progress against business goal.

Plan

The plan step of IMPACT enables organisations to select models appropriate to their needs; prepare staff for introducing and making changes; and build awareness about software process improvement.

From the baseline resulting from the measure stage, the BIG-TTM guide outlines the steps to follow to achieve a clear and well-defined improvement plan. This will cover the sequence of activities to achieve the time to market business goal and will be specifically tailored to the assessed software organisation unit. The plan also covers the organisational issues and aspects of behaviour change necessary for any organisation to undertake an improvement activity.

Act

The act step of IMPACT provides appropriate and useful information for an effective implementation of the improvement plan.

All BIG-products provide information about how to implement each of the activities of the derived plan. In the case of BIG-TTM, these guidelines have been customised with techniques and methods appropriate to the relevant TTM factors. For instance, the customer elicitation process best practices have been enriched based on the JAD method.

Confirm

The confirm step of IMPACT provides the basis for verifying and sustaining increased performance.

A common way to confirm a successful implementation of the guides derived from the BIG series is a SPICE assessment. In the BIG-TTM guide the confirmation step is reinforced with metrics focused on measuring the ability to deliver products on time.

Transfer

To complete the improvement lifecycle it is important to document the benefits obtained, the experiences gained, and the lessons for the future. Through this approach it is possible to establish and maintain the organisational knowledge, culture, and assets. These are the key elements to long term successful process improvement. The transfer step of IMPACT enables these goals to be achieved not only in an individual organisational context but also in a collaborative European setting focused on sharing experiences for the benefit of all. ESI has developed a Repository of Experience for this purpose.

Under the framework of the BIG-TTM project, ESI is executing a number of trials. The experience gained will be added to the Repository of Experience.

Summary

This work represents ESI's work contributing to SPI for SMEs through its BIG-product series and, in particular, with the BIG-TTM guide. Starting in 1997, ESI has been developing a series of BIG-products of which BIG-ISO 9001 was the first. Based on the development method deduced from that experience and the new time to market business goal, we have shown how it is possible to derive the BIG-TTM guide.

BIG-TTM is made to support software organisation to become more competitive. In order to test the validity of this approach, a number of trials will be executed. The set of metrics provided with the guide measure the time gained in developing a project similar to one developed previously in the organisation. This will allow us to establish the successful of this approach.

Finally, ESI is defining an improvement cycle framework for its BIG products that will be completed in 1999 and that has been briefly explained in the context of BIG-TTM.

Bibliography

- [1] Improvement Speed and Productivity of Software development: A Global Survey of Software Developers. Joseph D. Blackburn, Gary D. Scudder and Luk N. Van Wassenhove.
- [2] ISO9000 Certification as a business driver: the SPICE road. Ana Andrés, Pablo Ferrer, Pedro Gutiérrez and Giuseppe Satriani.
- [3] ISO15504, SPICE 98 set of documents.
- [4] Why is Software Late? An Empirical Study of Reasons For Delay in Software Development. Michiel van Genuchten.
- [5] Management Impact on Software Cost and Schedule. Dr. Randall W. Jensen.
- [6] Soft factors and their impact on time to market. Claes Wohlin and Magnus Ahlgren.
- [7] Driving Process Improvement with TTM. ESI 1998.

Appendix 1

European Software Institute (ESI) - <http://www.esi.es>

(Tel. ++34-4-4209519 Fax ++34-4-4209420 e-mail:info@esi.es)

ESI is a major industrial initiative focusing on the improvement, dissemination and usage of processes critical for software intensive systems development, procurement, quality and maintenance to make these processes predictable in terms of cost and time to market

ESI is a non profit, industry led, membership-based organisation. It has support from public institutions, independent of commercial interests. ESI co-operates with key R&D institutes in this field and relies on regional partnerships for the exploitation of results and products for all proximity services that cannot be efficiently brought to the market by electronic means.

At the same time ESI has established institutional links with applied research organisations such as the Software Engineering Institute (SEI), the Applied Software Engineering Centre (ASEC), the European Software Process Improvement Foundation (ESPI), the Fraunhofer Institute for Experimental Software Engineering (IESE).

ESI best serves the European industry by pooling its own resources with those of its members and collaborative partners. Due to its neutrality and independence as well as to its European nature, ESI is also eligible to access the results of the European Commission programs in line with confidentiality rules, in order to enhance their use and make them widely available.

Appendix 2

Elixabete Ostolaza's CV

ESI-Parque Tecnológico de Zamudio Ed. 204

E-48170 Bilbao (Spain)

Tel. +34-4-4209519

Fax +34-4-4209420

e-mail: Elixabete.Ostolaza@esi.es

Elixabete Ostolaza is Project Leader at the European Software Institute since March 1998. She is currently leader of two projects related with SPICE and EFQM models.

Elixabete Ostolaza was previously granted by the Basque Government and the European Social Fund taking part in several ESPRIT projects related with Artificial Intelligence, afterwards she was Project Manager in the Information System for Alliances and Subsidiaries Department of Telefónica, S.A. , afterwards she has been Manager of Development and Maintenance of Information Systems in Directorate General XVI of the European Commission.

Elixabete Ostolaza received her BS degree in computer science from the University of the Basque Country and her Master degree in AI from the University of Aberdeen.

Managing customer's requirements in a SME: a process improvement initiative using an IT-based methodology and tool

by Antonio Cicu (MetriQs Srl)

Fabio Valle (Visiting Fellowship, Brighton University)

Fabrizio Conicella (Consorzio per il Distretto Tecnologico del Canavese).

(Consultant team)

Domenico Tappero Merlo, Francesco Bonelli and Sandro Francesconi (OSRA SISTEMI Srl)

(Representatives of the SME)

Managing customer's requirements in a SME: a process improvement initiative using an IT-based methodology and tool - 1 di 10

Presentation at Quality Week Europe, Brussels, November 9-13, 1998

Copyright MetriQs S.r.l., OSRA SISTEMI, DTC 1998

The SME (OSRA SISTEMI Srl), and the project

- A **medium (85 people) software development company** based in Ivrea (Piemonte, Italy), belonging to a group of 170 people
- **Turnover:** 14000 US K\$, with 150 distributors/resellers
- The **product:** modular multi-user applications (named SISPAC) for yearly income tax returns, accounting and bookkeeping, salary management, budget management; operating in MS-DOS, WINDOWS, NOVELL and UNIX environment; coded in Microfocus COBOL and C.
- **Market:** individual consultants for accounting and income tax returns, medium and large business consulting offices (80.000 units, of which 8% served by OSRA)
- The **project:** the migration of SISPAC server from UNIX to Windows NT, but also including the support of new functional customer needs
- **Constraints:** 1) Fixed rigidly dates dictated yearly by fiscal schedules; 2) strict dependency from outside organizations like Finance, Industry, and Work National Govern Departments

The set of customer management processes (from SPICE Customer-Supplier process category)

- acquire software
- manage customer's needs
- supply software
- operate software
- provide customer service

the addressed
process

the base
practices

- obtain customer requirements and requests
- agree on requirements
- establish requirements baseline
- manage requirements changes
- understand customer expectations
- keep customer informed

*This schema corresponds also
basically to the
Requirement Management KPA
of CMM*

(See also paper, Fig. 1)

Requirement Management innovation (1)
Four elements addressed:
People, Organization, Methodology and Technology.

People

Training

- process
- requirement analysis methodology;
product/process measures
- technology

Management commitment

The two above factors provided:

- cultural awareness
- people commitment
- responsibilities clarification

Organization

Definition of the process:

- based on ISO/IEC 12207, CMM and SPICE
(*see paper, Figure 2*)
- in the framework of a ISO/IEC 12207 and
IEEE compliant project life cycle (*Fig.3*)

Definition of User Requirements
based on the analysis of clients'
business process (*See paper, Fig. 4*)

Participation of all the three
interested departments of OSRA
(Product Development, Product Planning,
Customer Support)

Requirement Management innovation (2)

Methodology

Guidance for the User Requirements document preparation:

- the IEEE Guide for “Concept of Operations Document”

The approach sustained by the mentioned Guide stimulates the production of a process-based view of the user needs (*See paper, Fig. 5*)

The definition of measurements of process effectiveness and efficiency was based on:

- **ami** (*See paper, Fig. 6*)
- GQM (*See paper, Tables 1 and 2*)

Requirement Management innovation (3)

Technology

A specialized and user friendly tool was adopted to facilitate the writing of documents. The tool (ProDoc, by SPC, Vancouver; Canada) is usable with the most diffused word-processors, and provides:

- a set of templates
- an on-line guide

Successful example of adoption of a cheap and relatively low-tech tool: low-tech tools can be an abundant source for organizational and technical improvements in the software industry

Results (1)

Costs

- 1) The training: cost required anyway within the ongoing Quality System effort
- 2) No specific additional costs, due to the innovation (except the acquisition of the tool), were sustained in the writing of the documents.
- 3) Costs of measurements. Less than 0,2 % of additional cost, for counting the requirements; time/effort and defect data collection was already an established practice

Results (2)

Benefits

- 1) increased involvement and cooperation of external and internal customers
- 2) clearer understanding of the current customer's process and of their future process, and correspondingly of the current and of the new SISPAC systems;
- 3) the user process is now a reference used for evaluating the completeness, consistency and adequacy of the requirements of the new system;
- 4) a much more professional image toward the external customers;
- 5) greater opportunity of handling conflicts of requirements, because customer's needs are clearer;
- 6) A good premise for reducing future post-sale problems.

Lessons learned

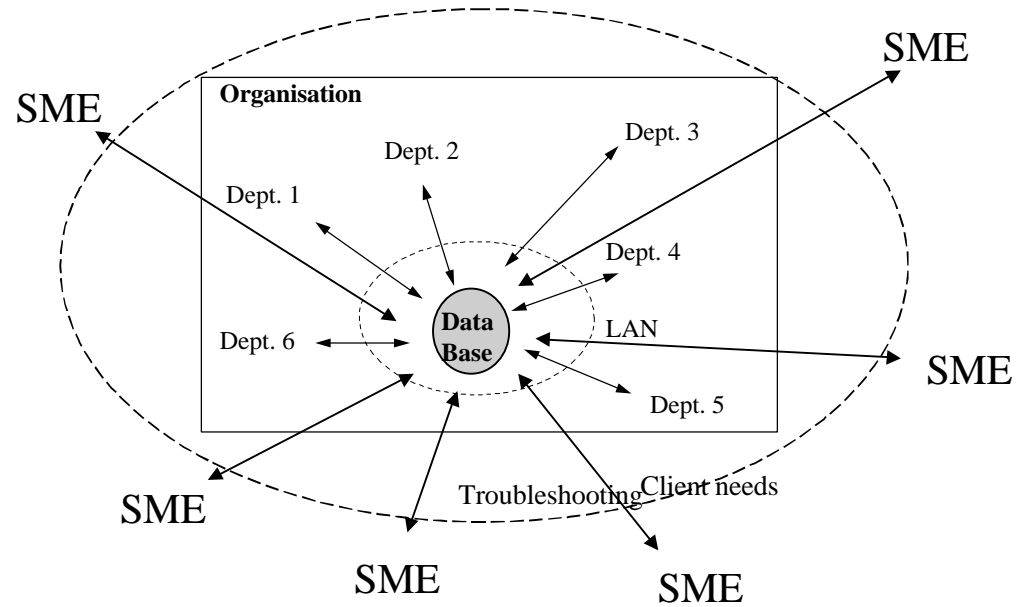
The main lesson learned regards the better control over product and process quality that can be obtained through three main means:

- a) the use of process model and related document template for getting guidance in the work
- b) the use of a shared and controlled database as repository where to keep the baseline versions of the results, which makes easier to the participants the access to the updated information, with the guarantee of working on the correct and controlled version.
- c) the collection of requirements, defect and time/cost data for producing effectiveness and efficiency indicators.

Future developments

OSRA CASE: INTERNET/INTRANET

for improving communication between the actors



Managing customer's requirements in a SME: a process improvement initiative using an IT-based methodology and tool.

by Antonio Cicu (MetriQs Srl)¹, Fabio Valle (Visiting Fellowship, Brighton University)²,
Domenico Tappero Merlo, Francesco Bonelli and Sandro Francesconi (OSRA SISTEMI Srl)³,
Fabrizio Conicella (Consorzio per il Distretto Tecnologico del Canavese)⁴

Abstract

The focus of this paper is the application of IT solutions together with methods of process improvements to manage the customer's requirements. According to the terminology of the ISO 9001 and ISO/IEC 12207 Software process standard, the area covered by the initiative of process improvement is the User Requirements Definition.

The paper is based on a real process improvement initiative conducted by MetriQs for six months in a small Italian software company producing accounting and income tax returns packages.

The results were presented also as a case study to the European ESPRIT Project 21461 TBPTIME, having as objective to collect and diffuse for training purposes a set of significant case studies of process improvements for SMEs in different process areas.

During this improvement initiative the following methodologies and standards have been used: ISO/IEC 12207 [1], ISO/IEC PDTR 15504-1/9[2], J-STD-016-1995 EIA/IEEE [3], ami [4], GQM [5]. The application of the mentioned guidelines and standards was facilitated embedding the guiding tips in a template (for MS WORD), produced with the ProDoc [10] tool.

1. General presentation of the company and of the sector

1.1 Business Nature.

The company. The described case deals with OSRA SISTEMI (located in Ivrea, Torino, Italy), a company that operates in the market of solutions for accounting, tax, salary and budget management for consultants, business consulting offices, companies, business category and professional associations.

OSRA SISTEMI is part of OSRA Group, founded in the middle 70's and composed of 4 companies: OSRA SISTEMI, OSRA PAGHE, OSRA TELECOM, SISPA. The OSRA Group employs 170 people (85 of which in OSRA SISTEMI) with a global turnover of 25 Billions Italian Lire, and 150 distributors/resellers throughout Italy.

Environment: market, competitors, customers, technologies, constraints, regulations.

Market and customers: the Italian vertical market of the company is represented by individual consultants for accounting and income tax returns, medium and large business consulting offices, business category and professional associations which support their associates in their accounting and fiscal needs.

Market dimension: about 80000 units [from small (individual consultants) to large multi-user business offices].

¹ MetriQs Srl, Via don Gnocchi, 33 -20148 Milano (Italy) -Tel/Fax (+39) - 2- 4870 8691; e-mail: acicu@metriqs.com

² University of Brighton, Village Way Falmer - Brighton BN1 9PH - UK - e-mail: Fabio_Valle@compuserve.com.

³ OSRA SISTEMI Srl- Via Ribes, 5 - 10010 Colletterto Giacosa (Torino) -Italy- Tel. (+39) - 125 - 561511- Fax (+39) - 125 - 561510; e-mail: tapdom@osra.it

⁴ Consorzio per il Distretto Tecnologico del Canavese - P.O.Box 192 - 10015 Ivrea (TO) - Italy - Ph. (+39)-125 - 2331201 - Fax (+39) - 125 - 2331224 - email: f.conicella@eponet.it

Competitors: there are about 300 from small to medium software houses operating with specialized products in this sector in Italy. The majority of them serves about 60% of the market with small, specialized single user solutions.

The 4-5 leaders (among which OSRA SISTEMI) serve about 40% of the market. Each of them has a share from 8% to 10% of the market, including the totality of large business consulting offices, and provides them complete Information Systems which modularly integrate different applications needed by large users.

Used **technology** : OSRA uses Microfocus COBOL and C languages as development languages, on Index-sequential Databases. In this way they avoid the use of relational DB that is not affordable for users. Up to now OSRA has developed and delivered on UNIX environment.

Constraints: 1) Fixed rigidly dates dictated yearly by fiscal schedules; 2) strict dependency from outside organizations like Finance, Industry, and Work National Govern Departments; 3) Current market not open to invest in new technologies: users want good functionalities at a low price, otherwise they provide their services manually.

Regulations: yearly evolving rules dictated by the above mentioned Finance, Industry, and Work National Government Departments.

Business history and future developments

Middle '70s: strong cooperation with Olivetti on proprietary Hardware and Software systems

Middle '80s: migration to UNIX and MS-DOS, always on Olivetti hardware

Late '80s: migrate on other Hardware vendors, continuing to use UNIX, DOS, and extending the platform to Network products, making the developed products more and more independent from the platforms

Current development: Client server architecture, where the servers are, in addition to UNIX, first Windows NT, and then AS/400. Client Windows 3.1 and Windows 95, and next move also toward Internet and Intranet with a client named OSRANET.

1.2 Product/market portfolio

The product of the company is a modular information system (named SISPAC) operating in MS-DOS, WINDOWS, NOVELL and UNIX environment. The main modules of the product are represented by applications supporting the yearly income tax returns 730, 740, 750, 760, 770, accounting and bookkeeping, salary management, budget management.

On the basis of a common architecture the company has created a family of products that support single applications among the above mentioned ones, or full integrated Information Systems that integrate all the applications or part of them according to custom needs.

1.3 Product/market selection

The experimentation of the process innovation has been done on the SISPAC product, and specifically on the project which will implement the migration of SISPAC server from UNIX to Windows NT, but also including the incorporation of new functional customer needs.

2. Organizational and technological improvement initiatives

The product has an intense evolution life during the first six months of every year, its updates have to be delivered yearly and quickly to all the users (many thousands). First of all it must support the evolving work organization of the user. Moreover the product must be constantly and fully aligned to the evolving legislation and administration rules and must have a very high level of functionality and reliability. Within such a scenario, with the objective of controlling better the schedules and quality, two years ago OSRA SISTEMI has begun a series of changes defining more clearly the responsibilities and roles in the software development process for all the people, and creating and tuning an efficient HelpDesk and Maintenance service.

The current improvement step is based on two lines of improvement:

- 1) consolidation, with maximum priority, of the Requirement Management process, as the main foundation on which to base an accurate and efficient management of customers and of their needs.
- 2) preparation and application of Quality System and the related Quality Manual, using the ISO^{plus} product [12].

The first initiative is described in this paper.

Requirement Management has been innovated considering the four elements: People, Organization, Methodology and Technology. This global approach to the process improvement initiative, combined with the corporate management commitment (necessary for every organizational change [6]), has prevented from cultural, organizational and technical difficulties. The details of the approach are provided in this paper in section 3 (People), section 4 (Organization), section 5 (Methodology), section 6 (Technology). The methods and tools were identified and evaluated with the consulting help of the same specialized company which provided the training.

3. People

On the People side an external consultant provided detailed and practical training of all people involved in the Requirements Management. The training gave visibility of the methodology described later (Section 5) in the paper, made people more aware of the needs of the internal and external customers, bought commitment and avoided cultural obstacles to the initiative.

4. Organization

On the organizational side the new customer management global process was represented (in conformity to international standards [1], [2]) as a set of processes and practices that involve the supplier in its support to the customer across the life cycle. Particular attention, among the mentioned customer management processes and practices, was posed in the management of customer's needs because this process produces a document, the "User Requirements Specifications", that becomes the reference for any other project activity and that affects the best mapping of Software Requirements to the business needs.

4.1 View of the customer management global process

The **customer management global process** is constituted by a set of processes and practices that involve the supplier in its support to the customer across the life cycle. These processes and practices are performed in order to solicitate the active participation of the customer from the beginning of the project and to maintain the best level of customer satisfaction.

The **set of customer management processes (PRO) and practices (PRA)** is the following:

- manage customer's needs (PRO)
- contract preparation, agreement and monitoring (PRA)
- joint reviews (PRO)
- product delivery, installation and acceptance(PRA)
- operate software (PRO)
- provide customer service (PRO).

In document [1] (see in [1] the description of the acquisition process), and in document [2] (see document [2] in Part 2 for the Customer-Supplier process category, and in Part 5 for the details of Base Practices for the Customer-Supplier processes) a detailed description is given of a standard model of processes and practices where supplier and software buyer interact with each other and cooperate toward project goals.

Figure 1 provides a graphical representation of processes and base practices. According to the model of document [2], these processes and base practices should be performed by the supplier to manage the customer issues, and to cooperate with and support the customer in such a way that the customer can best perform tasks that typically go under its responsibility.

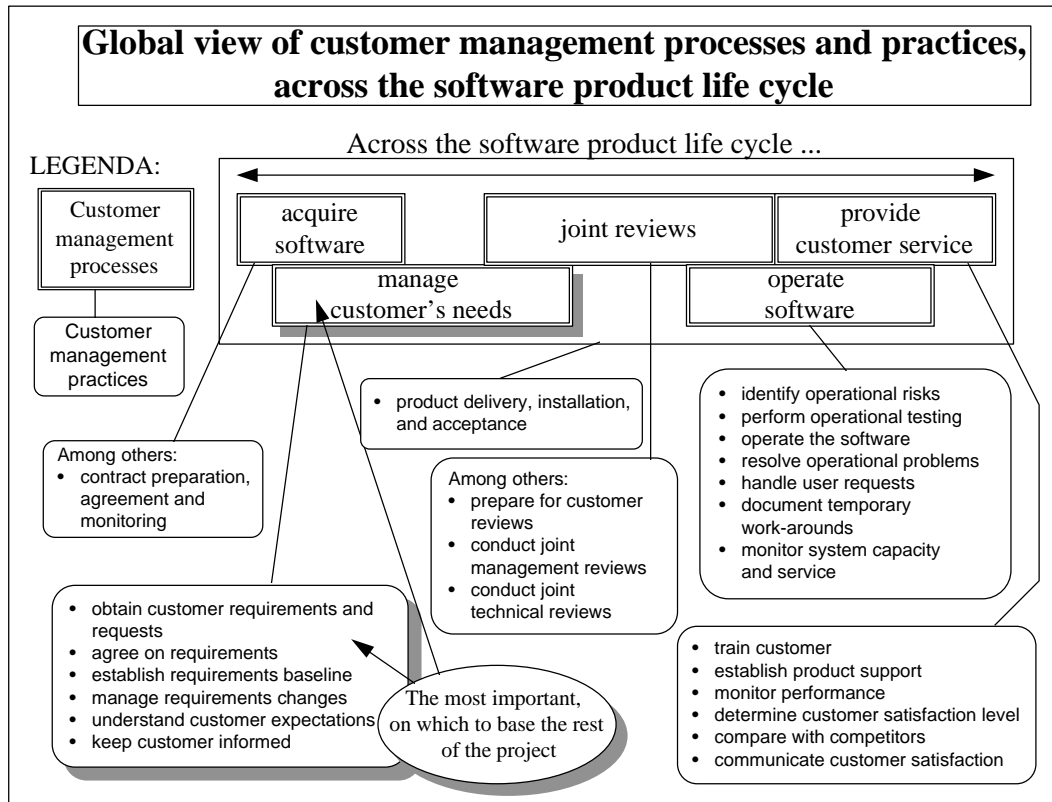


Fig.1 Global view of customer management

Among the mentioned customer management processes and practices, the management of customer's needs is by far the most important, because this process produces a document (the "User Requirements Specifications" or "Application Requirements Specifications") that, provided that it contains a complete definition of consistent, adequate and stable user requirements, becomes the reference for any other project activity (software requirements definition, development and quality plans, testing, product acceptance, and an important part of customer training).

4.2 The Customer's Needs Management process

Manage customer needs process is to manage the gathering, processing, and tracking of ongoing customer needs and requirements throughout the operational life of the software; to establish a software requirements baseline which serves as the basis for the project's software work products, and activities; and to manage changes to this baseline." (from document [2] part 5).

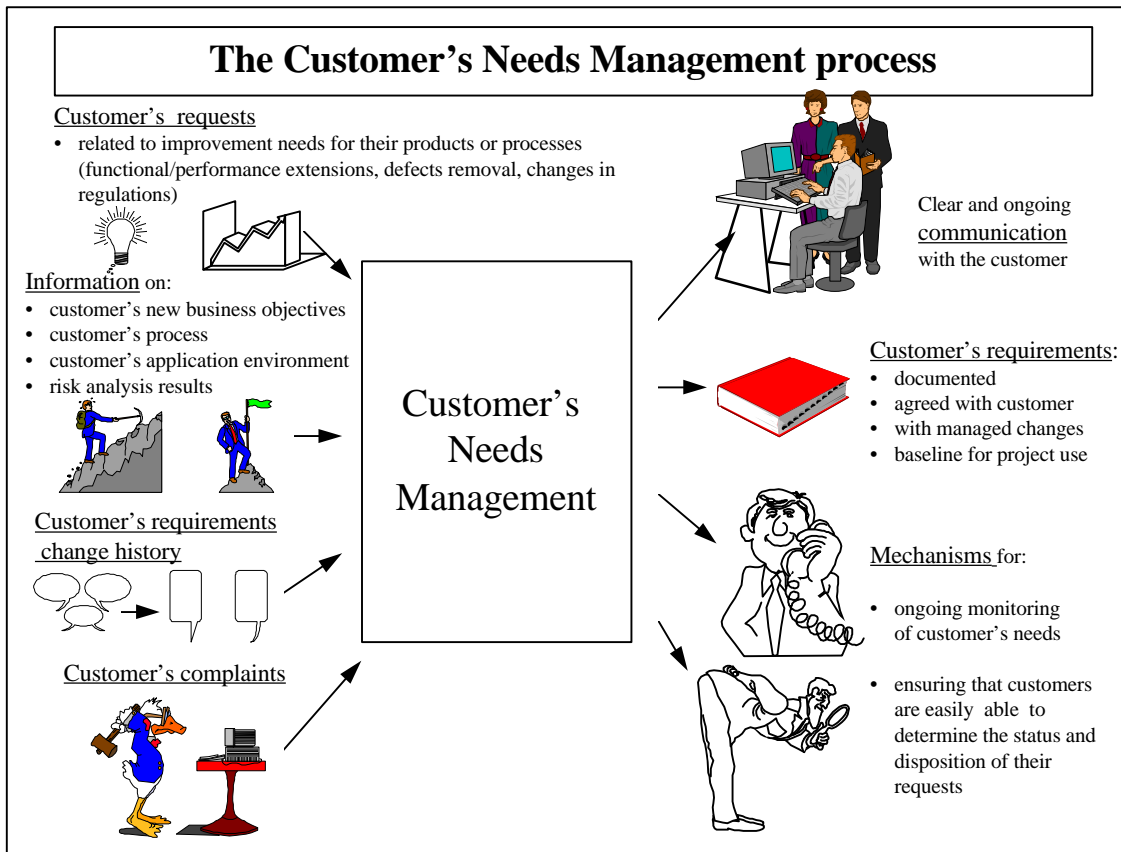


Fig. 2 Customer's Needs Management

Inputs and outputs of the above process are described in Figure 2.

4.3 Description of the innovation experimented for improving the Customer's Needs Management process

4.3.1 The adopted software life cycle. Which part of the life cycle is addressed by the improvement.

Before describing the specific process addressed by the experiment, it is necessary to describe the adopted life cycle framework, represented in Figure 3.

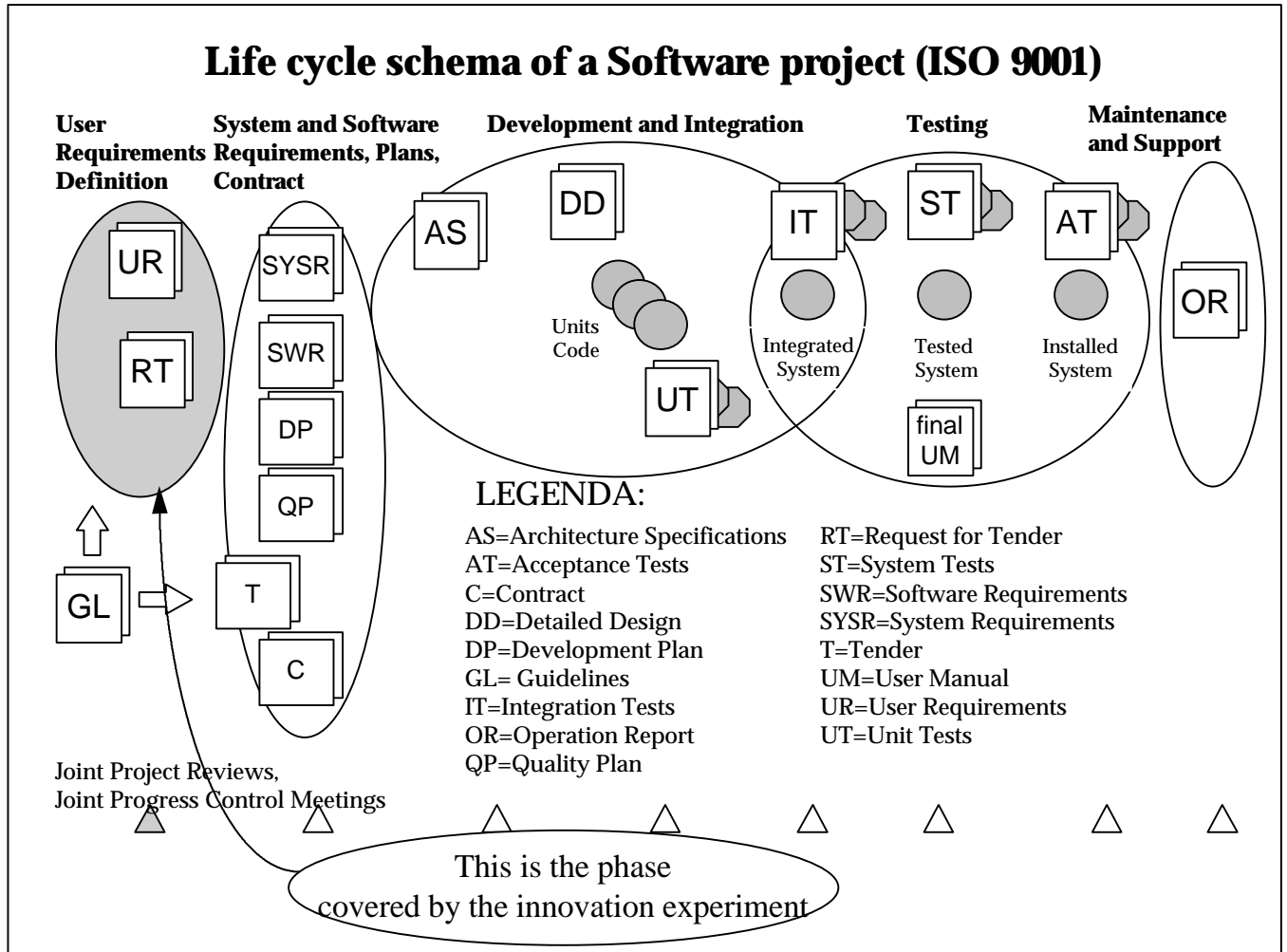


Figure 3

Figure 3 provides a software life cycle schema compliant to ISO 9001 standard, showing the various software work products produced in the different phases, where squares are documents, circles are executable code, and octagons are test cases.

The legend in the Figure 3 clarifies the nature of each specific software work product.

The experiment has covered the first phase of the figure, where the UR (User Requirements) document is the result of the Customer's Needs Management process, illustrated in section 4.2.

4.3.2 Process view of the supplier-customer cooperation to produce a good quality “User Requirements” document. Actors involved.

The scenario of Figure 4 shows the Customer’s Business process (let’s call it the CB process) of the direct Customers of OSRA SISTEMI which, making the best use of the SISPAC product, provide a specialized service to their customers (the end-users of the global process). This CB business process is affected (and forced to evolve) every year by two categories of factors: some external ones such as the periodic changes in accounting and income tax returns laws and regulations, and some internal ones like the interest of the business consulting offices to perform their services with higher levels of integration, effectiveness and efficiency.

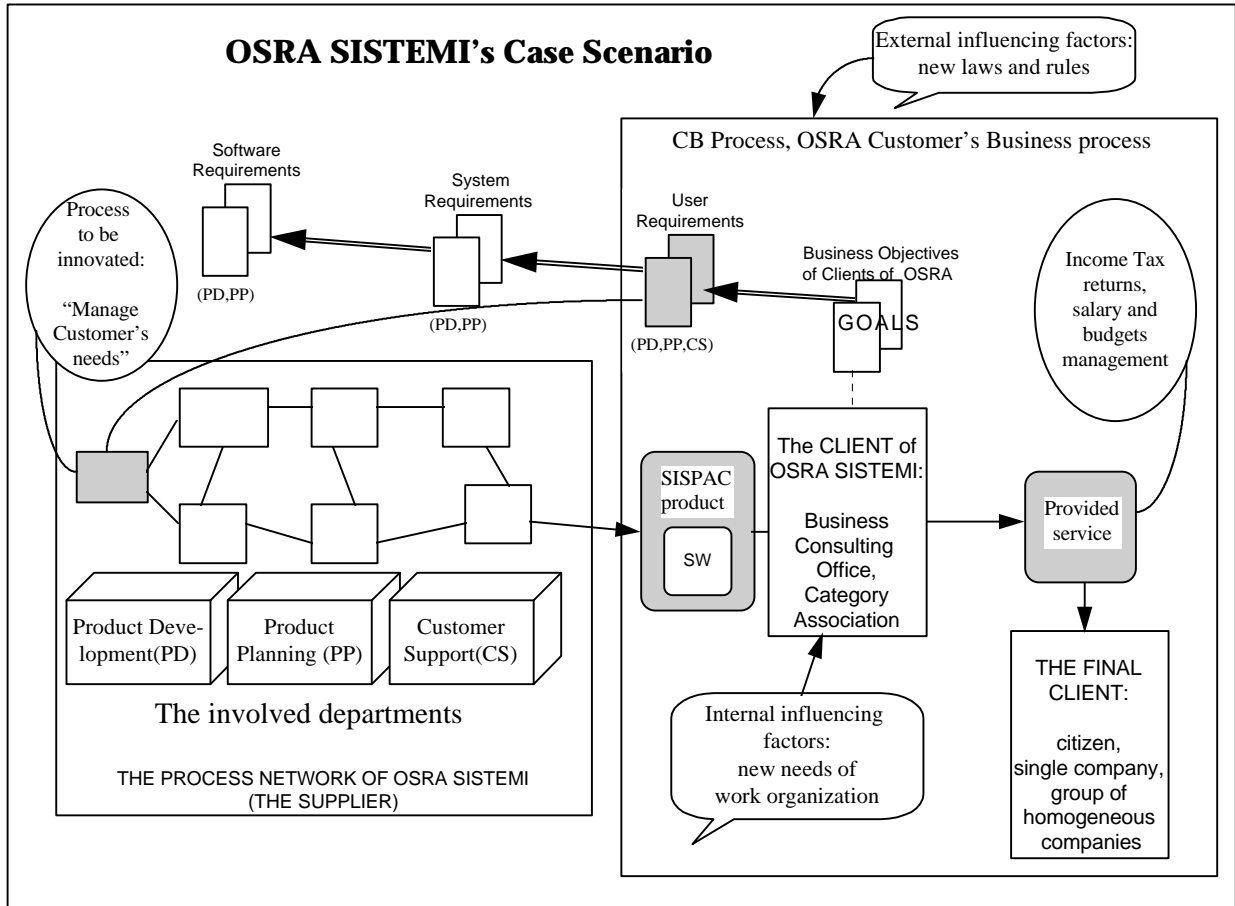


Figure 4

The better is the support provided by the new versions of the SISPAC software to the evolving business needs, the higher will be the success of OSRA Customers in their business, and the higher will be their satisfaction.

So the problem is to guarantee the best mapping of SISPAC Software Requirements to the business needs. In order to achieve this result, OSRA management has decided to take great care of the production of the initial document of the life cycle of each new SISPAC version, the “User Requirements” document. OSRA has kept innovating the approach and the content of the document, in such a way to mirror the evolving CB process and stimulate the best understanding of the customer’s process by the interested OSRA people.

All the three interested departments of OSRA (Product Development, Product Planning, Customer Support) have participated actively to the preparation of the document, each with its specific competence and responsibility. In particular, Product Planning plays also the role of representative and sponsor of all the SISPAC customers (both current and future), and carries their voice into the specification process.

In order to provide the best guide to the preparation of the document, a specialized guideline has been

adopted, which is described in the next subsection.

The application of this new guideline together with a outstanding cooperation of the mentioned three OSRA departments, properly trained, constitutes the core of the innovative experiment, that has allowed to perform in a new way the “Customer’s Needs Management” process (Fig. 4).

5. Methodology

5.1 *The specialized guideline*

On the methodological side the guidance for the preparation of the User Requirements document was provided by the IEEE “Guide for Concept of Operations Document”, which is the initial component of the guidelines belonging to the body of guides available in document [3].

Figure 5 provides the first level Outline of the guiding Table of Contents adopted for the documents which specify the user needs of the next version of SISPAAC software, consisting of:

- migration of SISPAAC software to Windows/NT
- functional extensions, and new printers handling
- new user interface, including a desktop integrating all SISPAAC applications

The approach sustained by the mentioned Guide stimulates a process-based view of the user needs (consistent with the improvements on the organizational side). The main contents of User Requirements document are described in Fig. 5 and in the explanatory text provided in this subsection.

The guideline supports both a first case where user needs refer to a scenario in which the software is required to be embedded into a product in order to improve the product, and a second case where the software is required to automate a part of a process in order to improve the process. In both cases the authors are required to describe needs using a language and a terminology specific to the applicational problems and context of the user, avoiding as much as possible the use of IT jargon and avoiding to anticipate at this time the IT based solution to the needs.

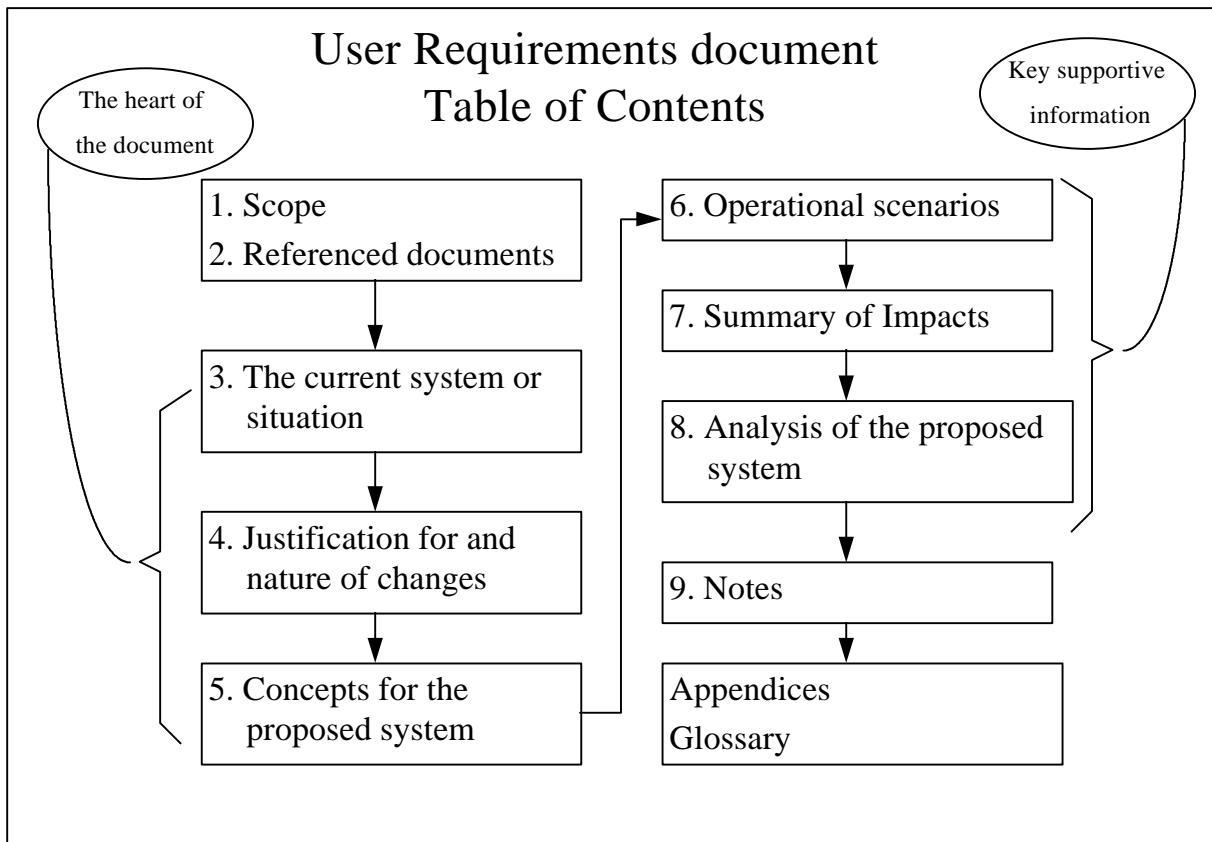


Figure 5

With reference to Figure 5, section 1 and 2 are introductory. Sections 3, 4, 5 are the heart of the document. Section 3 (The current system or situation) describes the current product or process with their static and dynamic characteristics, including also the following information:

- background and objectives, in relation with specified business objectives of the user
- operational policies and constraints
- modes of operation
- involved user classes
- support policy and environment.

Section 4 (Justification for and nature of changes) first specifies which are the new business objectives (or the old ones, if any, which are not well supported by the current product or process). In front of the new objectives this section specifies which are the limitations (still present defects, weak points or missing capabilities of the current system) being an obstacle toward the achievement of the new objectives. This list of limitations is the reference for justifying the changes (and related priorities) to be made to the current system (product or process) to make it able to support the achievement, by the user, of the new business goals. The guideline suggests also to specify, if any, the changes considered but not included (and the reason for the “don’t include” decision) in support to consideration for future extensions, in order to capitalize also the decisions taken.

Section 5 (Concepts for the proposed system) describes the new system, conceived as answer to the limitations described in Section 4. The template for this description is the same used for Section 3 for the current system. At this point it is possible to understand that the description of the new system can be easily derived from the one provided in Section 3, specifying easily the qualifying distinguishing requirements of the new system that are the answer to the needs specified in section 4.

Assuming that the language and terminology used in Sections 3, 4, 5 is the application oriented one normally used by the customer when dealing with its problems, and that the systems (current and new) are described in terms of flow of real work products between processes (with related entry/exit criteria and rules), we can understand that such document becomes really the reference, understood and easily agreeable by the customer, on which the parties can compare every technical progress and achievement.

Section 6 has the purpose of describing selected operational scenarios. A scenario is a step-wise description of how a system should operate, under selected representative circumstances, and communicate with other external systems, and the end-users. The scenarios should allow the readers to walk through the new system and understand how the various parts work together, to provide operational capabilities, and also to understand better user's roles, and how the features are provided to them. A scenario provides a key help for evaluating the completeness of the requirements of the new system versus the needs.

Section 7 has the purpose of supporting analysis and documentation of impacts of the new system from the operational and organizational point of view, as well as the impacts foreseen during development.

Section 8 has the purpose of documenting in organized way the balance of known costs and benefits associated to the new system. Such analysis again stimulates the analysts to go in depth in understanding pros and cons associated to the new characteristics, including the advantages that the new system can produce on the costs of process workflow.

5.2 Measurements of the process and of the results of the first experience

The adopted methodological approach addresses also the measurements of effectiveness and efficiency of the process to be improved (process effectiveness is defined as the capability of the process to produce, in schedule and within budget, results of good quality; process efficiency is defined as the capability of the process to produce the results in the fastest and cheapest possible ways). On this side **ami** [4] and GQM (Goal, Question, Metric) [5] were the adopted methods: **ami** helped in defining the goals/subgoals tree (Fig. 6), and GQM helped in defining the proper questions and metrics to be used for quantitative evaluation of the achievements (Tables 1 and 2).

The process model used to assess the current process before introducing the innovation was constituted by the mentioned guidelines of the IEEE standard for User Requirements [3], enriched by the model elements given by the ISO Software Process Assessment Standard available from the SPICE project [2]. Considering the main business goals (improve quality of the product, reduce time to market, reduce costs, increase productivity, see Figure 6) there was not a single business goal selected, with priority on the others. The rationale behind this consideration is that an improvement in Customer's Needs Management can generate balanced benefits on all the above business goals, assuming that the process improvement would consist in improvements both of process effectiveness and process efficiency.

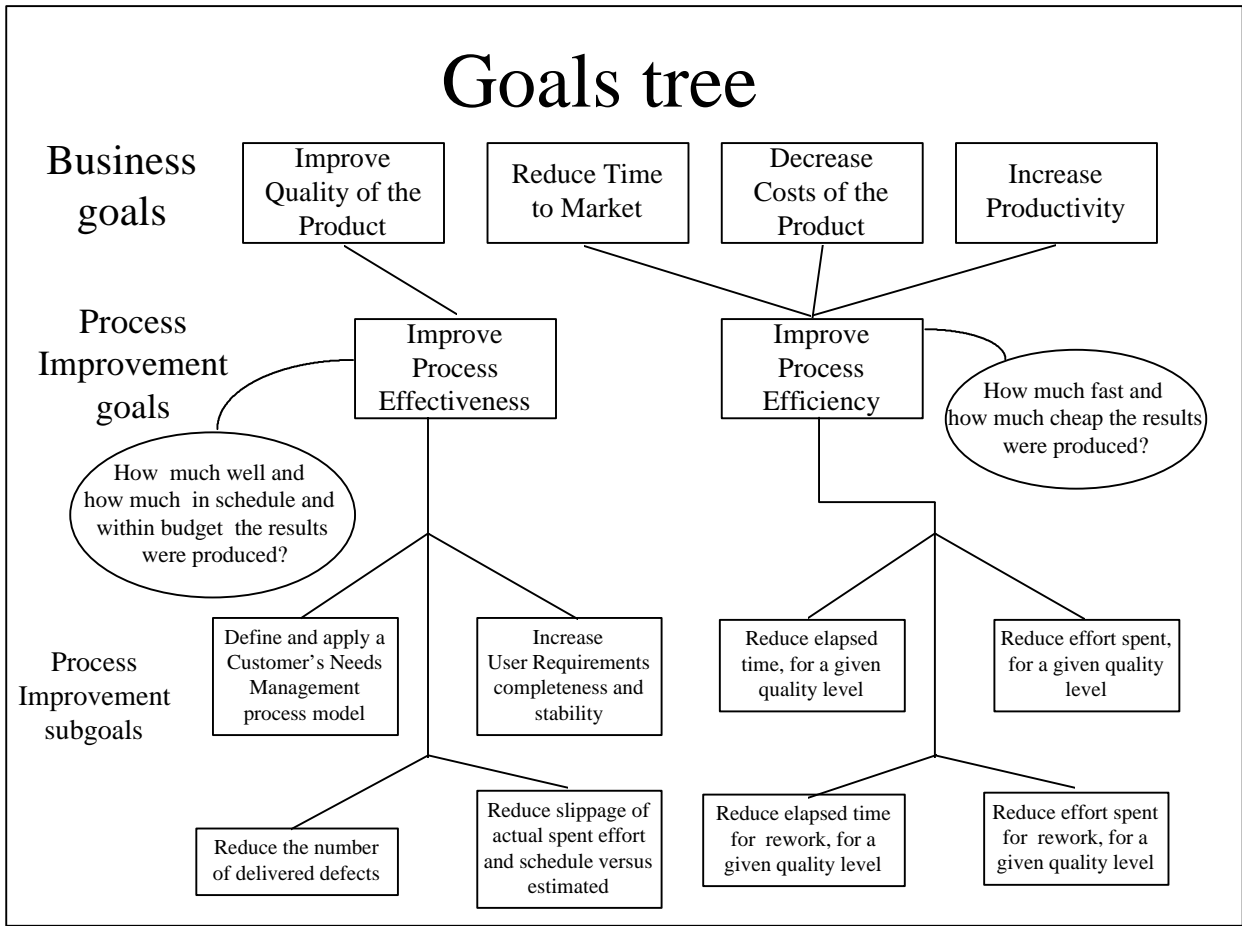


Figure 6

An existing Customer's Needs Management process model defined for the organization, and a previous schema of related process measurements being both not available, we selected as process improvement objectives both the process effectiveness and efficiency. In this way we created a basis of data from this first innovation experience to be used as reference for the comparative evaluation of future further improvement steps. Table 1 provides the GQM schema of subgoals selected in support of improvement of process effectiveness: the subgoals, the related questions, the metrics with definitions, and the results. (Remark: in Table 1, at Subgoal 1.3, customers were represented by Product Planning department of OSRA SISTEMI)

Table 1. Goal 1: Improve Customer's Needs Management process effectiveness

Question: How much well and how much in schedule and within budget were the results produced?
 (LEGENDA: Q=Quality; NQ=Non-Quality)

Subgoal	Question	Metric	Measured value
1.1 Define and apply a Customer's Needs Management process model	How much is the adopted OSRA standard model compliant to the international standard model of the process?	Completeness of the standard OSRA process model, versus the international standard model = = Percentage of international std template items which were adopted in OSRA std defined model , over the total of international std template items <i>[Q metric, the higher the better]</i>	100%
	How faithfully was the adopted OSRA defined process standard applied in this experience?	Completeness of the model used in this experience, versus the OSRA standard model = = Percentage of OSRA adopted template items really applied in this project, over the total of adopted OSRA template items. <i>(This metric is calculated at the end of a phase when the process faults have been corrected)</i> <i>[Q metric, the higher the better]</i>	90%
	How much the process and document model was violated, before consolidating the results?	Percentage of detected process faults, over the total of process template application instances. <i>[NQ metric, the lower the better]</i>	10%
1.2 Increase User Requirements completeness and stability	How completely the Input Information was taken into account to specify User Requirements?	Customer's Needs Management Coverage (%) = =Percentage of Input Information Items taken into account in the actual specifications of User Requirements, versus total number of Input Information Items <i>[Q metric, the higher the better]</i>	100%
	How much complete and stable were the requirements, through the successive Requirements Baselines?	Averages, over all available baseline versions, of percentages of (versus the total of requirements of previous baseline version): - requirements changed - requirements added - requirements deleted <i>[NQ metrics: for each, the lower the better]</i>	2,8% 0,9% 0,5%
1.3 Reduce the number of delivered defects	How many remained defects were detected by customers, per 100 Requirements?	Detected (by customers) Remained Defects Density (%) = = (Total No. of detected (by customers) remained defects/Tot.No. of Reqs)*100 <i>[NQ metric, the lower the better]</i>	4,2%
	How many defects were removed versus the total detected?	Percentage of removed over total detected defects <i>[Q metric, the higher the better]</i>	100%
	How many defects remained undetected or unremoved going into the next phase, per 100 Requirements?	Remained Defects Density (%) = (Total No. of defects remained after this phase/Tot.No. of Reqs)*100 <i>[NQ metric, the lower the better]</i>	0%
	How much time a defect remains into the product before being detected and removed?	Defect Average Life (DAL)= =Σ _{each defect} (Ordinal number of Detection phase - Ordinal number of Injection phase)/(To.no. of detected defects) <i>(Reference is made to phases of the adopted life cycle)</i> <i>[NQ metric, the lower the better]</i>	0
1.4 Reduce slippage of actual spent effort and schedule versus estimated	How much the actual spent effort did match the estimated one?	Spent_effort_matching (in %)= =[1-abs_value_of(actual_eff - est_eff)/est_eff]*100(%) <i>[Q metric, the higher the better]</i>	89%
	How much the actual schedule did match the estimated one?	Schedule_matching= =[1-abs_value_of(actual_sched - est_sched)/est_sched]*100(%) <i>[Q metric, the higher the better]</i>	92%

Table 2 provides the GQM schema of subgoals selected in support of improvement of process efficiency: the subgoals, the related questions, the metrics with definitions, and the results.

Table 2. Goal 2: Improve Customer's Needs Management process efficiency
Question: How much fast and how much cheap were the results produced?

Subgoal	Question	Metric	Measured value
2.1 Decrease elapsed time from process start to process end, for given product quality levels and given effort spent levels	How much fast is the User Requirement Specification process?	Process quickness = Customer's Needs Management Process Elapsed Time efficiency= =Amount of User Requirements specified per elapsed time unit= =(Total number of Requirements specified in the last baseline)/(total elapsed time) <i>[Time efficiency metric, the higher the better]</i>	29 Requirements specified per elapsed calendar day
2.2 Decrease effort spent from process start to process end, for given product quality levels and given elapsed time levels	How much cheap is the User Requirement Specification process?	Process cheapness = Customer's Needs Management Process Effort Spent efficiency= =Amount of User Requirements specified per effort spent unit= =(Total number of Requirements specified in the last baseline)/(total effort spent) <i>[Effort efficiency metric, the higher the better]</i>	18 Requirements specified per man-day
2.3 Decrease elapsed time, from process start to process end, for Reworking on defective requirements, for given product quality levels and given effort spent levels	How much fast is the User Requirement Specification Rework process?	Process quickness for rework = Customer's Needs Management Process Elapsed Time efficiency, for Rework= =Amount of User Requirements reworked per elapsed (in rework) time unit= =(Total number of Requirements reworked in the last baseline)/(total elapsed time for rework) <i>[Time efficiency metric, the higher the better]</i>	10 Requirements reworked per elapsed calendar day
2.4 Decrease effort spent, from process start to process end, for Reworking on defective requirements, for given product quality levels and given elapsed time levels	How much cheap is the User Requirement Specification Rework process?	Process cheapness for rework = Customer's Needs Management Process Effort Spent efficiency, for Rework= =Amount of User Requirements reworked per effort spent (in rework) unit= =(Total number of Requirements reworked in the last baseline)/(total effort spent in rework) <i>[Effort efficiency metric, the higher the better]</i>	7,5 Requirements reworked per man-day

The experience was important as first step to tune the OSRA SISTEMI Requirements Management methodology, and to establish a measurement schema and approach to be applied also to the next Requirement Specification steps, which are foreseen in the other life cycle phases described in Figure 3.

6. Technology

A specialized and user friendly tool was adopted to facilitate the writing of documents. The tool ([11], see Appendix A) is constituted by a set of templates, usable with the most diffused word-processors, which has two main characteristics:

- 1) It provides the users a set of templates for the most important documents of a software project. The templates guide the authors to define the characteristics of a software product according to a content compatible with IEEE standards for software documentation and ISO 9001 compliant. The templates are easily adaptable to the specific needs of a given project. In the case of OSRA SISTEMI the templates were enriched with the specific template described in section 5.1.
- 2) The tool provides an on-line guide to explain what type of information is required for each section: the on-line guide is designed to be used "as is" by software engineers when writing technical documents. An ease-of-use aspect of the on-line guide is also that for each type of information required the author (depending on her/his skill level) can consult additional instructions that give additional explanations to understand topics in a given section. To help clarify the guidelines, examples are provided to indicate the specific types of information required. Such additional explanations and examples are located in boxes which can be activated or deactivated using the "See Instructions" or "Hide Instructions" commands of the "Visualize" pull-down Menu of the used word-processor. After completion of the document, the user with a simple mouse click can hide both the guidelines and examples to create the final document. Examples of how the guidelines are presented to the authors are given in Appendix A.
From the formal point of view, the sections for each document are already formatted using the standard fonts and features of your documentation, that were chosen as standard for the whole organization. The user can simply enter the appropriate information required for each section without worrying about fonts, formatting etc.. This guarantees a uniform documentation style throughout the organization.

The tool is relatively cheap and the return of the investment is very high because it can largely improve the efficiency and effectiveness of the process. The tool can be considered relatively low-tech (requiring for this reason a very limited amount of training) but provides a great support to the process of definition of customer's requirements. Once again low-tech tools can be an abundant source for organizational and technical improvements in the software industry [8,10,12,13].

7. Results of the improvement initiative

7.1 The production of specifications of the new product

The focus of the initiative has been the process through which the direct customers of SISPAC provide their services to the end-users. This initiative gave evidence that the main needs were on one side to use the SISPAC applications in the emerging Windows/NT operating environment, on the other side to exploit of Graphic User Interface facilities enhancing the ease of use of the single application, and to provide easy access to each SISPAC application from one unified desktop.

The adopted structure of User Requirements document has allowed to produce effective descriptions of the current and of the new wanted solutions, and to understand easily differences, advantages and impacts of the envisaged solutions.

The following OSRA Departments have participated actively to the writing and review of the documents:

- Software Development Department
- Product Planning Department
- Customer Support Department

The Product Planning Department has played the role of representative and carrier of the needs of all the

customers, collecting and coordinating requests and needs directly and through the Customer Support Department.

Three documents were produced applying the innovative approach: the User Requirements regarding the Single SISPAC Application Module User Interface, the Printers Control User Interface, and the SISPAC/NT Desktop User Interface. (See Section 5.2 for measurements regarding the results).

7.2 Organizational, cultural, and technical difficulties. How were these difficulties overcome?

No specific organizational difficulties were encountered, because corporate management did sponsor the effort.

Cultural difficulties were approached. We provided to people a specific two-days training on the contents and motivations of the documents through the life cycle, with special emphasis on the User Requirements document. This training has given to the actors of the process a full understanding of the role of the different kinds of requirements in the different phases of the project and on the corresponding process oriented culture.

Technical difficulties (in this case the risk of having a not uniform approach by different people in the preparation of the different contributions for requirements) were solved adopting the common document template described above.

The above mentioned training accompanied by the use of the interactive supporting tool allowed people to be quickly operational in writing complete, understandable, controllable user requirements.

7.3 Costs and benefits

Costs:

- 1) The training: this cost was required anyway within the ongoing effort of establishing the OSRA Quality System.
- 2) No specific additional costs, due to the innovation (except the acquisition of the tool, some hundreds US dollars), were sustained in the writing of the documents.
- 3) Costs of measurements. Less than 0,2 % of additional cost, for counting the requirements: collection of time/effort and defect data was already an established practice.

Benefits:

- 1) A greatly increased involvement and cooperation with both external and internal (Product Planning) customers;
- 2) A clearer understanding of the current customer's process and of their future process, and correspondingly of the current and of the new SISPAC systems;
- 3) The new project seen in the context of user process, being the user process a reference to be used for evaluating the completeness, consistency and adequacy of the requirements of the new system;
- 4) a much more professional image toward the external customers;
- 5) greater opportunity of handling conflicts of requirements, because customer's needs are clearer;
- 6) A good premise for reducing future post-sale problems.

7.4 Learned Lessons

The main lesson learned regards the better control over product and process quality that can be obtained through three main means:

- a) the use of process model and related document template for getting guidance in the work
- b) the use of a shared and controlled database as repository where to keep the baseline versions of the results, which makes easier to the participants the access to the updated information, with the guarantee of working on the correct and controlled version.
- c) the collection of requirements, defects and time/cost data for producing effectiveness and efficiency indicators

Using low-tech tools and process improvement methods to collect and distribute information is highly effective, particularly for the small companies (a huge population of the software industry) because it minimizes the risks of adoption, the cost of using high tech tools, and the need of training [10].

As future development the usage of Internet/Intranet is foreseen as the mean to make available the above communication channel also directly to the customers, enhancing so their involvement.

8. OSRA SISTEMI Case: notes and comments

Results of this experience were presented also as a case study to the European ESPRIT Project 21461 TBPTIME, having as objective to collect and diffuse for training purposes a set of significant case studies of process improvements for SMEs in different process areas.

The significant aspects pointed out in the case study are the following:

- CASE GOALS:

- to show how a SME is facing to the marketing problem of client needs identification and analysis
- to show how the employment of IT, also not so much sophisticated, could improve the Customer's needs management process part of the Customer management process
- to show how a SME could utilize IT to collect information about clients
- to show how a SME could utilize IT to share information in the organization
- to show the importance of the process quality management

- PROCESSES CONCERNED

- Customer management Process
- Production process
- Information management process
- Quality management Process

- IT CONCERNED

- SW (word processor) + pc
- LAN, DB
- Workgroup
- Internet (future)
- e-mail (future)
- Intranet (future)

- THE OSRA MODEL

The OSRA case represents and shows a model that could be exported in other contexts, both service and product oriented. The lesson learned could be summarized in the following picture (Figure 7):

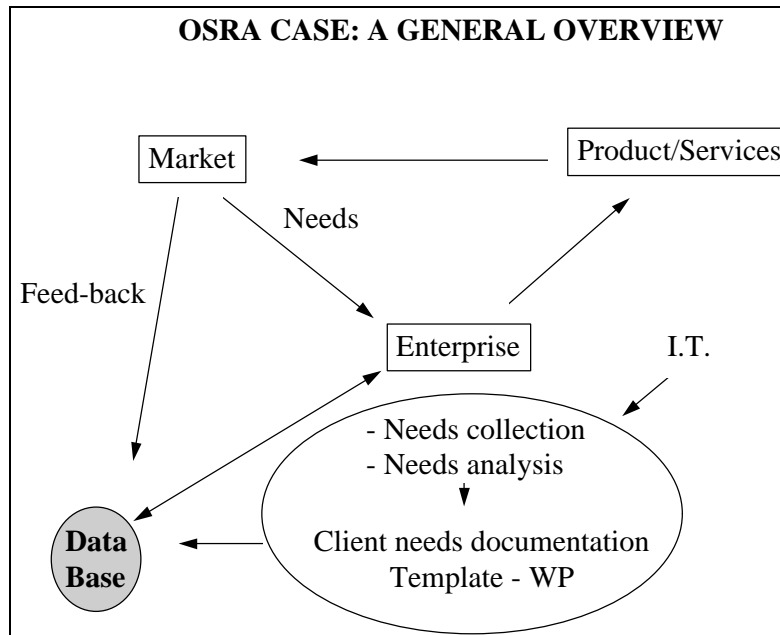


Figure 7

SMEs should manage the Client needs using the techniques available. The IT could support the SMEs efforts providing Data-Bases and the tools to manage data. The data collection with IT based tools could help the SMEs in the diffusion of information into the organization. The introduction of this innovation must be strongly supported by the corporate management with a top-down approach: without this involvement there are huge difficulties to modify a process, or part of a process [6]. The Marketing aspect of the innovation consists in the collection and organization of data about client needs and in the possibility to check the market and the customers through the development of the client documentation Data Base. The use of a common template assures the same standard in the data collection and avoids misleading interpretation to be inserted into the D.B. But the Client Needs documentation is also the starting point of the production process. So it is necessary that all the information are diffused in the SMEs. The common way to do that is the production of a paper (e.g. a marketing plan). In this case the Information System allows certain number of departments, the departments involved in the innovation, an inter-functional team, to access the Data Base and to share the knowledge about client needs.

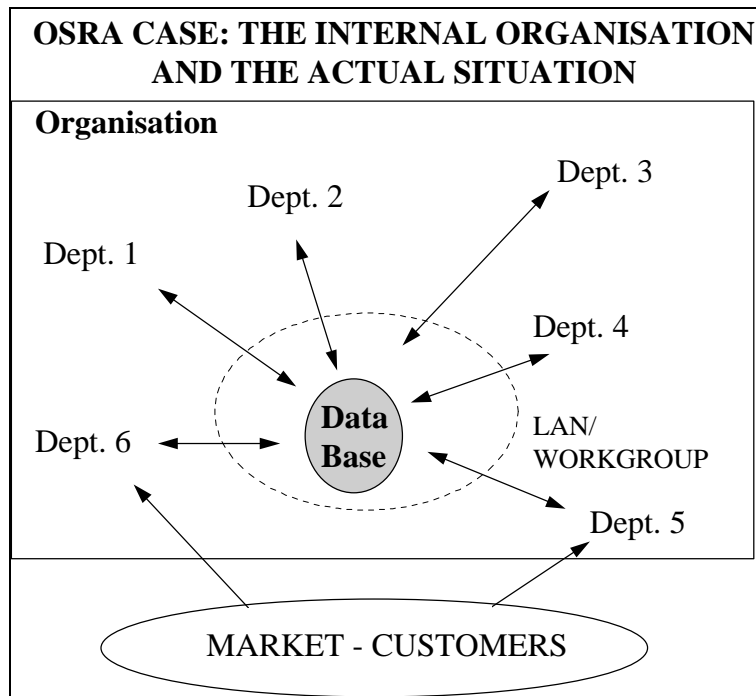
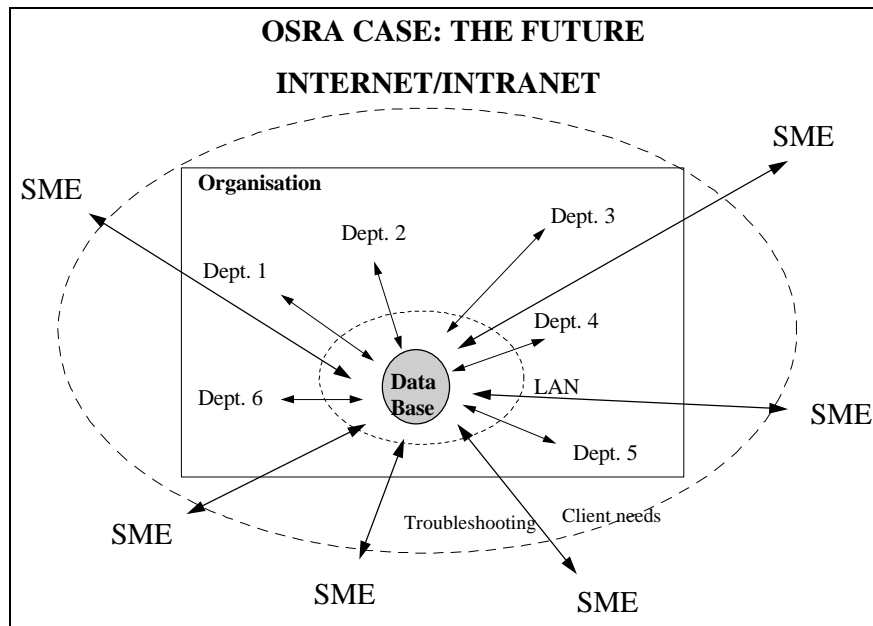


Figure 8

In this way it is also possible to enrich the Client Documentation Data-Base with all the feed-backs that are coming from the market and the customer. So the Data-Base is the real focal point not only of the development of the actual product but also of the future releases and versions. With the implementation of LAN and Work-Group techniques it is possible to do that (Figure 8)

The future development of this model is towards the implementation of a Internet/Intranet innovation. The Data Base could contains not only the Client Needs but also other information, such as troubleshooting, product test results, etc. A certain number of customers (a test panel for example) could be interested in the access to the Data Base for the automation of the Client Needs Documentation activity (through e-mail) or for the troubleshooting activity (through Internet/Intranet). (Figure 9).

Figure 9



Finally the Client Needs Documentation Activity is integral part of the Quality System of the SMEs and the IT permits the measuring of performances of the process. This is really important not only for SMEs that make products but also is fundamental for SMEs producing services. In the Software development area, as for the OSRA case, it is now fundamental to assure high level of quality and the possibility to monitor the real performances, both of the software and of the organization.

To explain the goal that OSRA SISTEMI has in the adoption of the IT innovation we have to think that in the actual market condition the competition requires quality, efficiency and short time-to-market. The quality aspects must consider two factors: the total quality of the product and the cost factor. The efficiency must be considered in terms of cost of a particular project/product and the productivity of the organization. These two factors could improve the time-to-market and could give the “first move advantage” to the SME. Advantages derived from a short time-to-market are the possibility to have fast feed-back from customers that could be structured in a data base for the future development of the product. In the OSRA CASE the customer’s feedbacks were used for taking care of the aspect of “user-friendliness” of the product.

9. Conclusions

The literature of the Total Quality Management has always stressed the importance of collecting and using needs and feedbacks from the (internal and external) customers [7]. We believe that this case study can be a practical and successful example about the structured collection of the requirements for the new products. In the software field, in particular, the complete, consistent and structured specification of user requirements is the critical starting point to deploy products of the required quality [8], [9]. Moreover the technologies and the methodologies used in this case study could be exported in other contexts, both service and product oriented, to structure the development of technical requirements.

References

- [1] ISO/IEC 12207, Information technology - Software life cycle processes, First edition, August 1, 1995
- [2] ISO/IEC PDTR 15504-1/9: Information Technology - Software Process Assessment Parts 1 through 9, November 13, 1996
- [3] J-STD-016-1995 EIA/IEEE Interim Standard for Information Technology - Software Life Cycle Process - Software Development Acquirer-Supplier Agreement (Issued for Trial Use)
- [4] ami (Application of Metrics in Industry), Metric Users’ Handbook - A quantitative approach to software management -

- The ami consortium c/o The ami User Group, CSSE, South Bank University, 103 Borough Road, London SE1 0AA, UK
- [5] Basili V.R., Rombach H.D., 1988, "The TAME project: towards improvement-oriented software environment", IEEE Transaction on Software Engineering, vol.14, n. 6, June, pp. 322-331.
- [6] Thomas H. Davenport "Process Innovation. Reengineering Work through Information Technology, Harvard Business School Press, Boston, Mass., U.S.A., 1993
- [7] Crosby P., 1979, Quality is Free, McGraw-Hill, New York, USA.
- [8] Humphrey W.S., 1989, Managing the software process, SEI series in software engineering, Addison Wesley Publishing Company, Reading, Massachusetts, USA
- [9] Paulk M.C., Curtis B., Chrissis M.B., Weber C.V., 1993, Capability Maturity Model for Software, Version 1.1, SEI Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.
- [10] Agresti W.W., 1991, "Low tech tips for high quality software", IEEE Software, vol. 7, n. 6, November, pp. 86-88
- [11] ProDoc, a software documentation support tool, produced by Software Productivity Centre (SPC) (Vancouver, Canada), distributed in Italy by MetriQs - For information, contact SPC (to tools@spc.ca), or MetriQs (to support@metriqs.com)
- [12] ISOplus, a tool supporting the preparation of a ISO 9001 compliant Software Quality System, produced by Software Productivity Centre (SPC) (Vancouver, Canada), distributed in Italy by MetriQs - For information, contact SPC (to tools@spc.ca), or MetriQs (to support@metriqs.com)
- [13] Fabio Valle, The software quality: a new paradigm in the software industry and the case study SPICE, Master Engineering Degree Thesis, University of Padova, 1995-1996.

Attachment:

Appendix A - Some samples of what the support tool provides

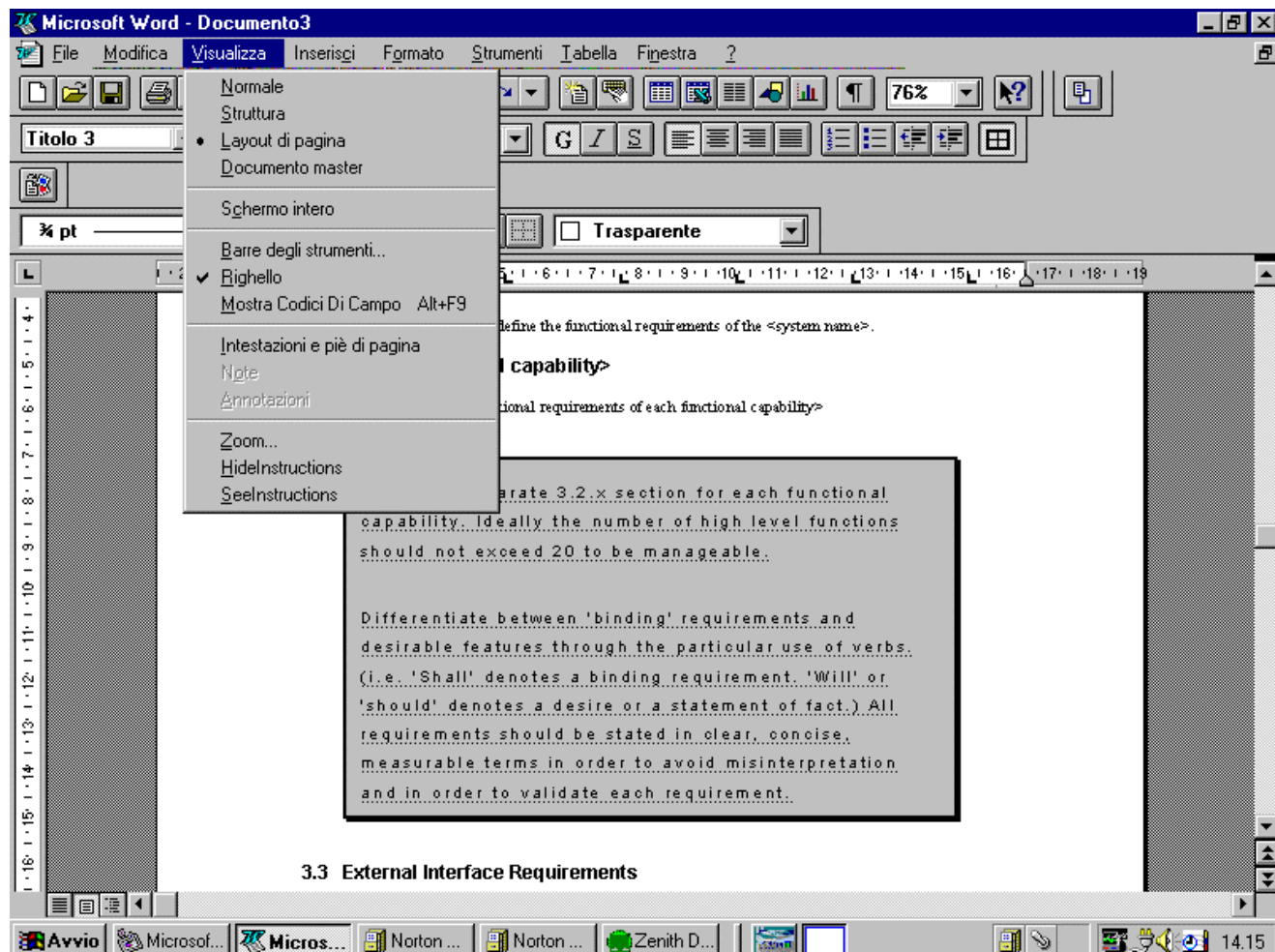
Managing customer's requirements in a SME: a process improvement initiative using an IT-based methodology and tool.

Some samples of what the support template and tool provide.

This Appendix provides:

- a copy of a Word window showing a portion of a ProDoc [11] template. In the pulldown Visualize menu, the commands "HideInstructions" and "SeeInstructions" (one of the characteristics of ProDoc) allow the user to visualize or to hide the gray boxes of the template. The boxes contain explanations for helping the user in understanding better the topics to be addressed in a given section of the template.
- a view of one sample page of one of the documents produced in the experience, where the help boxes are visualized.

View of the Menu commands "SeeInstructions" and "HideInstructions" of the support tool ProDoc



What follows in this page is a view of one sample page of one of the documents produced in the experience, where the help boxes are visualized.

1. Scopo

Questa sezione deve essere divisa nei seguenti paragrafi.

1.1 Identificazione

Questo paragrafo deve contenere una completa identificazione del sistema a cui si riferisce questo documento, includendo se identificabile il numero d'identificazione, titolo, abbreviazioni, versione e numero di release.

Questo documento descrive le esigenze e le possibilità della nuova interfaccia con sfondo grafico da adottare nell'applicativo SISPAC relativamente alla gestione delle stampanti.

1.2 Visione d'insieme del sistema

Questo paragrafo deve enunciare brevemente lo scopo del sistema a cui si riferisce questo documento. Deve descrivere la natura generale del sistema; deve riassumere la storia dello sviluppo del sistema, funzionamento e manutenzione; deve identificare lo sponsor del progetto, l'acquirente, l'utente, chi ne cura lo sviluppo e le procedure di supporto; deve identificare i luoghi operativi attuali e quelli pianificati; deve elencare altri documenti rilevanti.

Lo scopo del sistema è quello di proporre una più gradevole interfaccia di SISPAC proponendo uno sfondo grafico alle attuali videate di gestione della stampante (e del documento).

1.3 Visione d'insieme del documento

Questo paragrafo deve riassumere lo scopo ed i contenuti di questo documento e descrivere le considerazioni di sicurezza o privacy associate al suo utilizzo

Questo documento descrive, in linguaggio utente, le problematiche relative al processo di simulazione grafica di un'interfaccia a caratteri, nonché le caratteristiche di efficienza ed efficacia rispetto alla attuale situazione.

2 Documentazione di riferimento

In questa sezione si devono elencare il numero, il titolo, la revisione e la data di tutti i documenti a cui si fa riferimento in questo documento.

THE EURO CONVERSION

MYTH VERSUS REALITY!

A Special Panel Session

Quality Week Europe

Conference Day #2

12 November 1998 @ 1600

Chaired by Thomas Drake, Coastal Research & Technology, Inc.

Panelists

John Corden, Cyrano
Patrick O'Beirne, Systems Modelling Ltd., Consultant
Jens Pas, ps_testware
Graham Titterington, Ovum

The Euro Conversion – Myth vs. Reality!

This special panel session is designed to provide a forum for discussing the Euro conversion with a decided focus on the technical, economic, cultural, and liability concerns posed by the Euro conversion. Questions that will be presented and discussed include the following:

- What are the real-world challenges and experiences of those currently working the Euro problem?
- Who and what will be impacted by the Euro conversion?
- What are the facts about the Euro conversion versus what is myth?
- Is the Euro conversion the ultimate millennium challenge for Europe?
- Are the business and technical challenges posed by the Euro conversion more difficult to resolve from those surrounding the Year 2000 problem and what difference, if any, can quality make?

The Euro conversion would appear to pose not only economic challenges but also a number of technical and business obstacles and issues, and perhaps most importantly a degree of risk involving historic proportions. The core period for the Euro conversion is marked by transition over a three-year period beginning on 1 January 1999 and the numerous technical and managerial challenges posed by the Euro conversion appear very similar to the Year 2000 problem.

There are a number of common issues relating to applications and package software, source code, service vendors, testing, program management, information technology resource support, domain expertise, and time! But what is unique about the Euro conversion? Perhaps the most critical aspect of the conversion lies in the area of requirements management.


Every business process appears related to the Euro issue and involves far more than just a technical upgrade and modification of existing business and enterprise systems. More importantly are the strategic decisions that effect how businesses conduct their enterprises each and every day. The Euro poses perhaps the greatest challenge in these areas and perhaps the greatest impact of the Euro conversion will occur at the retail level in the buying and selling of goods and services each and every day.

There are also technical risks and obstacles associated with the Euro conversion. Consider the problem of possible data pollution and corruption, conversion errors, and display problems. The data migration paths posed by the Euro conversion encompass a whole host of challenges for testing, quality assurance, and configuration management including database conversions, numeric translations, and modified pricing structures.

And what is the definition of Euro conversion compliance and how does one test for it? Some are even saying that the Euro conversion is actually more complex and has greater impact than even the Year 2000 problem. Why? It will implicitly shift and even perhaps radically alter the day to day lives of the people affected and have a major and lasting impact on all the business and institutions who invest and trade and engage in daily economic transactions within Europe and from without Europe vis-à-vis Europe.

The primary intent of this panel session is having a facilitated discussion among the panel members and the audience on the impact, change, and reality posed by the Euro conversion and explore the question of what does it all mean from a technical, managerial, and information technology perspective.

Slide 1



Introducing structured testing into a dynamic, low-mature organisation

Track Presentation at Quality Week Europe '98
Mark Buenen, GiTek Software n.v.


GiTek Software n.v. (QWE '98) 1

Slide 2

Introducing structured testing into a dynamic, low-mature organisation Structure

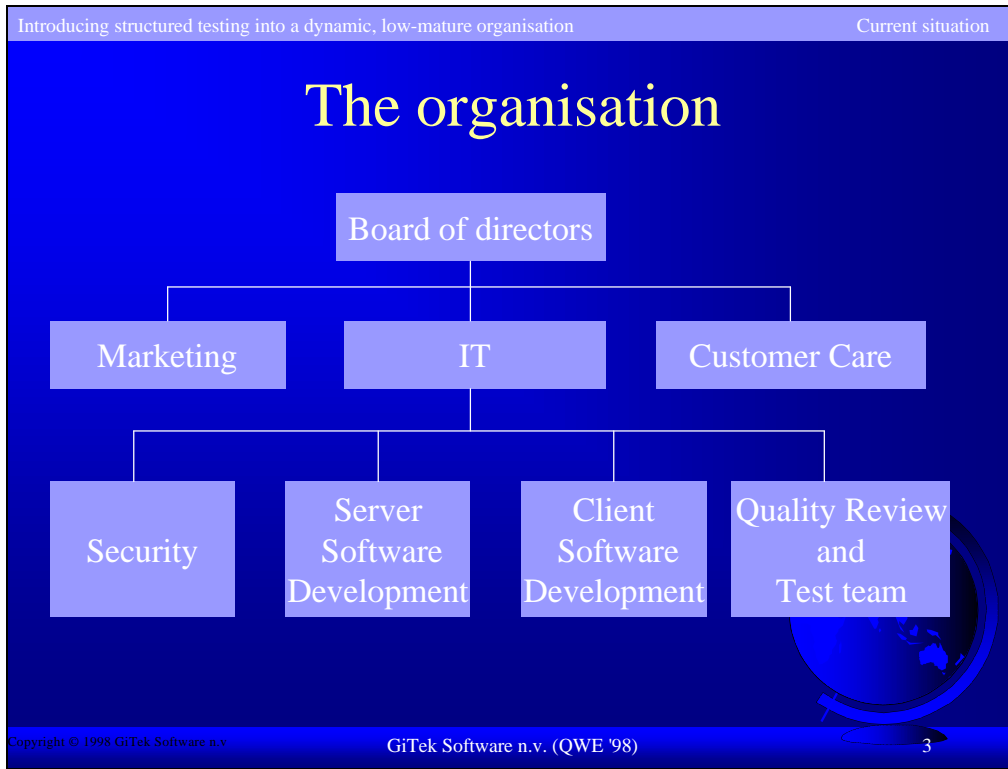
Structure

- ◆ Current situation
- ◆ Plan for introducing structured testing
- ◆ Execution of improvement actions
- ◆ Evaluation
- ◆ Conclusions

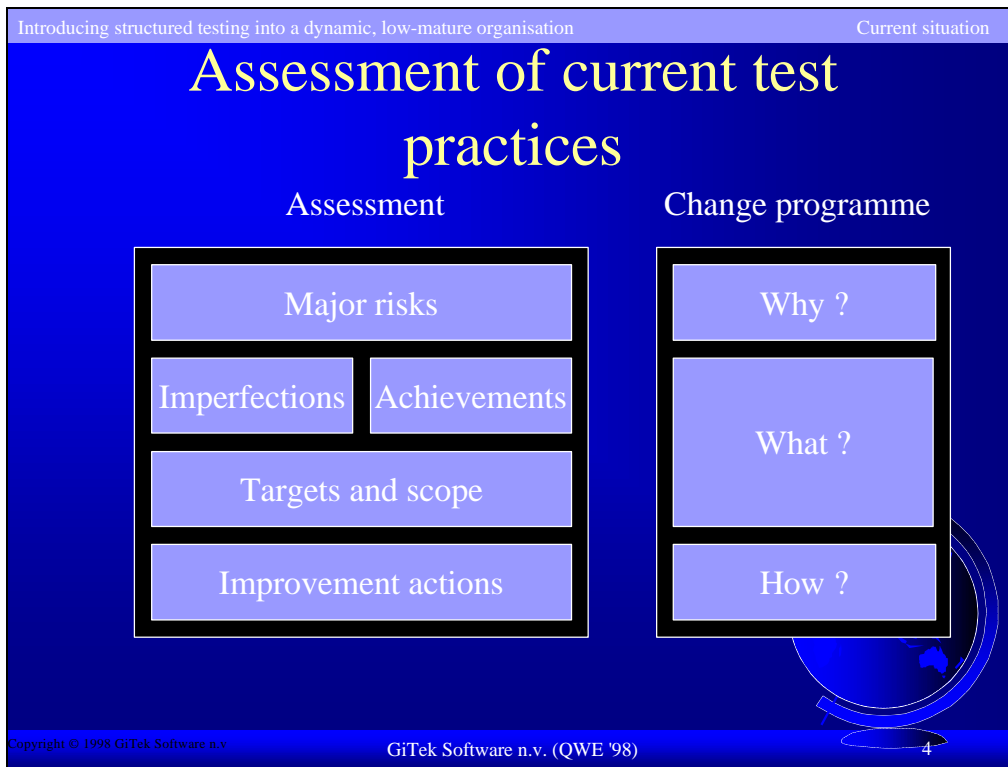


GiTek Software n.v. (QWE '98) 2

Slide 3



Slide 4




Slide 5

Introducing structured testing into a dynamic, low-mature organisation Assessment

Major risks

- ◆ Focus only on on-time delivery
- ◆ Quality level of the product is lacking
- ◆ Immature development process




Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 5

Slide 6

Introducing structured testing into a dynamic, low-mature organisation Assessment

Imperfections

- ◆ Formal specifications are lacking
- ◆ Lacking insight in white box tests
- ◆ Too many successive releases
- ◆ Poor preparation of black box tests




Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 6

Slide 7

Introducing structured testing into a dynamic, low-mature organisation Assessment

Positive trends

- ◆ Management commitment
- ◆ Separation of white box and black box tests
- ◆ Presence of a dedicated test team
- ◆ Presence of a quality team
- ◆ Use of test tools




Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 7

Slide 8

Introducing structured testing into a dynamic, low-mature organisation Assessment

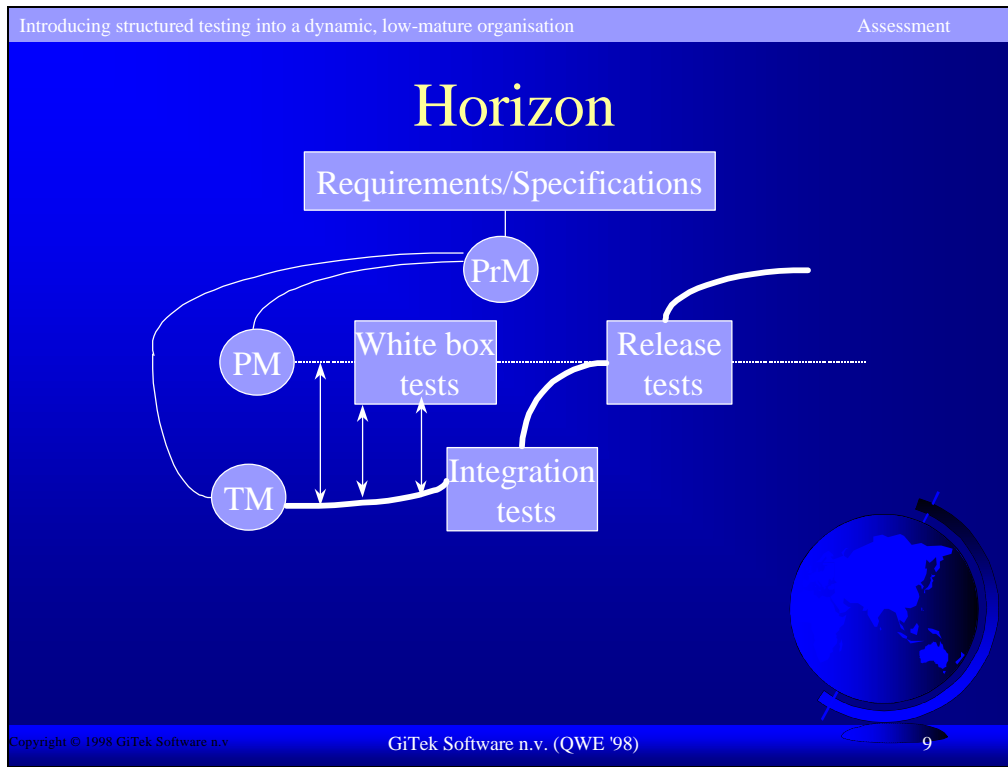
Improving the test process

- ◆ Implement a test management approach
- ◆ Create a risk based test strategy
- ◆ Formalise test methods, techniques and tools
- ◆ Formalise test procedures
- ◆ Improve white box test
- ◆ Arrange training and coaching



Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 8

Slide 9



Slide 10

Introducing structured testing into a dynamic, low-mature organisation Proposal

Proposal

- ◆ Phase 1:
 - Test methodology
 - Test organisation
 - Test infrastructure
 - Training and Coaching
 - Pilot projects
- ◆ Phase 2:
 - Test automation

◆ 80 man days

Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 10


Slide 11

Introducing structured testing into a dynamic, low-mature organisation

Proposal

Suggestions

- ◆ Get key-players involved
- ◆ Recognize roles
- ◆ Do not overlook positive trends
- ◆ Await the go/nogo decision



Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 11


Slide 12

Introducing structured testing into a dynamic, low-mature organisation

Plan

Implementation plan

- ◆ Testing methodology
- ◆ Testing organisation
- ◆ Testing infrastructure
- ◆ Training and coaching
- ◆ Pilot projects




Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 12

Slide 13

Introducing structured testing into a dynamic, low-mature organisation Plan

Suggestions

- ◆ Get others involved
- ◆ Keep IT-management informed
- ◆ Communicate the plan to all parties




Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 13

Slide 14

Introducing structured testing into a dynamic, low-mature organisation Execution

Execution

- ◆ Describing the workflow
- ◆ Challenges
- ◆ Suggestions



Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 14

Slide 15



Slide 16

Introducing structured testing into a dynamic, low-mature organisation

Execution

Challenges

- ◆ Coverage tool
- ◆ Defect tracking tool
- ◆ Test environment
- ◆ Developers lack time to participate

Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 16


Slide 17

Introducing structured testing into a dynamic, low-mature organisation

Execution

Suggestions

- ◆ Acknowledge resistance
- ◆ Be flexible but firm
- ◆ Use sponsors
- ◆ Check roles of the players
- ◆ Look for quick wins
- ◆ Communicate the progress



Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 17


Slide 18

Introducing structured testing into a dynamic, low-mature organisation

Evaluation

Evaluation, results achieved

- ◆ Test efficiency improved
- ◆ Co-operation with developers
- ◆ Re-usable and flexible test scripts
- ◆ Improved testing quality



Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 18

Introducing structured testing into a dynamic, low-mature organisation

Evaluation

Evaluation, to be resolved

- ◆ Changing the rest of the organisation
- ◆ Formal cooperation with developers
- ◆ Improving designs
- ◆ Spread the achievements




Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 19

Introducing structured testing into a dynamic, low-mature organisation

Evaluation

Continuation

- ◆ Reduce speed of further improvements
- ◆ Introduce a phase of preservation and communication
- ◆ Pick up the improvements later on



Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 20

Introducing structured testing into a dynamic, low-mature organisation Conclusions

Conclusions, the phases

```
graph TD; A[Assessment] --> B[Proposal]; B --> C[Plan]; C --> D[Evaluation]; D --> C; C --> E[Improvement]; E --> D;
```

Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 21

Introducing structured testing into a dynamic, low-mature organisation Conclusions

Conclusions, lessons learned

- ◆ Recognize role of communication
- ◆ Improvement phases 6-12 months
- ◆ Quick wins
- ◆ Simplicity over thoroughness
- ◆ Flexibility and firmness
- ◆ Endurance

Copyright © 1998 GiTek Software n.v. GiTek Software n.v. (QWE '98) 22

Introducing structured testing into a dynamic, low-mature organisation

Track Presentation at Quality Week Europe '98.

**Mark Buenen (GiTek Software n.v.)
18-09-1998**

1 Introduction

This presentation is an account of the practical experience obtained in a Belgian organisation during the execution of a program for structuring the testing approach. In this young and very dynamic organisation the on-going business demanded major attention of all employees. This puts a lot of restrictions on the feasibility of any change program. This presentation will show how this dilemma was recognised and was dealt with in practice. The presentation offers attendees an example of a successful implementation of structured testing. This provides attendees with a reference for their own change programs, and the benefit of getting acquainted with major pitfalls and lessons learned in practice.

The presentation has the following structure:

- The reasons for introducing structured testing in this organisation;
- Planning the improvement actions;
- The implementation of the improvement actions in the first phases;
- The results achieved during the first phase;
- The planning of the next phases;
- Conclusions and lessons learned.

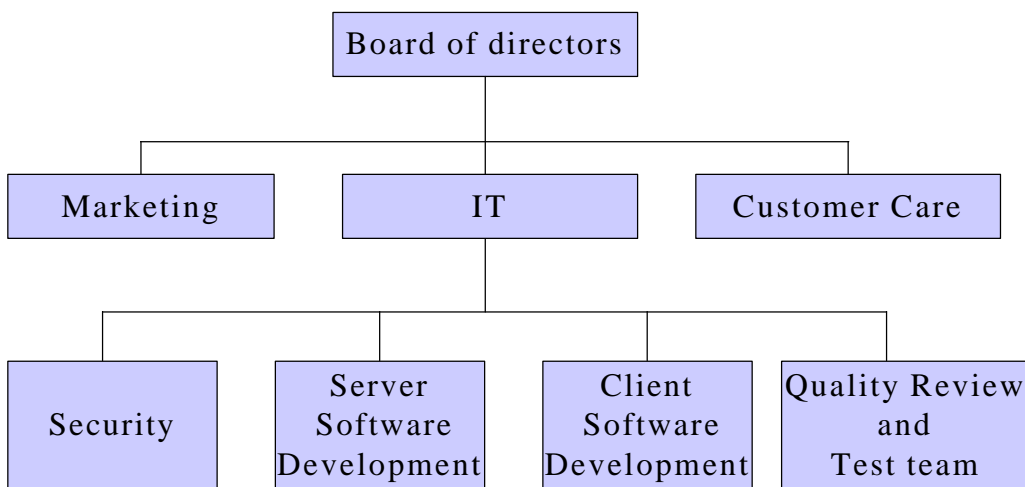
The testing methodology Test Management Approach (TMap®), was used as a reference for this test structuring program.

2 The current situation

2.1 Description of the organisation

The organisation in case is a young company, that has been building a package for the Belgium financial world. The major banks of Belgium provide this organisation with assignments. End users are the business clients of the banks. The package has to work together with the information systems of the banks. The organisation also provides access to a Wide Area network, with couplings to the Internet.

The organisational structure is typical for a package developer. It is depicted by the following diagram:



The first challenge was to meet the date agreed for releasing the first version, with an acceptable level of quality. Since that first release there has been a continuous demand for additional features. Together with the resolution of some technical problems this has resulted in a continuous flow of releases of new versions, and a growing time-to-market challenge. General management recognised the threat of the insufficient quality level of both the product and the development process, and decided to introduce a quality department. The quality manager department felt the necessity to introduce structured testing for final acceptance tests. The first action was constituted by a quick 1 day assessment in order to determine the scope of the improvement program.

2.2 Scope of the test structuring program

The assessment consisted of interviewing the key-players, and studying relevant documents such as testing and development guidelines. In general we recognise the following key-players: IT-management, project management, quality management, end-users, developers. In this case the following key-players participated:

- **The IT-manager**

The IT-manager is the one who commissioned to introduce structured testing. This person decides on releasing budget for projects. The main areas of attention are:

What goals does this person hope to achieve by introducing structured testing;

Based on what results does this person consider the program to be a success;
 Who else have to be involved in this assessment ?

- **Project managers software development**

Project managers are responsible for software development. They provided valuable insight in the current state of the software development process and of the testing process.

- **Quality Manager**

The quality manager is responsible for introducing and maintaining quality control in the development program. In many small organisation this is not a full-time job. Sometimes this role is fulfilled by the IT-manager. The quality manager was asked for his opinion on the current state of quality assurance measures within the organisation (specification reviews, development standards etc.).

- **Developers and testers**

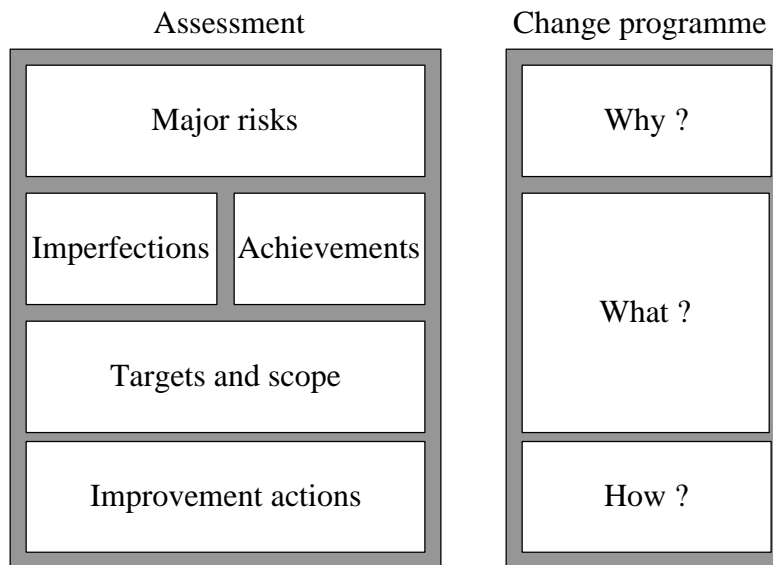
The developers provided valuable insight in the actual application of development and testing methodologies in practice. It is very revealing when you are able to observe test-execution in practice.

The product

The product of this first phase is a report describing the current situation. Key-elements of this report are:

- Major risks in terms of business targets. Recognition of these risks in an early stage has two advantages;
 1. It helps you to focus your improvement actions on the really important issues.
 2. When you encounter resistance it enables you to convince people of the importance and necessity of the changes. Identifying the major risks creates the awareness for the necessity of a program of change.
- Imperfections of the development and testing process. This identifies the elements that need to be improved.
- Positive trends and achievements. These are the cornerstones of the improvement actions. Keeping these items and building on them will help to minimise resistance. What must be kept ?
- Goals and scope of the program of change: Describe the basics of the change actions. What imperfections will be addressed by improving the testing activity. Which achievements will be used as a starting point ?
- Improvement actions. This is a high level description of the action. It gives management insight in the direction of the improvement actions.

The objectives of this product are depicted in the following figure:



2.3 Results of the assessment

The key reasons for introducing structured testing in this organisations were:

- The core business of this organisation is delivering a software package and providing access to their server. The mere existence of this organisation depends on an appropriate level of quality of the delivered software and infrastructure;
- The current software development and testing process is unpredictable and time consuming. It provides too little insight in the quality level of the products;
- Too many errors in the software after installation on end users work stations;

The major risks identified were:

- The constant time pressure to deliver software, whether it is new or altered;
- The informal process of obtaining the requirements and specifications. The input is not only given by banks, but also by testers, user help desk, installers, standards, legislation, technical requirements etc. By lack of Product Management resources, IT staff mostly takes the responsibility for the specification process;
- An unpredictable and time consuming testing process, caused by the lack of scheduling, metrics, estimation, test organisation and test management.

Some of the imperfections were:

- The white-box tests are performed very informally and the process offers no insight;
- There are too many successive software releases;
- The problem reporting and change procedures are not formalised. The Test Team is not informed on a structural basis of the problem reports;

The positive trends were:

- Within the IT-department and at management level there is already an awareness that the test process needs to be improved. The implementation of a test methodology is part of a company wide quality improvement process;
- The test environment is set up in such a way that there is a separation of development test, functional tests, beta tests and production;
- The IT Department is contemplating about future ideas, such as the reusability of the testware and test packages;
- The Management agreed to create a test laboratory for functional tests.

Based on these findings a step by step improvement of the test process was proposed. Therefore the improvement actions were defined and grouped in short term, medium long term and long term:

Short term:

1. Give the Test Team the necessary responsibilities in both preventive and detective matters, fitting with the total process flow;
2. Implement a test organisation and establish the accompanying test functions;
3. Recruit the necessary test personnel and take care of adequate training and coaching;
4. Implement a generic, proven test management approach. Mind not to reinvent the wheel again.
5. Formalise the applied test methods, techniques and tools.
6. Formalise the test procedures for all tests. These procedures ought to work as a protocol for all disciplines involved;
7. Agree on quality level before the modules are presented to the Test Team for functional tests;
8. Create a risk based test strategy early in the development life cycle to avoid uncontrollable parallel test execution processes and to reduce the large number of beta tests;
9. Install a communication platform where test planning, test specifications, defects and change control can be discussed;
10. Improve the white box tests and arrange training and coaching;
11. Implement tools for management and defect registration;

Medium long term

1. Organise the specification process by adequate change control procedures and the installation of product management;
2. Arrange the actual execution of the white box tests to be reviewed by the Test Team;
3. Define and install testware management and make testware reusable;
4. Define, collect, analyse and provide test metrics on a regular basis;

5. Use tools to automate the integral test process (management, execution and analysis);
6. Use the achieved test experience and methods company wide;

Long term

Integrate the applied test methods, procedures, testware and experience to a professional test process executed in a test laboratory, that enables the organisation to execute any test effectively, quickly and at low costs. In the future such a laboratory could be exploited as a profitable unit.

In that situation the requirements will be collected and fixed by Product Management of course supported by Information Analysts and Functional Designers of IT as well as by Business Experts. Based on the specifications Project Management will run the project to design, realise, (development) test and implement the system, and Test Management will take care of the organisation, preparation, specification and execution of the integration and system level tests. This process will be suitable for both new developments and for maintenance.

2.4 Hints

- The assessment is required to help the sponsor in deciding to commission the change program;
- Do not overlook the positive trends and achievements of the current approach;
- Get all important people involved in this early stage. The first step is interviewing them during the assessment. The second step is discussing this report as a separate product of the change program;
- Recognise the roles of influential people: who are supportive, who are change agents, who are blocking factors. This helps you to determine a strategy for the actual program of change;
- Be sure that this phase is concluded by a formal go/nogo decision by your sponsor.

2.5 The proposal

After acceptance of the assessment report a proposal was drawn up, in which the required budget and the important conditions were discussed.

Based on the discussion the assessment report it was decided to tackle the following issues:

1. The testing process must supply insight in the quality level of the product as early as possible in the life-cycle;
2. Planning and scheduling of the duration and resources required for the test process must be improved;
3. The number of releases must be reduced and control must be improved;
4. White box tests must be improved and must provide more insight;
5. The application of the test automation tool must be improved;
6. Defect registration and change control procedures must be formalised;
7. Communication between disciplines must be improved.

The improvement actions were separated in two distinct phases:

First phase: December 1997 - March 1998

Second Phase: April 1998 - June 1999

It was decided to deal with the prerequisites for a structured test process in the first phase:

1. Test methodology;
2. Test organisation;
3. Test infrastructure;
4. Training and coaching of involved employees;

5. Application of the new methods techniques, infrastructure and utilities in pilot projects.

During the second phase the focus would be on optimising the achievements of the first phase.

A global estimation of the total required effort of the first phase of the improvement project is 80 man days.

3 The plan for introducing structured testing

3.1 Introduction

The plan is a means for communicating the intended change program to people directly involved in the improvement process. And it is an aid for monitoring the change program by the change manager. The plan has the features of any project plan. Standard items within this plan are:

- Introduction;
- Horizon;
- Improvement Actions;
- Activity planning;

3.2 The horizon

The objective of the test structuring program was to achieve a test laboratory, where tests are performed in a structured way. The main features of the structured testing process are:

- There is a co-ordination of the scope and depth of the various tests (unit test, integration test, system test, acceptance test, beta-tests etc.). This is achieved by creating a master test plan. The master test plan is based on a validation and verification strategy, that is based on a risk taxation of the application and its quality criteria;
- The presence of a formal quality review of functional analysis documents that are used as test basis;
- Verification of the quality of delivered software to the test team, by providing insight in the contents and results of the white-box tests;
- Separation of the functional tests by the test team in phases : planning, preparation, specification, execution and consolidation;
- Scheduling the tests to be performed on the basis of release-plans that cover a period of 6 months;
- A supplementary test strategy for maintenance.

3.3 Improvement actions

In the plan the improvement areas, mentioned in the proposal, are detailed in further actions:

Testing methodology.

A structured test process requires well-defined guidelines describing the testing activities, supplemented with templates for testing products such as test plans, test specifications, test scenarios, test reports, defect tracking form, standard checklists and a clear description of various procedures, especially those which require the involvement of other disciplines: defect tracking procedure, version control procedure, change control procedure, specification review. The proposed actions are:

- Drawing up a master test plan;
- Drawing up testing guidelines and procedures;
- Implementation of a defect control procedure;
- Implementation of a change control procedure;
- Implementation of a configuration control procedure;

- Implementation of quality assurance of functional specifications;

The proposed actions for improving white-box tests were:

- Selection of white-box testing techniques;
- Creating a reference card for white-box testing
- Introducing statement coverage analysis;
- Introducing branch coverage analysis;
- Introducing quality assurance on white box tests.

Probing the test methodology

Within this improvement the following actions were mentioned:

- Drawing up a functional test plan;
- Executing the functional test plan for the pilot project;
- Evaluating of the functional tests;
- Building regression test ware;
- Collecting metrics

Testing organisation.

The objective of this activity is to create a professional testing organisation, with clear responsibilities and authorities.

- Describing the roles and functions of the Test Team;
- Assigning tasks to persons;
- Installing the decision forum;
- Defining standard report procedures;
- Describing agreements with other disciplines (developers, marketing, banks) in formal agreements;
- Installing the control of test ware, testing guidelines and metrics;
- Installing internal testing quality assurance measures;

Testing infrastructure.

More than once the creation of adequate testing environment has proven to be a major challenge. This often takes more time than expected. Acquiring and installing extra hardware and tools requires the effort and approval of many departments and disciplines. Building a new environment is a painstaking business, especially because it requires the realisation of couplings with other information systems, and creating and filling test databases and setting parameters. Therefore careful planning of the activities for this area is important. Also the selection and installation of various tools (defect tracking, coverage analysis, test automation, version control) requires much effort. The detailed actions are:

- Defining requirements for the test environment;
- Creating the test environment;
- Selecting and implementing of a defect tracking tool;
- Selecting and implementing of a coverage analysis tool;
- Designing and building the base structure of the automated test suite;
- Applying the test automation tool in pilot projects;
- Selecting and implementing of a configuration management tool;
- Selecting and implementing of a project management tool;

Training and coaching.

During the process more and more people will have to be trained in applying new techniques and procedures. This area needs various sub-steps: creating awareness (workshop to introduce the concept of structured testing), training programs: explanation and exercises to get acquainted with the new working methods, coaching: training on the job is a very important success factor for introducing structured testing. The detail actions are:

- Workshop structured testing;
- Training white box testing techniques;
- Training black box testing techniques;
- Coaching during pilot projects;
- Creating and maintaining organisation specific training courses;

For each action the following items are described in the plan;

- Objective: what do we expect to achieve with this action;
- Deliverable: the tangible result of the action: a plan, a template, a procedure-description etc.;
- Who need to be involved in this action;
- Indication of the importance of the action;
- Dependence of the realisation of other improvement actions.

3.4 Activity planning

The improvement actions are distributed among the two proposed phases of the improvement process. This is shown in the next table:

Improvement Area	Phase 1 Dec. 97 - March 98	Effort	Phase 2: April 98 - June 99
Testing methodology	Master test plan	8	Configuration control procedure Change control procedure Quality Assurance specifications Quality Assurance WB tests
	Testing guidelines and procedures	9	
	Defect control procedure	1	
	Selecting WB testing techniques	4	
	Reference card WB test	2	
	Coverage Analysis	4	
Probing Methodology	Drawing up functional test plan	5	Building regression test ware Collecting metrics
	Executing the functional test plan	PM	
	Evaluating the functional test	3	
Test Organisation	Describing of roles and functions	4	Agreements with other disciplines Installing test control Installing test quality assurance
	Assigning tasks to persons	4	
	Installing the decision forum	4	
	Defining report procedures	3	
Test Infrastructure	Defining test environment	2	Design and build base structure of automated testsuite Apply test automation in pilot Implement version control tool Implement project management tools
	Creating test environment	6	
	Implement defect tracking tool	3	
	Implement coverage analysis tool	3	
Training and coaching	Workshop structured testing	5	Coaching during phase 2 Maintenance training
	Training white box testing	5	
	Training black box testing	5	
	Coaching during pilot	PM	
Total		80	man days

3.5 Hints

- A realistic and useful plan, defines realistic feasible phases of the improvement process. These phases should cover a maximum time period of 6 - 12 months.
- Get other departments (development, users) involved e.g. by introducing a review committee.
- Keep IT-management informed of the progress.
- Communicate the plan to all levels (strategic, tactical and operational).

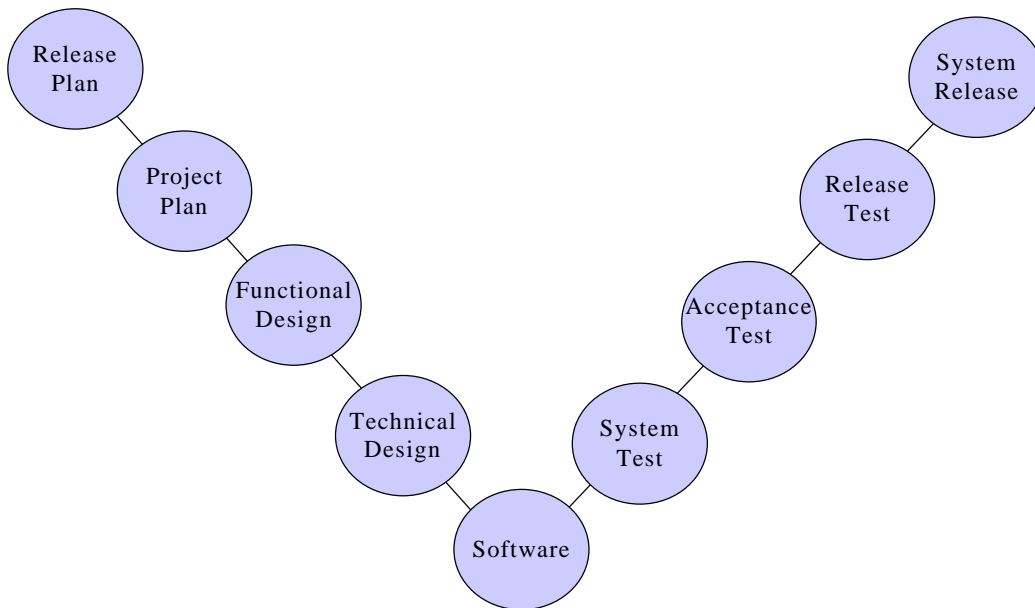
4 Execution of the first phase

Improvement actions are executed according to plan. In this paper only a selection of the improvement actions will be discussed. First we will focus on the major preliminary actions, and subsequently on some of the standard test improvement actions and on some actions that proved to be very difficult to realise in practice.

4.1 Describing the workflow

In this organisation many of the problems with testing were caused by the lack of control in the development process: releases were not clearly defined, release plans were missing and developers tended to start working on software without making any specifications.

In this case it is necessary to start with to define the workflow of structured development and testing. The heart of this workflow is represented by the following figure:



Project managers participated in setting up this document. Each step was described, by mentioning the departments involved, and the conditions for starting each step. After reviewing this document, the IT-management approve this product, and called a work meeting where this workflow was explained to all project managers, developers and testers. This did not change the world overnight. But it was a first and very important step in laying a foundation for a structured testing process. The workflow described the following elementary conditions for a structured test process:

- Producing a release plan, by a release-committee that would cover a period of 6-12 months.
- Producing functional and technical designs: required for both structured development and structured testing. A workgroup of experienced developers was commissioned to start working on guidelines for these designs, and a few projects were selected as pilot projects for producing these designs.

4.2 Improvement actions for black box testing

Master test plan

In the mean time the test team started to work on the master test plan. The heart of the test plan is the test strategy depicting the aspects to be tested and the relevant importance of these aspects.

In this organisation we defined the following major test strategy:

Quality aspect	Focus of the release test
Functionality	<ol style="list-style-type: none"> Does the application work together with the chain of applications at the banks. This is tested by using a few regular cases. There is a distinction between correct and incorrect input. The correct input is mostly derived from the regular cases used in the integration test. The incorrect input is identified based on the specified validity controls in the application chain of the banks, and based on experience and knowledge of the testing team. If possible the tests will be performed on the production chain. Are the correct files, directories and databases created after set up, and after upgrade.
Efficiency	<ol style="list-style-type: none"> Is the response time for handling large amounts of data acceptable. If applicable the efficiency test will be performed for the three recommended platforms Win 3.x, Win 95 and Win NT, using configurations with 80486 and pentium processors.
Platform compliancy	<ol style="list-style-type: none"> Is set up possible for the three platforms Win 3.x, Win 95 and Win NT.
Compatibility	<ol style="list-style-type: none"> Is upgrade by CD from version 1.5, 2.0 and 2.1 possible for the three platforms Win 3.x, Win95 and Win NT. Is automatic upgrade by Upgrade Server from version 2.0 and 2.1 possible for the three platforms Win 3.x, Win95 and Win NT.
MultiLanguage	<ol style="list-style-type: none"> Are messages, field names and help information displayed in the correct language. This is tested for the selected screens only. It is tested for a number of randomly forced messages. Note that the correctness of the contents of the messages and help files is NOT tested.

In the test plans the test strategy is described in more detail. For a given release the application is divided in test units and for each unit the tests are described, as can be seen in the next example

The following tests will be performed for all test units:

- Check if the expected Multiselect options are available ;
- Test whether the Multiselect options can be used for registration, modification, deleting, printing, sending, retrieving or any other option as logically expected;
- Check if all lay-out adaptations of main screen are available;
- Check if lay-out of main screen is as to be expected for a Win95 application;
- Check every functions of the module: does activating the function lead to the expected screens;

For test units of high importance the following tests will be performed:

- Test whether the main functions of the module operate correctly by using regular data only;

If there is still time available some general tests of the application will be performed using the error guessing technique. The focus will be on memory and file release.

Introducing testing techniques

In this organisation the following testing approach was used: After the developer had created the software and had convinced himself that his program worked well, he would hand over the application to a tester. The tester would install a copy of the program in the test environment, would try to run the application, and would start testing by trial and error. Tests were not specified prior to execution, tests were not documented and tests could not be reproduced. This approach is very common of many young organisations.

In such a situation it is important to get testers to derive test cases in a structured way, and to document test cases in a structured way. This is achieved by creating a work instruction for testers. This instruction must be based on an actual example of testing an application within this organisation.

Standard test documents.

During the pilot project the following templates for testing documents were conceived:

- test scripts;
- test plans;
- test reports.

Checklist of do's and don'ts

In this organisation we introduced the following checklist of do's and don'ts. The aim of this checklist was to adjust the attitude of the testers. The items in this checklist were collected during the pilot project, in which some of the testers started to derive test cases in a structured way for the first time:

1. *Never assume anything*

In the test specification phase your only reference for inferring test cases is the test base (i.e. the functional designs + logical data model + screen designs). If this test base contains any inconsistencies, imperfections errors or ambiguities the tester will report a problem.

2. *Traceability of test actions*

For each test in a test set, one must be able to determine why this particular test is present. What is the purpose of this specific test. This is achieved by documenting the test actions in a test script, and relating these test actions to logical test cases, which are derived from test objectives. Test objectives and logical test cases are documented in test specifications.

3. *Independence of test actions*

When specifying the test actions to be executed in a test script, be aware of the independence of the test actions. Maximum independency means that each action can be executed or (re-executed) independent of any other specified test-action. This gives the highest degree of flexibility during execution. On the other hand this approach generally leads to a high number of test-records in the initial test set. This implies a lot of effort needed to specify, build and maintain the initial test set. As a tester you have to assure yourself that you have created the maximum independency that is reasonably attainable.

4. *Creating test scripts*

The last step in completing the test specification is estimating the importance of the logical test cases. The importance is derived from the risk involved (i.e. probability of occurrence and estimated cost to repair). The importance is ranked in categories high (I0), medium (I1) and low (I2). For each category a separate test script is produced. Within a script the test cases are arranged in order of execution.

5. *Order of test scripts*

When creating the test scenario the test scripts are arranged in order of importance (as much as possible).

6. *Pre-test*

After the required test scripts are created, specify the pre-test. A pre-test describes the test actions that are to be performed prior to the execution of the structured tests. The purpose of the pre-test is to ensure the minimal

4.3 *Problematic improvement actions*

During the first phase some improvement actions appeared to be irrelevant or even impossible. Examples are:

- **Creating a test environment:**
A separate testing environment appeared to be unnecessary: The first test of the application could be performed on the testers' stand alone work stations. It was decided that couplings with bank applications were to be tested in co-operation with the IT-department of banks, where test environments were already sufficient. The actual coupling with banks systems were to be tested using test accounts. This ensured the use of production systems within the banks. The probability that this approach would jeopardise the production systems at the banks were minimal.
- **Introducing statement coverage analysis.**
There appeared to be no tool available for the development platform that was used by this organisation.
- **Introducing white-box testing techniques.**
It appeared that the developers did not have any time in participating in selecting structured testing techniques and drawing up testing standards. In this situation it is not useful to define a technique and force the developers to use it. In order to tackle this problem the objective had to be adjusted. Instead of a structured testing technique we provided a simple checklist with elementary items the developers had to use during module testing. Developers were asked to start working according to this checklist, and improve it during usage. The aim of this checklist was creating test awareness, rather than introducing structured testing for developers.

4.5 *Cope with resistance*

In this phase the resistance to change was recognised. Some people were reluctant to participate. They argued that the first actions mainly addressed a conceptual level and were of little practical use to them. At a later stage some of the same people argued that they recognised the importance of the program, but that the current release required all of their attention. In order to cope with this attitude the order of some of the activities was changed, and some quick wins were realised earlier than planned.

4.6 *Hints*

- The projects for probing the improved test approach must be good examples and must be carefully selected.
- During a change program it is impossible to satisfy all persons concerned. But it is important to remember that this is a sign that there really is a change going on. Get the influential majority of persons in the organisation tuned to accepting the changes.
- Keep checking the role of all players (are change angels still supportive, are blocking factors still negative).
- Use the supportive change angels and management commitment to overcome the resistance.
- Look for the quick wins (e.g. a reference card with areas of attention for testing by developers, a tool for tracking problems).
- Keep communicating the plans and intermediate results of the program of change. Keep IT-management informed of the progress of the program of change.
- Use the review committee to discuss the bottlenecks and find solutions TOGETHER.
- The change manager must be firm (stick to the plan) and, at the same time, must be flexible (change order of actions, drop actions that appear not to be feasible).

5 Evaluation

5.1 Results achieved

The first phase of improvement actions was concluded with a formal evaluation. All those involved were asked what has been achieved. The achieved results were compared with the original plan.

The following results were experienced by the testers:

- **Efficiency:** By applying the test specification techniques the number of required test cases can be limited, while achieving the appropriate depth of testing. The execution of tests requires less time than expected, due to the thorough preparation of the tests.
- **Flexibility:** By assigning a relative importance to test scripts, the test team is able to select between tests to be executed and tests to be dropped, if the amount of testing time is diminished.
- **Co-operation with developers:** By documenting test cases in an early stage of the development process, it is possible to discuss the test cases with the Development Team. This feedback improves the quality of the tests.
- **Reusability:** Test cases and test sets are documented in a standard way. This makes them reusable for future releases.
- **Effectiveness:** Important problems are encountered earlier in the testing cycle. By applying the test strategy and the specification techniques the added value of the functional tests is increased: these tests are supplementary to the tests performed by the development team.
- **Control:** The structured testing approach provides the test team with more insight in the completeness of the tests. The test specification techniques aid in understanding the functional designs and translating the specifications into test cases. The checklist for reviewing the test base aids in determining the testability of the functional designs in an early stage.

5.2 To be resolved

The evaluation revealed the following items that were still to be resolved:

Organisation

- The workflow for both developing and testing the application is properly described, but in practice not yet fully implemented. Support from IT-management is required to achieve this;
- A Product Manager must be made responsible for release plans and the delivery of software to banks;
- The agreements and decisions concerning encountered problems must be met in a structured way involving test team, development team and the Product Manager.
- A quality review team must be involved in an earlier stage of the functional designs;
- Applying the new approach is not yet a common habit of all employees of the test team. Coaching by an external testing consultant is still required.
- Deliveries of DLL's, builds and set ups to the test team must be accompanied by unambiguous, standard description of the delivery;

6 Continuation

6.1 Adjustment of the plan

It was decided to split the subsequent improvement phase into two separate phases. This was based on the recognition that the pace of the change activities must be in accordance with the available resource capacity of the organisation.

1. Phase 2: Preserving achievements phase 1;
In this phase the speed of introducing improvement steps is reduced and the focus is on preserving the achieved results;
2. Phase 3: Next term improvement actions:
In this phase a second group of improvement actions will be carried out;

The activities and planning of the second phase is described in more detail by the following table:

Period: 1 June 1998 - 30 September 1998

Nr	Action	Result	Effort
2.1	<i>Coaching and support in structured testing</i>	Structured testing approach preserved	7
2.2	<i>Implementation of the decision forums for integration test and release test'</i>	Operational decision forum for accepting test plans and test reports	2
2.3	<i>Realisation of minimum work flow tests</i>	Pre-test for quickly assessing the correctness and stability of the builds	5
2.4	<i>Implementing defect tracking tool</i>	Tool in use for centrally registering and tracking defects.	5
2.5	<i>Presentation workflow and structured testing to development team</i>	Synchronisation of development team and test team	2
2.6	<i>Presentation workflow and structured testing to other banks</i>	Awareness of the testing approach of TT at all banks	4
2.7	<i>Drawing up the plan for optimising the use of WinRunner</i>	Plan for realisation of the automated testsuite	7
2.8	<i>Presentation of the plan for optimising the use of WinRunner to IT-management</i>	Commitment and approval of the plan for realisation of the testsuite	1
	Total phase 2		33 mandays

6.2 Hints

- If required, take a short break, and focus first on consolidation of the results achieved thus far.
- Provide a coach to help more people to get acquainted with the new working methods.
- Find a way of communicating the results to more people in the organisation.
- Do not go on full speed with the next heap of proposed improvement actions, when you recognise too much weariness within the organisation.

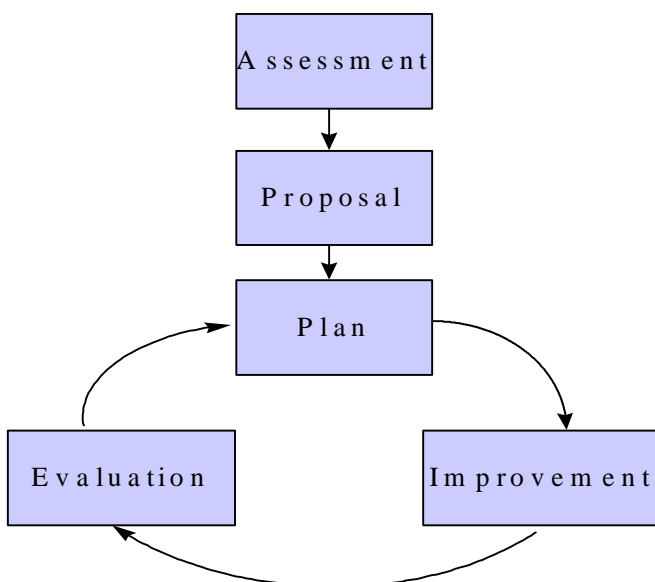
7 Conclusions

The first phase of introducing structured testing into this organisation was successful. Already the organisation experienced the results of the first improvement actions. Commitment and involvement was obtained by defining and implementing the improvement actions and all phases together with people involved in the change process (managers and employees). Communicating the stage and the progress of the improvement program to all levels, both formal and informal, was recognised to be of paramount importance. At present the organisation is going through the second phase of introducing structured testing.

7.1 The phases of introducing structured testing

Introducing structured testing in an organisation constitutes a change program. It means adaptations of existing working procedures, introducing new working procedures, introducing new standards for documentation and reporting. It must be realised that testing is not a solitary activity of a confined group of employees. Inevitably other disciplines are also involved: customers or sponsors have to approve test plans, sponsors have to decide on software release based on test reports, program managers have to get a grip of the development process by planning releases, designers have to produce design specifications that are fit for deriving test cases in a structured way, developers have to be able to provide insight into the scope and results of their tests. A successful program of change requires an approach that is based on discrete phases. Based on the account, described in this paper, the following phases are suggested:

1. Assessment of the current situation;
2. Proposal for a change program;
3. Plan for the change program;
4. Execution of improvement actions;
5. Evaluation and planning next improvement actions



7.2 Major lessons learned

The major lessons learned during this program are:

- **Communication is a separate area of attention in a test improvement process;**

A factor that is often overlooked or not given enough attention is the aspect area communication. We have to realise that in the beginning only a limited amount of people are directly involved in the process of change. The majority will still follow the current practice of development and testing. It is important to have everybody experience the existence of the improvement program. IT-management and project-management are informed via formal progress reports. It is important that others experience small benefits from the program of change as quickly as possible.

The best support for continuing a program of change will be the fact that someone, not directly involved, tells management how he/she has already benefited from the improvements.

- **Keep goals feasible and attainable;**
Divide the improvement process in phases of maximum 6 - 12 months. Only make a detailed planning of the first subsequent phase. Make sure that all improvement areas are dealt with in a phase. Give only a global planning of the next phases. Each phase is concluded by an evaluation and a detailed planning of the next phase.
- **Look for quick wins;**
An example of a quick win is a defect tracking procedure. A standard defect registration sheet is easily defined and easily explained. A simple tracking tool can be built e.g. using MS-ACCESS or MS-EXCEL. This result is quickly produced, and will very quickly be embraced by many people. The pitfall is to want to build and implement the ultimate tool. Build a basic tool. Give it a surplus by building simple reports. Number of defects registered in a given period, for the four severity-classes. Report these figures in graphical way.
- **Introducing elaborate structured testing techniques for developers will not work;**
They either lack the time or the discipline to keep it up. However a very useful and simple aid is setting up a checklist containing:
 - the standard aspects to be checked by a program before delivery of the software;
 - a top ten list of the most common programming errors;Present the list to all developers in a meeting in the form of a reference-card, a reminder-poster or any form that will enhance its usability. Do not specify the obvious elements (these need to be documented in a programmers training course), but specify the elements that are often overlooked, and lead to errors in later test phases.
- **Endurance is required ;**
Do not demand that the improvement effort will pay off immediately. Creating awareness, and laying the foundation takes time. But be aware of every success, and do not forget to communicate it. Be supportive to the development team. Collect metrics as a prove of the effectiveness of the change program.

ESI

**SOFTWARE CMM LEVEL 2:
THE HIDDEN STRUCTURE**

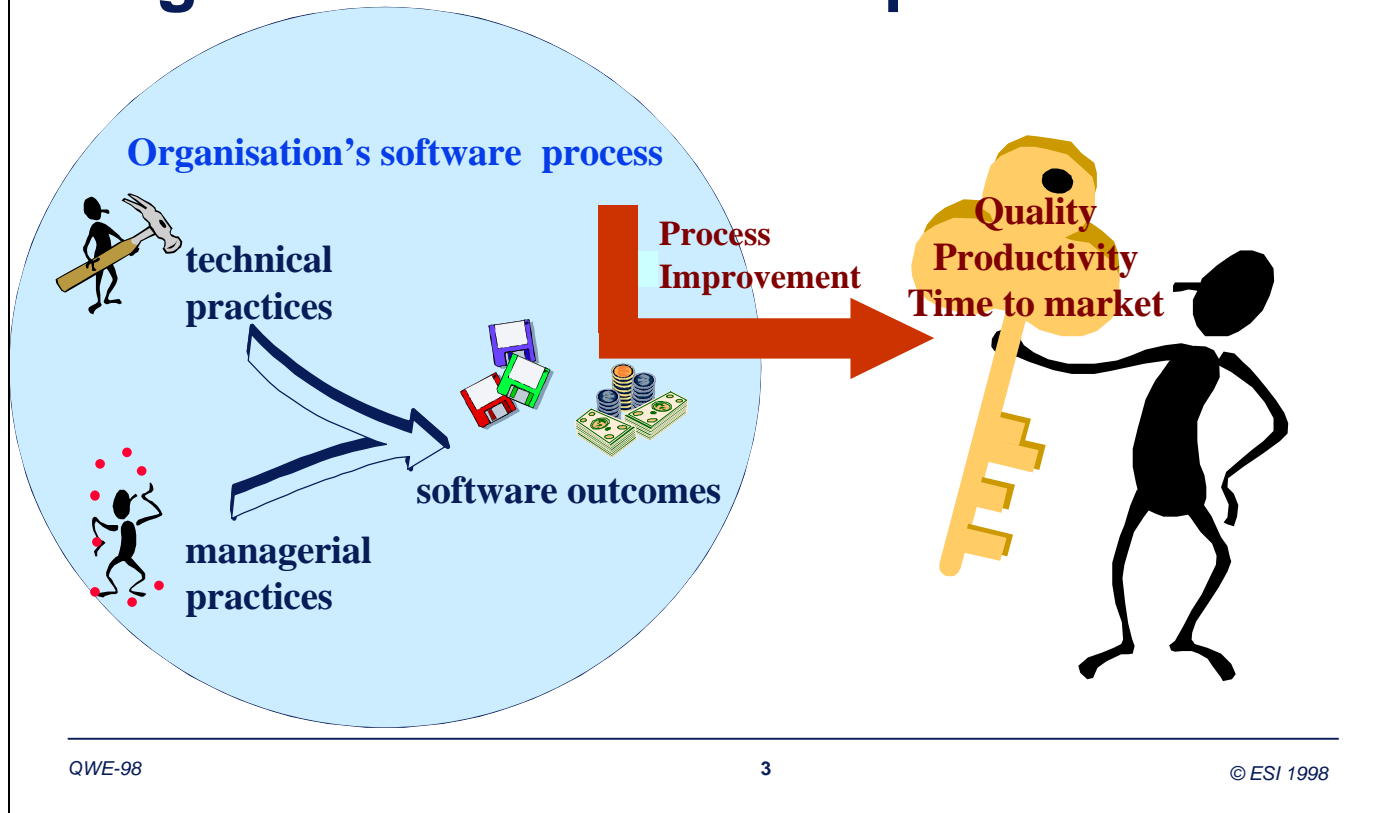
ESI

Objectives of the paper

- Present the **Self Standing Tool** for process improvement developed by BIG-CMM project.
- Explain the SW-CMM framework for **Small and Medium Enterprises**.
- Describe the architectural elements of the **BIG-CMM improvement guide**.
- Derivation mechanism of the **improvement plan**

ESI

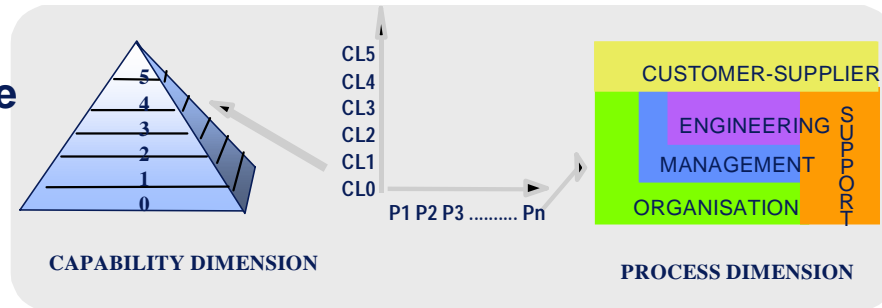
Organisation's software process



ESI

SPI: ISO 15504 SPICE

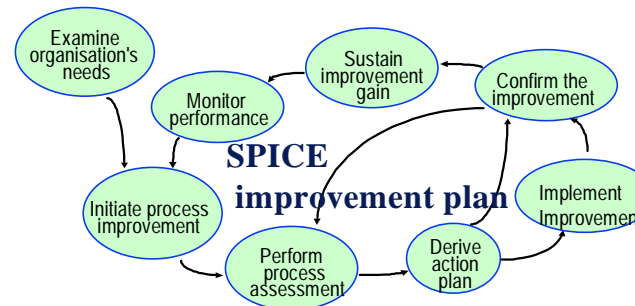
- Method for assessing the maturity of an organisation



- Flexibility



- Guide for continuous process improvement



ESI

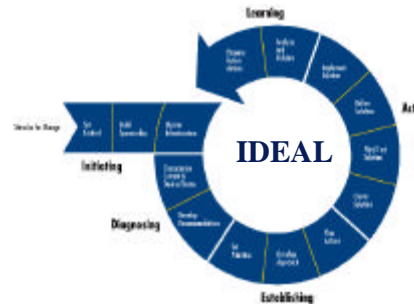
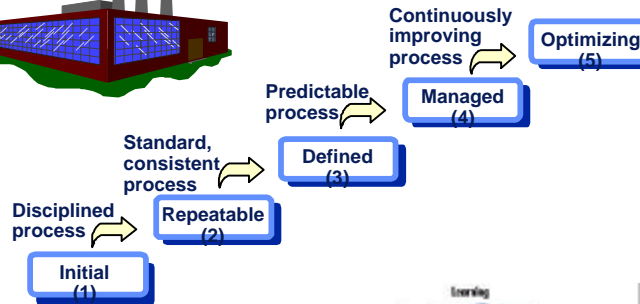
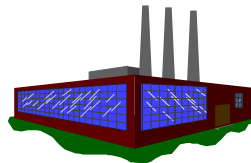
SPI: Software Capability Maturity Model

• Large Enterprises

• Method for assessing the maturity of an organisation

• Help an organisation prioritise its improvement efforts

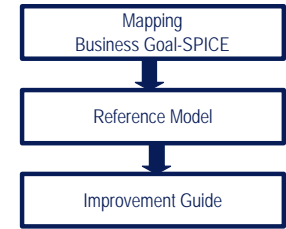
• ? SMEs?



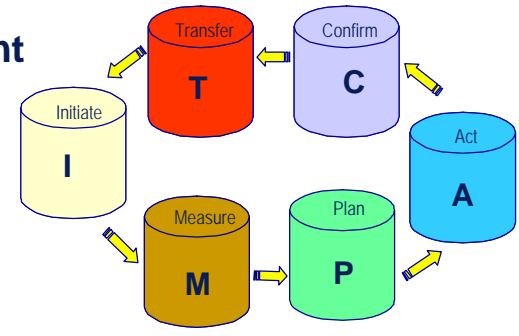
ESI

SPI: ESI's BIG guides

- Derive an **improvement plan** to achieve an explicit **Business Goal**.



- Continuous Process Improvement



- SMEs



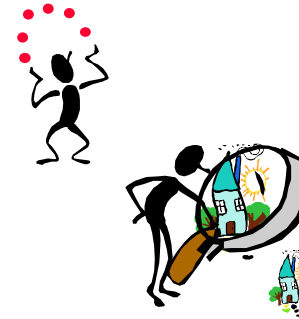
ESI

BIG-CMM



Level 2 of SW-CMM (hidden structure)

- establish effective management practices
- increase management visibility
- success repeatability



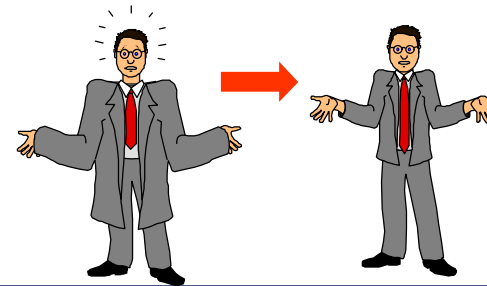
SMEs



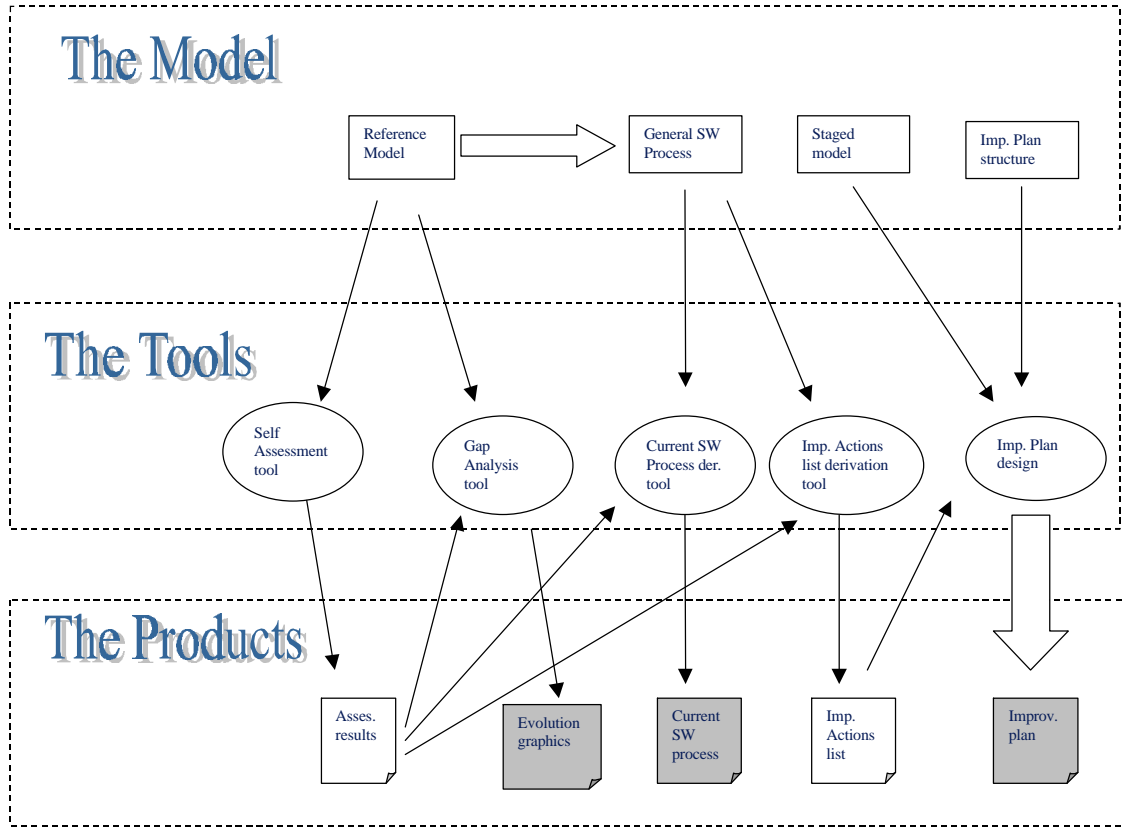
Self - standing



Tailoring the optimal software process

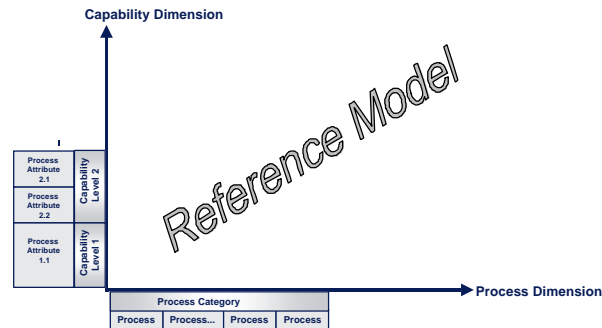


ESI ARCHITECTURAL DESCRIPTION OF THE GUIDE



The BIG-CMM reference model

The ISO-15504 (SPICE) like model that represents CMM Level 2 features that are thought relevant for SMEs. Elements that not add significant value for SMEs have been expurgated.



ESI

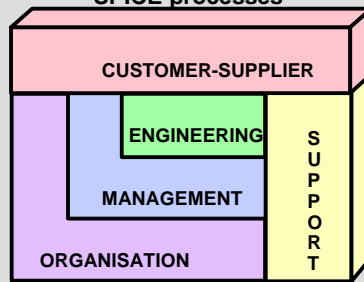
Process Dimension

- **MAPPING** between SPICE2.0 and SW-CMM v.1

SW-CMM L2 Repeatable

- Software configuration management
- Software quality assurance
- Software subcontract management
- Software project tracking oversight
- Software project planning
- Requirements management

SPICE processes

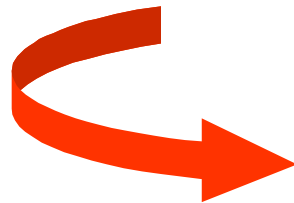


Capability Dimension

Reference Model

Process Dimension

CRM	PPM	PRM	SQA	SCM	SSM
-----	-----	-----	-----	-----	-----



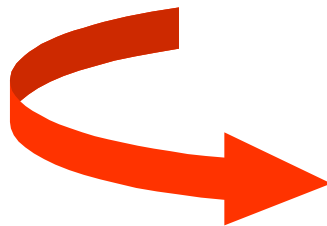
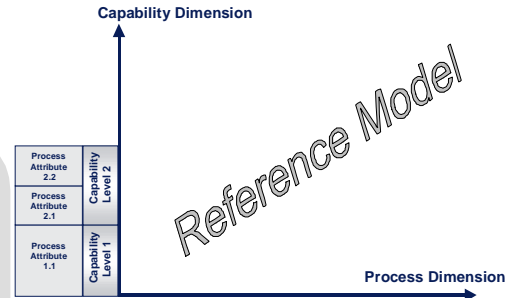
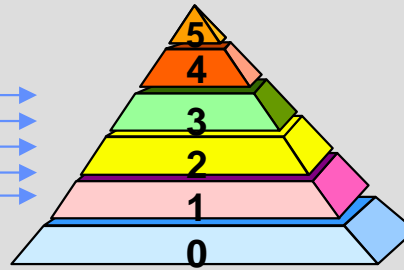
- Certain processes were enhanced to treat aspects considered **keys for success** of development.
- The model was reviewed against the new **SPICE 98** version.

Capability Dimension

- **MAPPING** between SPICE2.0 and SW-CMM v.1

SW-CMM L2 Repeatable

Software configuration management
 Software quality assurance
 Software subcontract management
 Software project tracking oversight
 Software project planning
 Requirements management

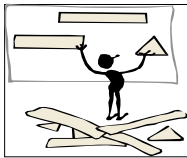


• Features offered by CMM at **institutional level** were added to the model at the capability dimension. The second capability level incorporated things like **policies, training,** and so on.

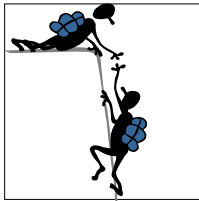
BIG CMM Processes



CRM Customer Requirements Management



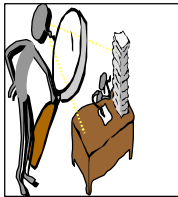
PPM Project Planning Management



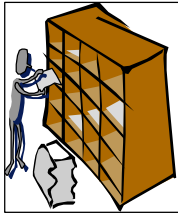
PRM Project Risk Management

ESI

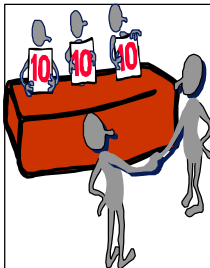
BIG CMM Processes



SQA Software Quality Assurance



SCM Software Configuration Management



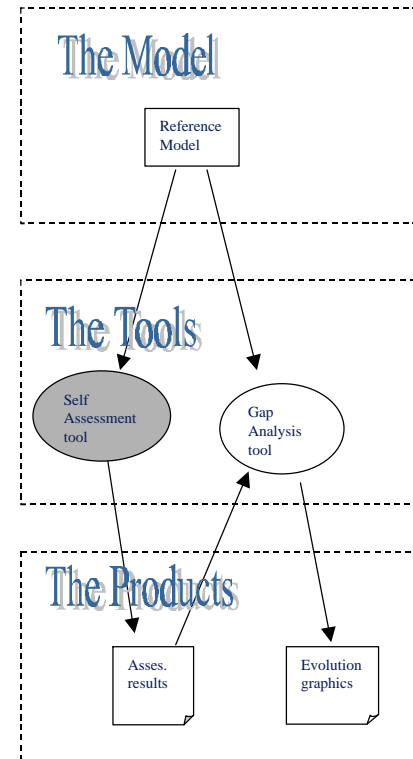
SSM Software Subcontract Management

ESI

Self assessment tool

Self standing tool based on questionnaires to implement the improvement action list.

Knowing what you want to find out, it is possible by formulating the proper question.



ESI

Self assessment tool

Weakness and strength questionnaires

Interviewers give their opinion on:

- what is enough,
- what can be improved and
- what is not relevant.



Detailed questionnaires

They cover all the aspects of the BIG-CMM model.

Categories of respondents :

- Managers
- Project leaders
- Engineers



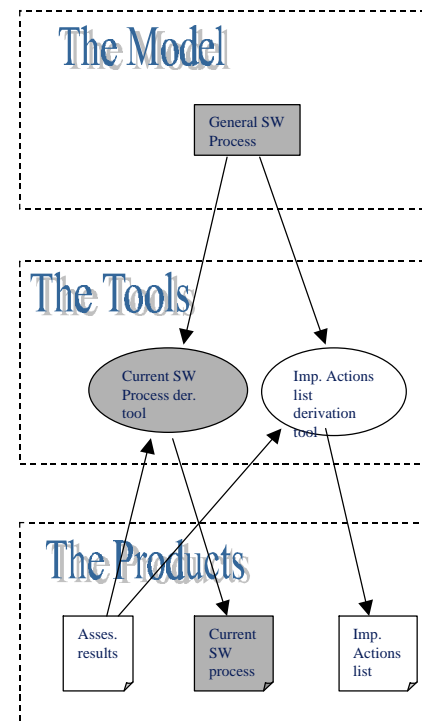
ESI

Current software process derivation tool

The *General Software Process (GSP)* is a group of ideal management actions that should be performed in an ideal company

The *Company's Current Software Process (CSP)* are the good practices out of the GSP that have been identified in a company as result of an assessment.

The *current software process derivation tool* is based on the answers to opinion and detailed questionnaires.

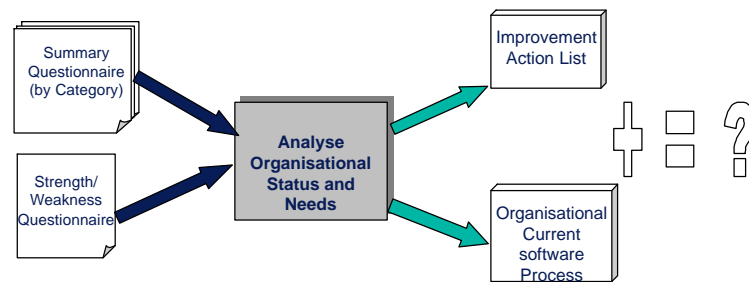


ESI

Improvement actions list derivation tool

As result of the questionnaire analysis are determined:

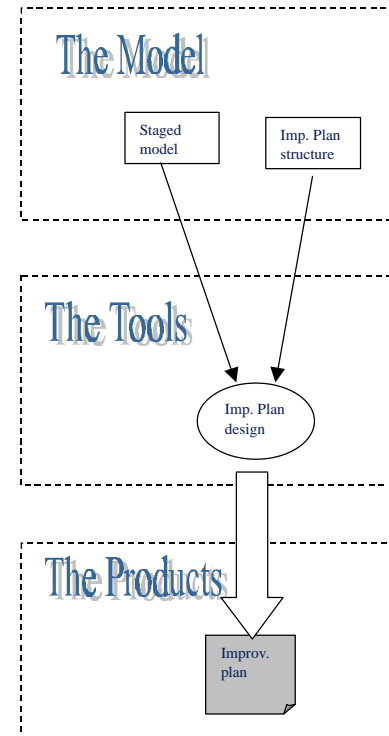
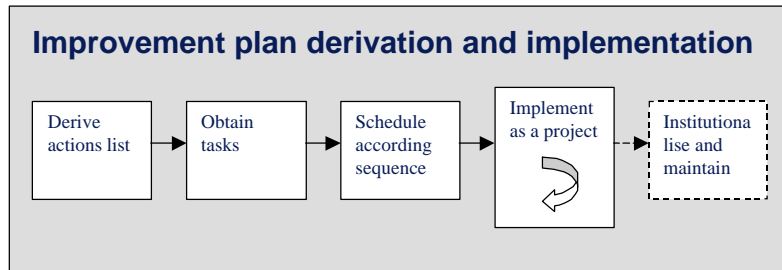
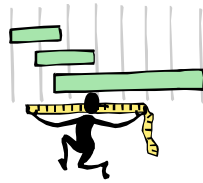
- The improvement actions
- The activities that form part of the good practice asset of the company.



ESI

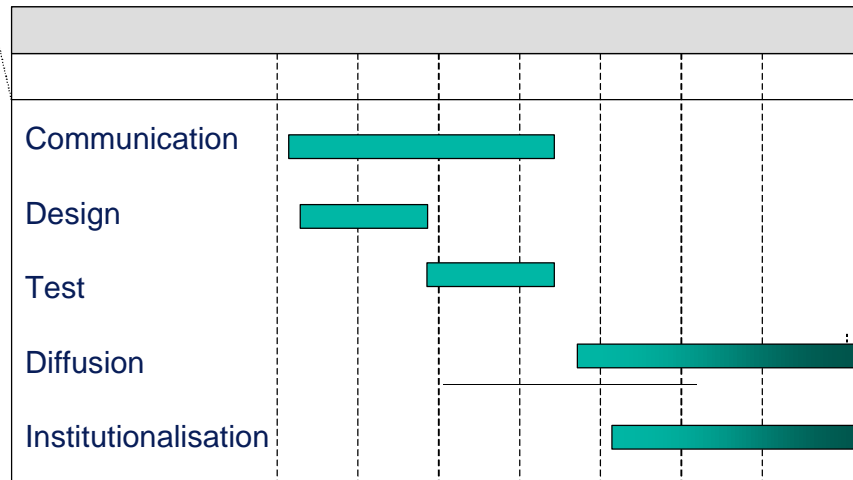
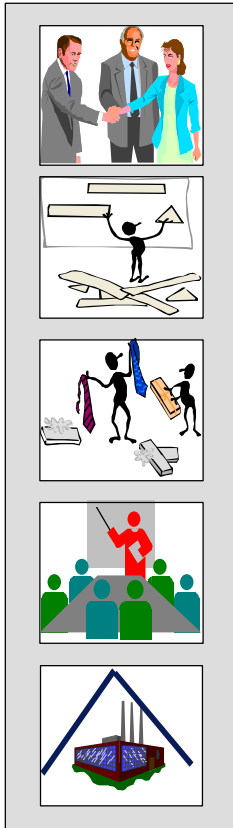
The improvement plan

The improvement should be treated as a project itself. Responsibilities are defined, resources are allocated, tasks specified and schedule is calculated.



ESI

The improvement plan



ESI

Summary

- **BIG-CMM is an aim for SMEs**
- **Help achieve the repeatability of success**
- **Must be validated**
- **Self - standing approach**
- **Questionnaires based**

SW CMM Level2: The hidden structure

*M.Elisa Gallo (melisa@esi.es), Pablo Ferrer (pablo@esi.es),
Mikel Vergara (mikel@esi.es), Chema Sanz (sanz@esi.es)*

European Software Institute

Parque Tecnológico Edificio 204

E48170 Zamudio Bizkaia SPAIN

Phone: ++34 94 420 95 19 Fax: ++34 94 420 94 20

This article explains the work performed in constructing a tool that helps small and medium sized enterprises (SME) to achieve SW CMM level 2's objective: repeatability of success in software development. It interprets SW CMM framework for SMEs and explains principles and architecture of the self standing tool: BIG-CMM. This paper is written for people involved in software process improvement (SPI) in organisations with no big budget and infrastructure. It proposes a practical improvement path for these entities with the idea that information hiding is a valid approach for improving in SMEs world.

Keywords: Software Process Improvement, CMM

Introduction

This section introduces the concepts needed to fully understand the work presented in the paper.

The set of technical and managerial practices used by an organisation to produce software make up the *organisation's software process*. The fact is that, organised or chaotic, this process always exists inside the organisation. However, the key for successful initiatives is to establish some discipline that permits co-operative work.

The need to work as a team, the survival of the products and competitiveness are just some of the reasons that lead companies to improve their software process hoping that efficiency and quality will flood through their structures and products. With this idea in mind, many official, non-official, altruistic or profit-driven organisations have investigated the way of conducting what is known as *software process improvement (SPI)*.

SW-CMM is a framework for this type of improvement initiative. It is mainly designed for large companies with significant budgets for improvement. In fact, *SW-CMM* was initially a tool for American Department of Defence (DoD) to assure that its subcontractors were working in benefit of its interests. *SW-CMM* constitutes an elaborated model that represents capability features together with general activities. The different stages form steps to defined maturity levels up to a state of continuous improvement.

ISO 15504 (SPICE) has a different objective to SW-CMM. It is an improvement framework with two interesting features: its process orientation in addition to the capability level and the use of base practices. SPICE makes it easier to focus on individual management issues and yields practical elements that can be translated into concrete improvement actions.

The European Software Institute (ESI) is a recognised and independent authority on SPI. Our particular focus is on helping businesses to achieve real commercial goals such as reduced costs and increased product quality. As part of this work, we have developed a set of tools to support companies in meeting specific business improvement objectives: the *Business Improvement Guides (BIGs)*. These guides cover different goals and different types of company, but their common objective is to help an organisation evaluate its status and develop and implement an improvement plan.

Several representatives of this BIG series can be found:

- BIG-ISO9000, focused on ISO9000 certification achievement.
- BIG-TTM, focused on reducing time to market
- BIG-CMM, which helps a company to achieve repeatability of success in projects.

All of them have a common ground: they use SPICE characteristics – process orientation, detail level, modularity – to construct an improvement model that fulfils specific objectives.

BIG-CMM

SW-CMM establishes five maturity levels that represent the historical phases, which an organisation usually goes through in software process improvement. The second level, also named *Repeatable Level*, represents a state where organisation has achieved a stable and disciplined software process based on project control. There is no need to have the same process for each of the projects, but policies and documented procedures help to establish effective management practices, which ultimately, increase management visibility and allow successful results to be repeated.

BIG-CMM is a tool that helps SMEs involved in software development to reach software CMM Level 2's objective: repeatability of success by institutionalisation of effective management practices. By SME we refer in this instance to those companies which usually have no sizeable budget to invest in initiatives that are not directly profitable and with comparatively small organisational infrastructure. An additional characteristic of SMEs is the size of the projects, which are small in terms of people and time.

To improve one organisation's software process means to evaluate company's status, establish a list of improvement actions, produce the necessary plan to implement them and carry this plan out. Doing this through the SW-CMM, requires considerable experience, knowledge and time, even when getting a certification is not the ultimate goal. For the SME, the solution can be based on information hiding: main steps of the improvement process are present, but the underlying structure is hidden.

Finally, as with projects, which may vary in complexity, size and duration, even similar organisations will have different requirements regarding the desired level of improvement. BIG-CMM permits an organisation to tailor the optimal software process and adapts it to their needs, thus stepping off the improvement process when it has met their requirements.

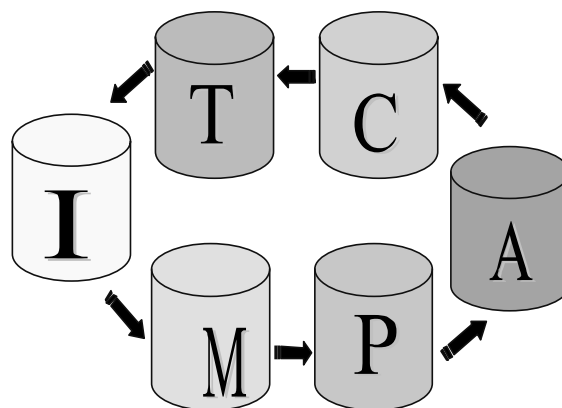
ESI's improvement cycle: IMPACT

Based on the idea of continuous improvement, IMPACT covers all the major improvement phases, since the need arises until the solution is assumed by the organisation's culture. IMPACT stands for Initiate, Measure, Plan, Act, Confirm and Transfer:

- Initiate process improvement. Raise organisation's awareness of the need for improvement identifying requirements and goals
- Measure the current situation. Establish the organisation's position versus an evaluation model
- Plan the improvement. Select strategies, obtain commitments and resources, identify tasks
- Act. Perform the necessary actions for the improvement.
- Confirm and sustain the improvement. Measure or assess the specific processes.
- Transfer successful technology to the rest of organisation.

This is a continuous loop that may be followed in a cyclical manner:

Figure 1 - IMPACT improvement cycle



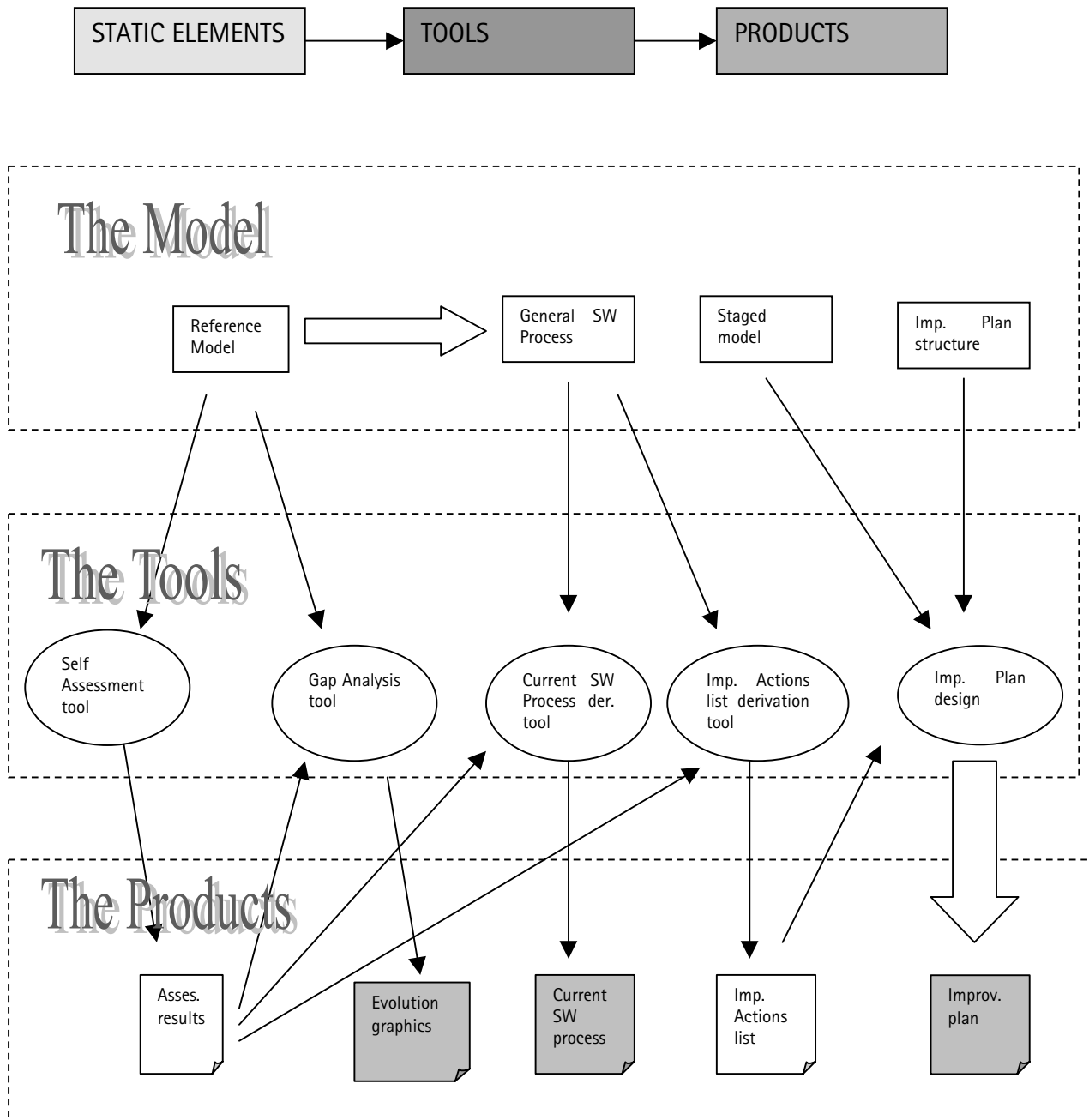
IMPACT forms the framework in which all ESI products and services are developed.

BIG-CMM addresses the Measure, Plan and Act steps of the cycle and also giving guidance on the Confirm and Transfer steps.

Architectural description of BIG-CMM

The guide can be seen as a set of static elements or model, and a set of procedures or mechanisms that use the static parts to yield a group of products used for process improvement.

Figure 1 - Architecture of BIG-CMM



Where:

Model components

Reference model Also called Development model, is the SPICE like model that represents SW-CMM features that are thought relevant for SMEs.

General SW process A set of actions deducted from the Reference Model and ordered by a project sequence criterion. By accomplishing all these actions, a company would be performing the ideal set of actions associated to repeatability.

Staged Model Establishes the sequence of steps in which improvement tasks must be executed

Improvement Plan structure Responsibilities, activities and procedure necessary to implement the improvement actions list

Tools components

Self assessment tool Self standing tool based on questionnaires to evaluate company's status against the reference model.

Gap Analysis tool Provides graphic representation of the assessment results

Current Software Process derivation tool Uses assessment results and General Software Process description to derive the company's Current Software Process

Improvement Actions list derivation tool Uses assessment results and General Software Process description to derive the proposed Improvement Actions' list

Improvement Plan Design Procedure Uses the Improvement Actions' list and the Improvement Plan structure to organise the Improvement Plan

Products

Current Software Process Represents the set of good practices currently performed in the company related with their development management tasks.

Improvement Actions' List List made up of improvement actions and orientations whose implementation should lead to repeatability according to the proposed model.

Improvement Plan Mechanism to implement the improvement action's list.

Evolution graphics Show the current and desired status in a bars diagram.

The reference model

Following the BIG-series, BIG-CMM achieves its objective through the ISO15504 standard (SPICE). For this reason, those elements of the SW-CMM Level2 that add complexity but no significant value to SMEs are removed, trying to keep the objective of each key process area (KPA) and the general one. The adapted model is mapped to a set of SPICE processes that cover all its features.

The mapping between SPICE 2.0 and SW-CMM v.1. resulting from the ESI's UNIFRAME project, was used to determine which SPICE processes fulfilled software CMM characteristics at Level 2. This set of processes was used to produce a version of SW-CMM easily applicable by SMEs. In addition, certain processes were enhanced to cover aspects that were considered key to the success of development. The final refinement came from reviewing and updating the model against the new SPICE 98 version.

The following table shows how the official SW-CMM's KPAs are processed and covered by the SPICE-like processes in the BIG-CMM reference model:

KPA	BIG CMM Process	Changes
RM (Requirements management)	CRM (Customer requirements management)	The process is seen from a managerial point of view instead of the restricted SW-CMM's vision of the Software Engineering Group (SEG). Management of the gathering, understanding and agreeing of customer requirements is added
SPP & SPTO (Software Project Planning & Software Project tracking and oversight)	PPM (Project planning management) PRM (Project risk management)	Project planning and Project tracking and oversight are joined in a unique process. Risk management is considered to be at an upper maturity level and is served by an independent process
SQA (Software quality assurance)	SQA (Software quality assurance)	No change
SCM (Software configuration management)	SCM (Software configuration management)	No change
SSM (Software subcontract management)	SSM (Software subcontract management)	No change

SW-CMM is a one dimension model that gathers activities, managerial, quality and organisational aspects for each process area. BIG-CMM has two dimensions:

1. Process dimension, for definition of the activities and their associated work products
2. Capability dimension for managerial, quality and organisational aspects.

The above table shows the mapping of processes to activities. The rest of common features of SW-CMM are mapped to the capability dimension.

The result is a SPICE-like assessment model against which a specific software process might be evaluated. The next step is defining a capability degree of coverage for each resulting process so, an optimum SPICE profile can be calculated. Approaching this profile is obviously approaching SW-CMM level 2's objective.

General Software process (GSP) and company's current software process (CSP)

The software process is highly dependant on the company's needs and context. The first step in process improvement is determining the set of practices that form this software process, analysing it and selecting what is useful in it.

BIG-CMM analyses its model's characteristics extracting a set of subactivities which form the *General Software Process (GSP)*. This GSP is a group of ideal management actions that should be performed in an ideal company to achieve repeatability of success. Identification of a company's needs and current good practices from this list produces the company's *Current Software Process (CSP)*.

Self assessment tool

To help SME's to substitute expensive assessments, BIG-CMM includes a questionnaire system composed of two different sets of questionnaires:

- Weakness and strength / opinion questionnaires. Used to determine areas inside which good practices could be extracted from and areas that could be subject to improvement. The range of answers introduces the company's context in an indirect way: interviewees give their opinion on what is satisfactory, what can be improved or what is not relevant.
- Detailed questionnaires. They cover all aspects of the BIG-CMM model. They must be fully completed except for areas covering non applicable issues (e.g.: subcontracting)

Questionnaires are addressed to three different categories of organisational representatives: managers, project leaders and engineers. And questions are written to taken this into account.

The actual questionnaire is based on a systematic analysis of model characteristics. Briefly

1. Base practices are divided in activities and subactivities.
2. Capability aspects are instantiated for each process and translated into activities and subactivities

- For each subactivity, mainlines and implications are considered. Each one will produce a question that must be answered within a short range of values (yes, no, don't know)

The underlying idea is that knowing what you want to find out, in essence, it is possible by formulating the proper question in the proper style to get the right information. A benefit of this approach is the establishment of a unique relationship between answers to the questions and possible actions within the improvement plan. In addition, categorisation of questions in mainlines and implications allows mandatory and optional improvement actions to be established.

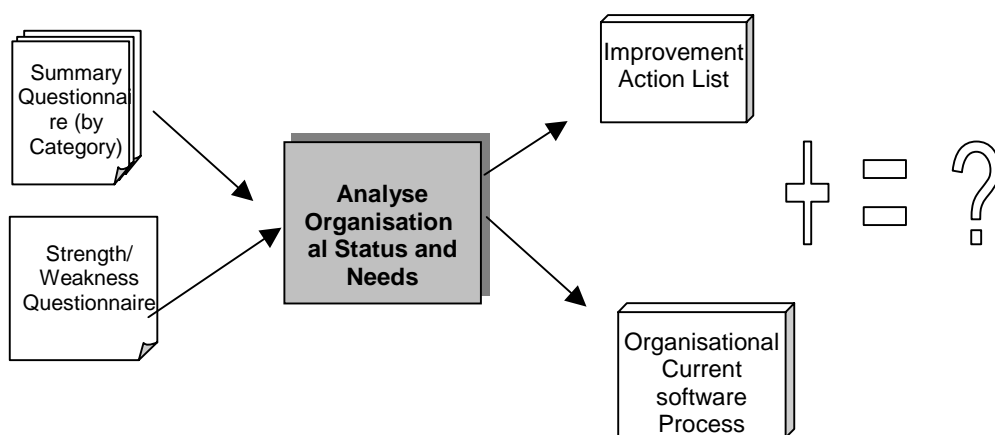
Current software process determination tool

The guide proposes a way to determine the adequate practices the company is performing when developing software to establish them as the initial point for the improvement action. The procedure takes answers to Strength and weakness and detailed questionnaires as a basis. If a company feels that certain issues of the proposed software process are properly performed, then it is very likely that these activities form part of the good practice asset of the company.

Improvement actions list derivation tool

The same principle than in current software determination tool is applied to establish the desired final software process for the organisation and the resulting improvement actions list. Answers to Strength and weakness questionnaires are compared with those from the detailed questionnaires. If the company feels that something is subject for improvement, then it can be added to the improvement actions list. This allows different levels of achievement to be established, depending on the company's needs.

Figure 3 - Current software process and Improvement actions list determination



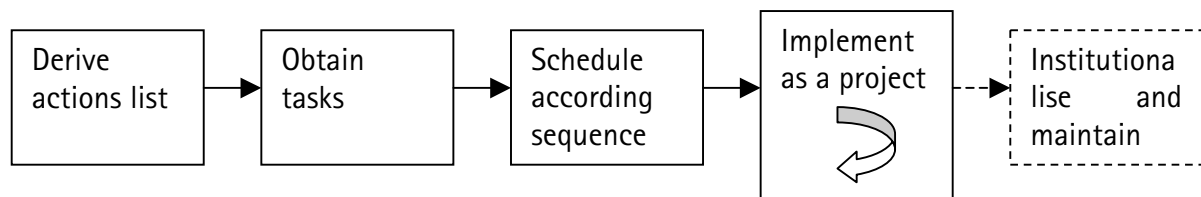
The improvement plan

The improvement plan as presented by BIG-CMM covers the Plan and Act phases of the IMPACT cycle. It builds on a generally accepted principle: the improvement should be treated as a project in itself. Thus, responsibilities are defined, resources are allocated, tasks are specified and a schedule is calculated.

BIG-CMM considers the improvement plan as the result of combining improvement actions with implementation mechanisms for these actions. This is done in a practical way:

1. By establishing a mechanism to derive improvement actions that lead to general improvement in the Company's Software process
2. By providing a criterion to gather single actions into action packages that can be treated as separate improvement tasks
3. By defining an implementation sequence (staged model) that leads the company through a natural improvement path
4. By identifying concrete activities for implementing action packages (improvement tasks), i.e. communication, design, test, diffusion and institutionalisation

Figure 4 - Improvement plan derivation and implementation



The last phase can be considered separately to the project plan implementation itself, but it is a similarly important action. Once the improvement is reached, it must be sustained by establishing control methods.

The implementation sequence proposed by BIG-CMM is outlined in the table:

Figure 5 - BIG-CMM's staged model

Phase	Contents
1	<ol style="list-style-type: none"> 1. Initial requirements for Software Quality Assurance and Software Configuration Management 2. Customer Requirements Management to performed level 3. Project Plan Management to performed level 4. Software Subcontract Management to performed level when necessary

2	1. Software Configuration Management to performed level
3	1. Software Quality Assurance to performed level
4	1. Project Risk Management to performed level
5	1. Customer Requirements Management to managed level 2. Project Plan Management to managed level 3. Software Subcontract Management to managed level when necessary
6	1. Software Configuration Management to managed level 2. Software Quality Assurance to managed level 3. Project Risk Management to management level

Action packages that make up the tasks for the improvement plan should be implemented in the order defined in this table.

Conclusions

BIG-CMM facilitates SME in achieving the objective of SW-CMM Level 2. The aim is to allow small and medium size enterprises with less resources and small infrastructure to obtain repeatability of success in software development. It should not be understood as a recipe to achieve any kind of certification.

The work explained in this article is still far from being adopted even by the early majority of the mentioned enterprises and must still be validated through trials, although some of its mechanisms such as the self assessment tool and the evolution graphics have already proved to be accurate.

The thesis of self-standing and information hiding is defended as a valid approach for improving. It is possible to achieve the desired information by sensibly asking for it, without expensive assessments that are inadequate for the needs of SMEs. It is possible to establish the elements necessary for improvement in the same way.

QWE '98

Year 2000 and the euro: Testing and Data Management using Data Commander™

11 November 1998

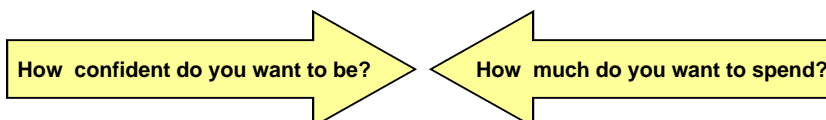
Charlie Crawford

Blackstone & Cullen, Inc.
2000 RiverEdge Pkwy Ste 750
Atlanta GA 30328-4600
www.datacommander.com

ccrawford@bac-atl.com
Phone: 770 612-1550
Fax: 770 612-1471
www.bac-atl.com

Year 2000 Compliance Testing

- 45-70% of compliance effort (Gartner Group)
- Essential element of risk management
 - Costs
 - Time
 - People
 - Equipment
 - Facilities
 - Implications
 - Effectiveness
 - Efficiency
 - Liability



Blackstone & Cullen, Inc.

www.bac-atl.com

Testing Considerations

- **System Components**
 - Hardware
 - Software
 - Programs - get most of the attention
 - Data - cause many of the problems
 - Output
 - Interfaces
- **Procedures**
 - Compliance Plan
 - Test Plan

Blackstone & Cullen, Inc.

www.bac-atl.com

Testing Characterizations

- **Levels**
 - Unit, Integration, System, Interoperability
- **Basic Types**
 - White Box v. Black Box
- **Techniques**
 - Static v. Dynamic
 - Manual v. Automated
 - Structural v. Functional
 - Destructive v. Non-destructive

Blackstone & Cullen, Inc.

www.bac-atl.com

Testing Decisions

- **Prioritization (“Triage”)**
- **Number and type of tests**
- **Definition of success**
- **Completion date**
- **Personnel**
 - In-house
 - Contractor / Consultant
- **Facilities & Equipment**
- **Tools**

Blackstone & Cullen, Inc.

www.bac-atl.com

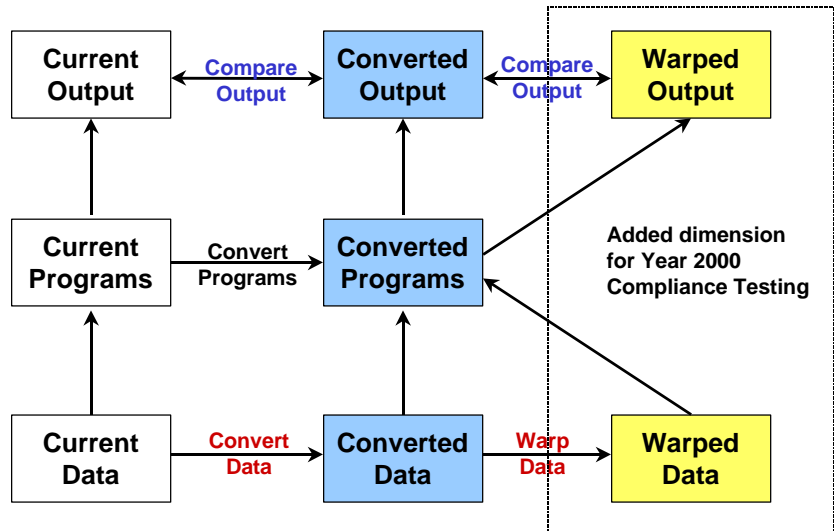
Commitment of People

- **Dedicate a few, but inform all**
- **Oversight Board**
- **Project Manager**
- **Systems Maintainers**
- **Users & Subject Matter Experts**

Blackstone & Cullen, Inc.

www.bac-atl.com

Uniqueness of Year 2000 Testing



Blackstone & Cullen, Inc.

www.bac-atl.com

Testing the Future

- **Important dates**
 - 31/12/99, 01/01/00, 29/02/00, etc.
- **“Warping” anomalies**
 - Leap years, month lengths, holidays, day of the week, 28 year phenomenon
- **Number of repetitions**
 - Associated costs

Blackstone & Cullen, Inc.

www.bac-atl.com

Year 2000 Compliance Truths

- **Takes longer and costs more than planned**
- **No one knows the whole system**
- **Will not heal a sick system**
- **A few programs cause most of the problems**
- **Reveals unanticipated information**
- **Not an independent event**
 - Hardware and software replacements and upgrades
 - Greater changes (e.g., merger, advent of euro)
 - Personnel turnover
- **Testing requirements vary with situation**

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

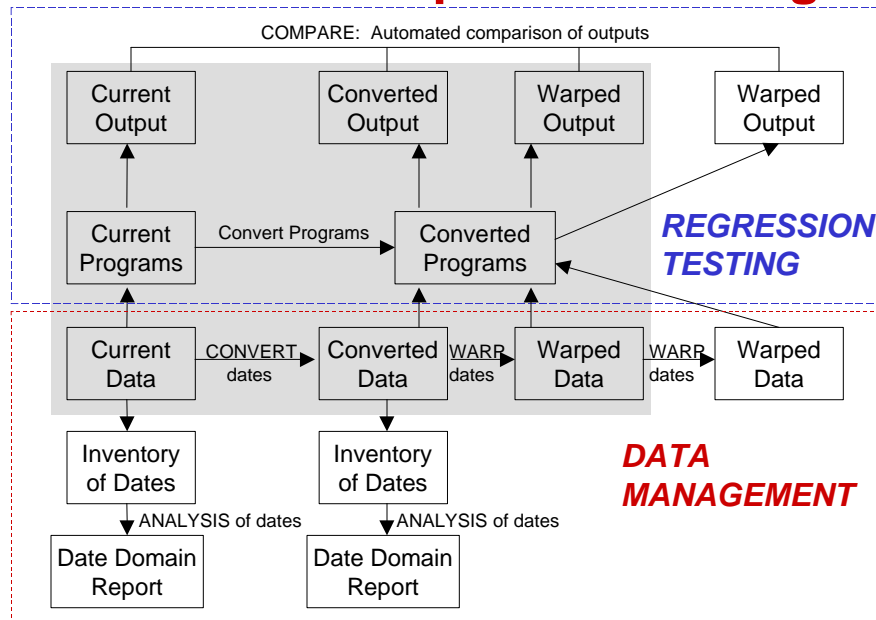
Examples

- **Fixed Asset System**
 - Information-oriented business
 - Running well, documentation poor
 - Wide range of date values
 - Large number of reports, some very long
- **Accounts Payable/Purchase Order System**
 - Manufacturer
 - Running poorly, documentation good
 - Narrow range of date values
 - Large number of interactions

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

Year 2000 Compliance Testing



©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

Data Management

- Clean data before testing begins
- Prevent data corruption after conversion
- Interface with data from other systems
- Deal with new data requirements

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com



Future Data Management: euro Conversions



National Currencies

Item	Currency	Amount
A	£	nnn.nn
B	DM	nn.nn
C	¥	nnn.nn
D	\$	nn.nn

Data Commander

All euros Conversion

Item		Amount
A		xx.xx
B		x.xx
C		xx.xx
D		x.xx

Data Commander

or

Both National and euros

Item	Amount	Amount
A	£ nnn.nn	xx.xx
B	DM nn.nn	x.xx
C	¥ nnn.nn	xx.xx
D	\$ nn.nn	x.xx

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

Data Commander™

- **Helps automate testing by:**
 - Discovering data problems
 - Converting data
 - Creating test data sets containing aged (“warped”) date values
 - Intelligently comparing output
- **Is platform and language independent**
 - Mainframe (OS/390), mid-range (AS/400), etc.
 - COBOL, PL/I, RPG, etc.
- **Has continuing value**
 - Test any system change
 - Manage data (e.g., convert currency)

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

ANALYSIS

- **Discover the range of date values in a data set**
 - Especially important for windowing decisions
- **Discover any invalid dates in a data set**
- **Identify exceptional values**
 - Embedded business rules (such as 9999=unknown)
- **Find data problems**
 - Date fields that are blank
 - Date fields that contain all zeroes
 - Values that are within or outside a specified range

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

CONVERT

- **Convert date formats**
 - Example: expand some or all dates from 2 to 4 digits
- **Specify “windows” for certain dates**
 - Year values 00 to 29 are preceded by 20 (2000 to 2029)
 - Year values 30 to 99 are preceded by 19 (1930 to 1999)
- **Manipulate some or all date fields**
- **Exclude some values from conversion**
 - Such as embedded business rules

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

WARP

- **Best test data is real data**
 - Warp real data to test programs and systems for future compliance
- **Flexible warping**
 - All dates together or some separately
 - Intervals by year, month, or day
 - Preservation of embedded business rules
 - Preservation of month and day-of-week
 - Weekend and holiday options

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

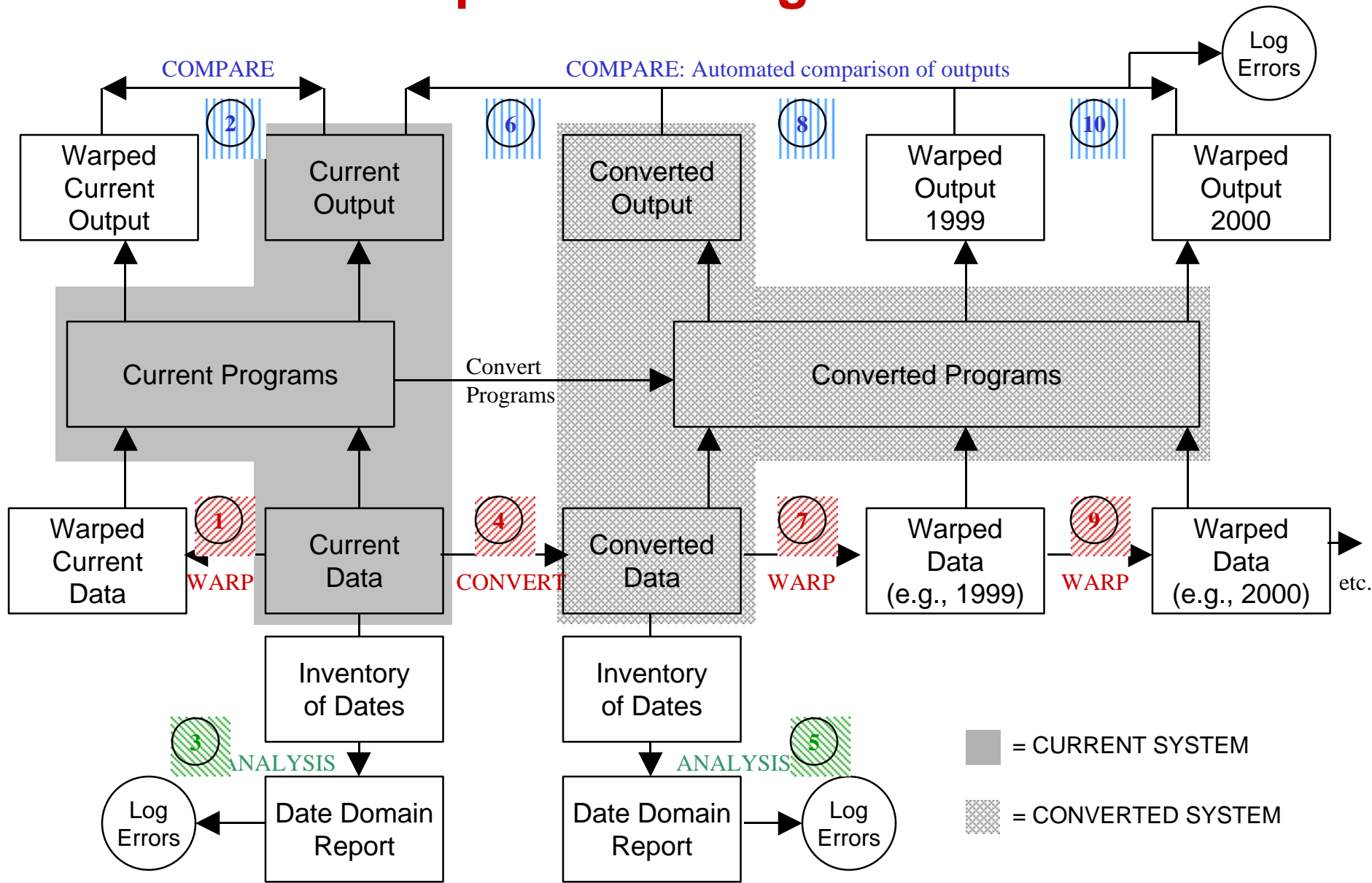
COMPARE

- **Automate comparison of outputs**
 - Reports and files
 - Analysis of dates and domain values
- **Compare warped output to un-warped output**
- **Accept changes to date formats and values**
 - 971120 to 19971120 (changed format)
 - 19971120 to 20011120 (warped date)
- **Non-procedural**
 - No loops, no gotos, no programming

©1998 Blackstone & Cullen, Inc.

www.bac-atl.com

Year 2000 Compliance Using Data Commander™



Currency Example

Directives

```
1  REMARK   *** Eurodollar Sample ***   Example of Eurodollar Conversion
2  OPTION   LINESPERPAGE 42
3  FILE DCMFSample "Sample DataCommander EuroDollar Conversion" KEY (1 30)
4  RECORD Heading1      LENGTH(54 56)      (COL(1) NOT ="I")
5  DATE     RunDate      FORMAT(YMMMDD (1),   CCYMMMDD (1))
6  FIELD    ReptID       FORMAT(POS(7,13),   POS(9,13))
7  FIELD    ReptTitle    FORMAT(POS(20,30),  POS(22,30))
8  NUMBER   Batch        FORMAT(NUM_D5(50),   NUM_D5V0(52))
9  RECORD InvoiceHdr_US  LENGTH(56 72)      (COL(1 2) ="IH"      &
                                         AND COL(46,47) = " $" )
10 FIELD    IH_ID        FORMAT(POS(1,2),   POS(1,2))
11 DATE     InvDate      FORMAT(YMMMDD (3),   CCYMMMDD (3))
12 FIELD    CustName     FORMAT(POS(9,31),   POS(11,31))
13 DATE     InvDueDate    FORMAT(YMMMDD (40),  CCYMMMDD (42))
14 FIELD    Currency     FORMAT(POS(46,2),   POS(50,2))
15 NUMBER   InvAmount     FORMAT(NUM_D7V2(48), NUM_D7V2(52))
16 FIELD    New_Currency  FORMAT(NULL,      POS(61,2))      &
                                         VALUE("E$")
17 NUMBER   New_InvAmount FORMAT(NULL,      NUM_D7V2(63)) &
                                         VALUE(1.12 * InvAmount)
18 RECORD InvoiceHdr_FF  LENGTH(56 72)      (COL(1 2) ="IH"      &
                                         AND COL(46,47) = "FF")
19 FIELD    IH_ID        FORMAT(POS(1,2),   POS(1,2))
20 DATE     InvDate      FORMAT(YMMMDD (3),   CCYMMMDD (3))
21 FIELD    CustName     FORMAT(POS(9,31),   POS(11,31))
22 DATE     InvDueDate    FORMAT(YMMMDD (40),  CCYMMMDD (42))
23 FIELD    Currency     FORMAT(POS(46,2),   POS(50,2))
24 NUMBER   InvAmount     FORMAT(NUM_D7V2(48), NUM_D7V2(52))
25 FIELD    New_Currency  FORMAT(NULL,      POS(61,2))      &
                                         VALUE("E$")
26 NUMBER   New_InvAmount FORMAT(NULL,      NUM_D7V2(63)) &
                                         VALUE(5.35 * InvAmount)
27 RECORD ItemDetail    LENGTH(55 57)      (COL(1 2) ="ID")
28 FIELD    ID_ID       FORMAT(POS(1,2),   POS(1,2))
29 NUMBER   ItemQuantity FORMAT(NUM_D8(3),   NUM_D8(3))
30 FIELD    ItemID      FORMAT(POS(11,4),  POS(11,4))
31 FIELD    ItemDesc    FORMAT(POS(15,35), POS(15,35))
32 DATE     ShipDate    FORMAT(YMMMDD (50), CCYMMMDD (50))
33 EXECUTE CONVERT PREPOST
```

Year 2000 and the euro: Testing and Data Management using Data Commander™

Introduction

This paper addresses testing (specifically concerning Year 2000) and data management (specifically concerning the advent of the euro currency) using the Blackstone & Cullen (BAC) Data Commander™ software tool. More information about the tool is on the product web site at www.datacommander.com. Information about other BAC products and services is on the company web site at www.bac-atl.com.

Year 2000 Testing

The Year 2000 problem is different from most computer problems in two basic ways. First, the cause of the problem is known. Normally, when a computer system performs incorrectly, the problem must be diagnosed and isolated. In contrast, the Year 2000 problem is one in which the cause is invariably in the capacity or processing of date fields. While it is helpful that the cause of the problem is known, the scope of the problem is unprecedented and the deadline is immovable.

A second basic difference to the Year 2000 problem is the need to test future dates once a system is converted. While testing for most computer conversions involves a before-and-after comparison, Year 2000 compliance testing requires a before-and-after-and-future comparison – i.e., you must test not only whether the converted system works today but also whether it will work when future dates are processed.

Testing is a form of risk management. The amount, type, and intensity of testing should be directly proportional to both the importance of the application being tested and the efficacy of the process used to change the system. Confidence is highly desirable but very elusive when a computer system--essential to the effectiveness if not the survival of the organization--has been changed. Testing provides confidence, but testing also has costs—i.e., time, people, facilities, and materials. Reducing these costs is critical and depends upon good management, efficient processes, skilled people, sufficient facilities, and quality tools.

Assessment & Planning: Are you vulnerable to Year 2000 problems? If so, what do you do?

The answers to some basic questions will reveal much about an organization's vulnerability to the Year 2000 problem:

- How dependent are we on computers?
 - How dependent are our suppliers?
 - How dependent are our customers?
- Does our computer system process the year 2000 correctly?
 - Can our programs handle dates after 1999?
 - Are our data files able to accommodate dates after 1999?
 - How many programs and data files do we have?
- What interfaces does our computer system have with other systems that may not become compliant at the same time?

Assuming an organization decides that it will continue to depend upon computers to some degree, the answer to the second major question can be found definitively by putting future dates into the system and seeing if it performs correctly. The concept is simple, but the test requires a set of data comprehensive enough to touch every program. We believe the best way to do this is to take your existing data and advance all the year values forward by an amount sufficient to represent what will happen for some time

before and after January 1, 2000. This is the first way in which the Data Commander™ tool can help. Data Commander's "warping" capability, which allows you to advance some or all date values in a data set by an amount of time (years, months, and days) you specify, can help you assess whether your computer system is vulnerable to problems induced by the advent of the Year 2000.

If this initial assessment reveals problems in the operations of your computer system (and—in most cases—there will be problems), you must decide on a strategy for coping with the problems. For many companies, the immovable Year 2000 deadline is already too near to allow some approaches to succeed: Not enough time remains to take all the actions necessary to enable the company's computer systems to correctly process dates subsequent to 1999. Consequently, you should rank systems, sub-systems, applications, programs, and files to help you decide which problems will receive less priority as money, time, and the supply of skilled technicians begin to run low.

Once the assessment questions have been answered and the constraints considered (a considerable task), you must decide on the conversion method for the system's "soft" components (as opposed to the hardware). Basically, repairs can be of three types:

- Change programs (install new or convert/repair old).
- Change data (being careful to keep within programs' capabilities).
- Change both programs and data.

You will naturally be tempted to choose the compliance method that takes the least time, uses the fewest people, and affects the fewest components; but this may lead to a solution that is not lasting. As with most decisions, each compliance method (expanding, windowing, compressing, etc.) has advantages and disadvantages, including the method's effect on the amount and type of testing required. Similarly, thinking ahead to the testing requirements might affect the choice of compliance methods.

Year 2000 Testing Considerations

This paper is not a tutorial on general software testing, which is explained more fully elsewhere. (For example, see William Perry, Effective Methods for Software Testing, John Wiley, 1995 and Ivar Jacobson, Object-Oriented Software Engineering, Addison Wesley, 1994.) We concentrate here on Year 2000 testing.

Some of the growing number of Year 2000 consultants state or at least suggest that Year 2000 testing is so different from other types of testing that many of the fundamental assumptions about testing are invalid. BAC agrees that the need to test future dates adds a unique dimension to the Year 2000 compliance process, but we disagree that the fundamentals of testing are somehow irrelevant. Every test should be adapted to the particular circumstances, such as availability of time, technicians, facilities, and equipment; importance of the function; specifications of the system being tested; etc. Still, the fundamental questions to be answered by the testing process are:

- What does the system look like before the change?
- What does the system look like after the change (e.g., conversion)?
- How do the two compare?

Answering these questions can involve considerable effort, but they remain the basic questions to ask when designing any test.

The diagram below depicts the fundamental process of Year 2000 compliance testing. The left-most column of boxes represents the system as it is:

- What does the system look like before the change?

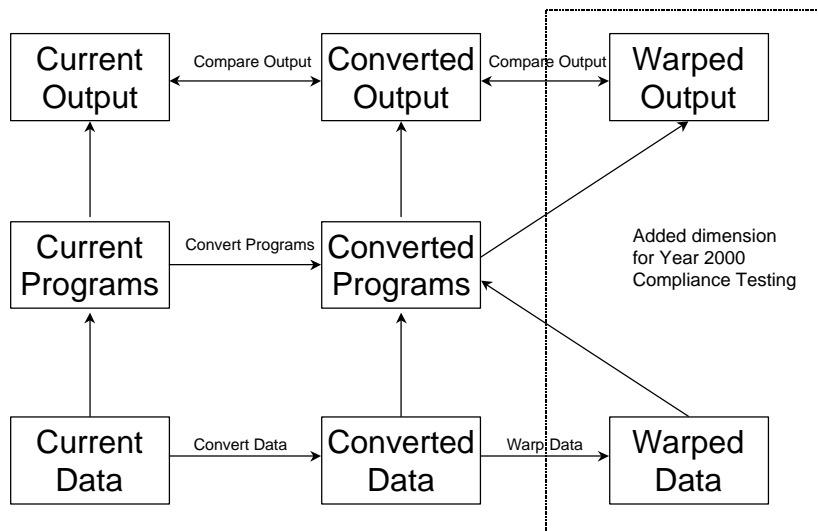
The middle column represents the system after changes:

- What does the system look like after the change?

The top two boxes (output) in these columns should be compared as the baseline test:

- How do the two compare?

Fundamentals of Year 2000 Compliance Testing



The critical, added dimension to Year 2000 testing is the need to test future dates, represented by the diagram's right-most column (enclosed by the dotted lines) containing Warped Data and Warped Output. To see if the converted system will handle dates in 1999, 2000, or beyond, you must process those dates through the system. You could do this by constructing a set of test data from scratch, but a much more thorough test would result from taking a subset of your existing data and "warping" the date values to simulate future dates. For example, if a data set contained a range of values from 1989 to 1997, warping all values in the set by six years would provide a range of values from 1995 to 2003, a good sample set for testing. Running this Warped Data through the Converted Programs and comparing the Warped Output to the Converted Output (or the Current Output) will be an essential part of Year 2000 compliance testing.

What is Data Warping?

Data "warping" is the process of advancing the date values in a file or data set by a specified number of years, months, and days. For example, warping 14 August 1997 forward by 3 years, 2 months, and 24 days yields a date of 7 November 2000. Others may use the term "aging" to describe the same process.

Data Commander™ provides a comprehensive approach to warping data for Y2K testing. The range of supported date formats, coupled with the ability to warp by years, months, and/or days, provides a great deal of flexibility in implementing a testing strategy. Further, Data Commander has specialized logical options (such as MONTHEND) which enhance the range of tests that can be set up and run. The end result is faster, less expensive, and more complete testing of applications for Y2K compliance.

Date Warp Problems

To make the testing effort as easy and as inexpensive as possible, you should try is to produce Warped Output that allows as close to an apples-to-apples comparison as practicable. Unfortunately, most calendars reflect the differences between the periods of the Earth's rotation (a day) and revolution (a year) as well as the Biblical recognition of seven days comprising a week and the Western European division of a year into months of varying lengths. Some effects of this calendar are:

- Both leap and non-leap years contain more than 52.0 weeks.
- Each year begins on a day of the week different from the prior year.
- Consecutive months often have a different number of days (e.g., April has 30; May has 31).
- Around leap years, consecutive years will have a different number of days (365, 366, 365).
- Every century year other than those divisible by 400 is not a leap year.
For example, 2000 is a leap year, while 1900 and 2100 are not.

Consequently, adding (or subtracting) a fixed time period to a collection of data having date fields will almost certainly introduce an error – either in the form of an internal data inconsistency (e.g., leap year problems) or business context problems (e.g., month-end close-outs or day-of-the-week mismatches).

Data Commander accommodates many of these problems with MONTHEND and WEEKEND OPTIONS and the DEFINE HOLIDAY statement. DEFINE HOLIDAY accommodates both specific date holidays (such as New Year's Day and Christmas, which occur on the same date every year) and holidays that may depend on day of the week or the lunar calendar (such as Thanksgiving or Easter). Data Commander can thus help you handle the following date warping problems:

Leap-year. Choosing a warp period different from an even multiple can cause several problems. Warping dates from a non-leap year to a leap-year will result in the date value of 29 Feb in the leap year (a legitimate date and important test value) not being present in the warped output date file. Possible problems in handling leap year logic for the programs and systems under test might slip through undetected. Similarly, warping data from a leap year into non-leap year poses the question of what to do with any 29 Feb values. Do they become 28 Feb or perhaps 1 March? The latter choice can upset month-end test runs, as now the February and March totals have both been altered. Finally, since leap years have 366 days while non-leap years have 365 days, computations based on per diem (daily) counts will be different. For example, interest compounded daily with an annual interest rate of X% will be effectively X/365 in non-leap years and X/366 in leap years.

Day-of-the-week. Many applications base computations on the day of the week. One example might be credit collections that do not penalize payments whose due dates happen to fall on Sunday, when mail is not delivered. In those situations, the choice of a date warping interval that does not preserve the day-of-the-week will cause the calculations to be made differently – making the apples-to-apples comparison of results difficult, if not impossible. The table below shows the day-of-the-week for 1 January for the next 28 years. Because 2000 is a leap year, this pattern works from 1901 through 2099.

Day-of-the-week for 1 January for the next 28 years

1997 Wednesday	2001 Monday	2005 Saturday	2009 Thursday	2013 Tuesday	2017 Sunday	2021 Friday
1998 Thursday	2002 Tuesday	2006 Sunday	2010 Friday	2014 Wednesday	2018 Monday	2022 Saturday
1999 Friday	2003 Wednesday	2007 Monday	2011 Saturday	2015 Thursday	2019 Tuesday	2023 Sunday
2000 Saturday	2004 Thursday	2008 Tuesday	2012 Sunday	2016 Friday	2020 Wednesday	2024 Monday

The pattern repeats with 1 January 2025 being a Wednesday. So, if the dates in the file to be tested are all in 1997 and you want to warp to a date after the year 2000 *and* preserve day-of-the-week in a non-leap year, candidate years are 2003 and 2014. To preserve day-of-the week in a leap year, a warp of 28 years is

necessary. If the file contains dates outside the current year, leap-year effects may come into play, and then a warp of 28 years is also necessary.

Month-End. Regular production runs in many applications include monthly summary reports. The problem of warping leap year data containing values for 29 February has been discussed above. Similar effects occur when the chosen date warp interval is a multiple of months other than 12. For example, with a 3-month warp interval, 31 August will warp to the end of November or beginning of December. Warping 31 August to 30 November will upset the daily value counts for 30 November, as 30 August will also warp there. Similarly, warping 31 August to December will upset the daily counts for 1 December since the values for 1 September will also be mapped there. Finally, the month-end totals for November (when compared to the unwarped month-end totals for August) will be changed with the movement of items to December.

Year-End. The same kind of problems encountered with month-end can occur for year-end comparisons when the date warping period is as fine as months or days. For example, warping by 2 months and 15 days can push all items after 16 October of a given year into the next year, possibly causing output mismatches.

The table below discusses techniques for avoiding some of the warp problems.

To avoid the Date Warping problem of:	Select the Warp Interval as follows:
Leap Year	Warp in 4-year multiples. Valid only for 1901 to 2099.
Month-End or Year-End	Warp in even year multiples. When unwarped data contains both leap year and non-leap year values, select a warp interval to be an even multiple of 28 years.
Day-of-the-Week	When unwarped data is all within a single, non-leap calendar year, use the day-of-the-week table on the previous page to select a warp interval which preserves the day. When the unwarped data spans more than one year or contains leap-year values, select a warp interval to be a multiple of 28 years. Strictly speaking, any warp interval that is an even multiple of 7 days will also preserve the Day-of-the-Week.

The table below lists the more common selections for warp intervals and discusses potential problems.

Warp Interval	Possible Problems
1, 2 or 3 years	When unwarped data contains leap year data or when the interval warps date values into leap years, Leap Year problems <i>will</i> occur. Month-End and Day-of-the-week problems <i>will</i> occur.
4 years	Leap Year problems will <i>not</i> occur between 1901 and 2099. Month-End problems will <i>not</i> occur. Day-of-the-Week problems <i>will</i> occur.
28 years	None.
Months and/or days	Leap Year, Month-End, and Day-of-the-Week problems <i>will</i> occur.

It would seem from the above table that the optimum strategy would be to always select a warp interval as a multiple of 28 years. However, this could easily warp all the dates for a field having a small (and recent) range of values well into the 21st century. This would preclude some logical coverage requirements caused by embedded business values (such as "99") in the production data.

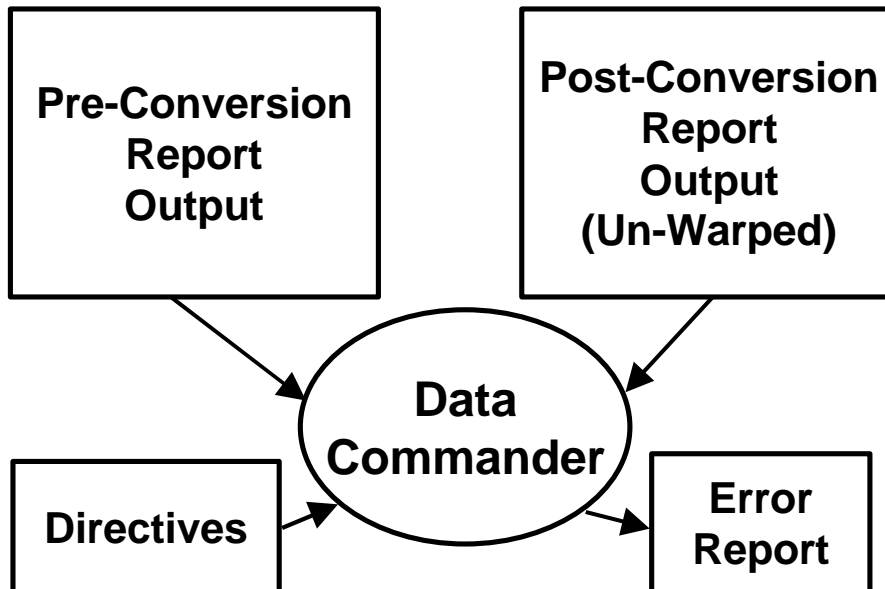
The table below lists four common date coverage objectives and the reasons to consider each.

Coverage	Reasons to Consider
Current and all-1900s	Provides a standard against which to compare warped output. Ensures that application will run in the near-term.
1999	Perhaps there are date values in the year 1999 that represent embedded rules (e.g., business conditions such as “No Expiration”), which can cause program/system failure.
Straddle	A date data sample that contains values from both the 1900s and the 2000s, so that sorts and collating sequences can be checked.
All-2000s	Ensures that the application will operate properly well past 1999.

Analysis of the date ranges in the data and thorough understanding of the operational characteristics of the application are essential to choosing the right warp intervals for testing. In fact, the correct approach is to determine test objectives first – with coverage issues being a prime consideration – before settling on test specifications (more specifically, warp intervals).

Year 2000 Output Testing

While data warping is an essential step in compliance testing, the first actual test is comparison of Pre-Conversion to Post-Conversion outputs. This test is to verify that the converted programs produce the same output after all the modifications have been made. On a report, these modifications usually involve a change in date format to add a century (e.g., YYMMDD becomes CCYYMMDD) or the slight movement or rearrangement of other fields on the report line to allow for date expansions.



Comparing *Un-Warped* Post-Conversion Output to *Warped* Post-conversion Output

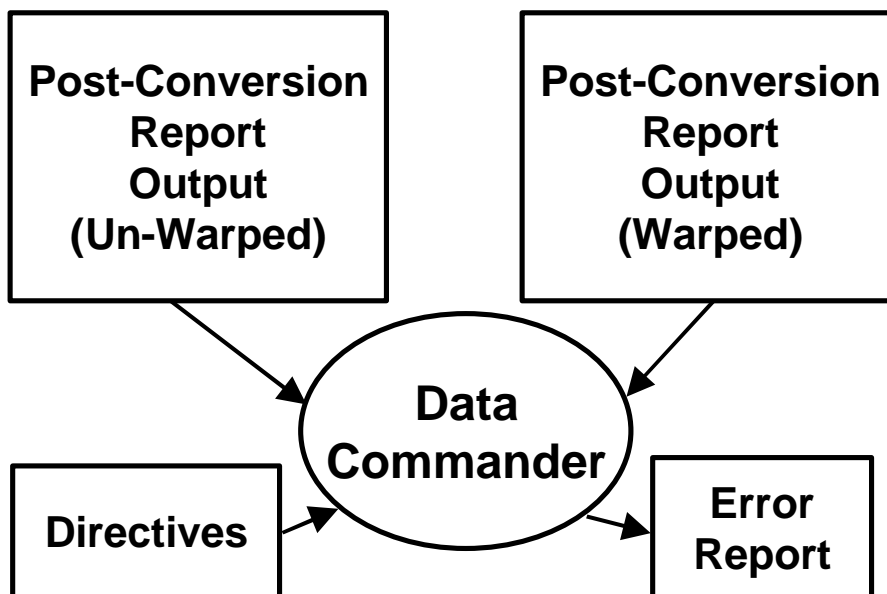
The next crucial test—unique to Year 2000 testing—is to verify that the converted programs can correctly handle dates and data in the future. This corresponds to validation of the programs using regression testing on Post-Conversion data warped by some chosen time period.

On two reports being compared under these circumstances, all the dates and data should be in exactly the same position on the output. The only difference is that the date values on the two reports are offset by the amount of the date warp. For example, the date field on a given line of the report resulting from a run of un-warped data might be 19921117. If the warped report results from a run of data warped by 28 years, the same date field should indicate 20201117. Any other date value would be an error.

This test (being sure that the converted program can correctly handle data beyond the year 2000) is also always necessary, though it is common for unit testing to exclude this test.

Without flexible and powerful tools, date warping unit tests are often considered too difficult or time-consuming. Data Commander helps in two ways:

- Ability to warp input data makes it easy to generate warped output.
- Ability to compare warped output to un-warped output makes the test easy.



Year 2000 Testing and Data Commander™

To this point, we've mentioned some of Data Commander's capabilities without laying out the full functionality of the tool. The following paragraphs review the range of features.

Refer again to the basic questions that must be answered during a Year 2000 compliance test (page 2). Data Commander's ANALYSIS function allows you to answer the "what does the system look like?" question. The COMPARE function allows you to answer "how do the current and changed systems compare?" question. The CONVERT function allows you to change date *formats* in your data, and the associated WARP function allows you to change date *values*, so you can see how the system will function in the future and thereby address the added dimension to compliance testing.

Specifically, Data Commander's three basic commands allow the following functions:

ANALYSIS Allows you to examine your data (files and records) to find invalid dates (such as April 31, 1997), blank entries, all-zero entries, or any exceptional values (e.g., 9999=end of file, xx=unknown). In this way, data can be cleaned up before it is processed. Note that this capability is continuously valuable: It is not limited to Year 2000-related problems.

CONVERT Allows you to convert date values (in almost any date format) so that they are Year 2000-compliant. For example, you can expand 08/25/97 and change its format to 19970825 with one application of the tool.

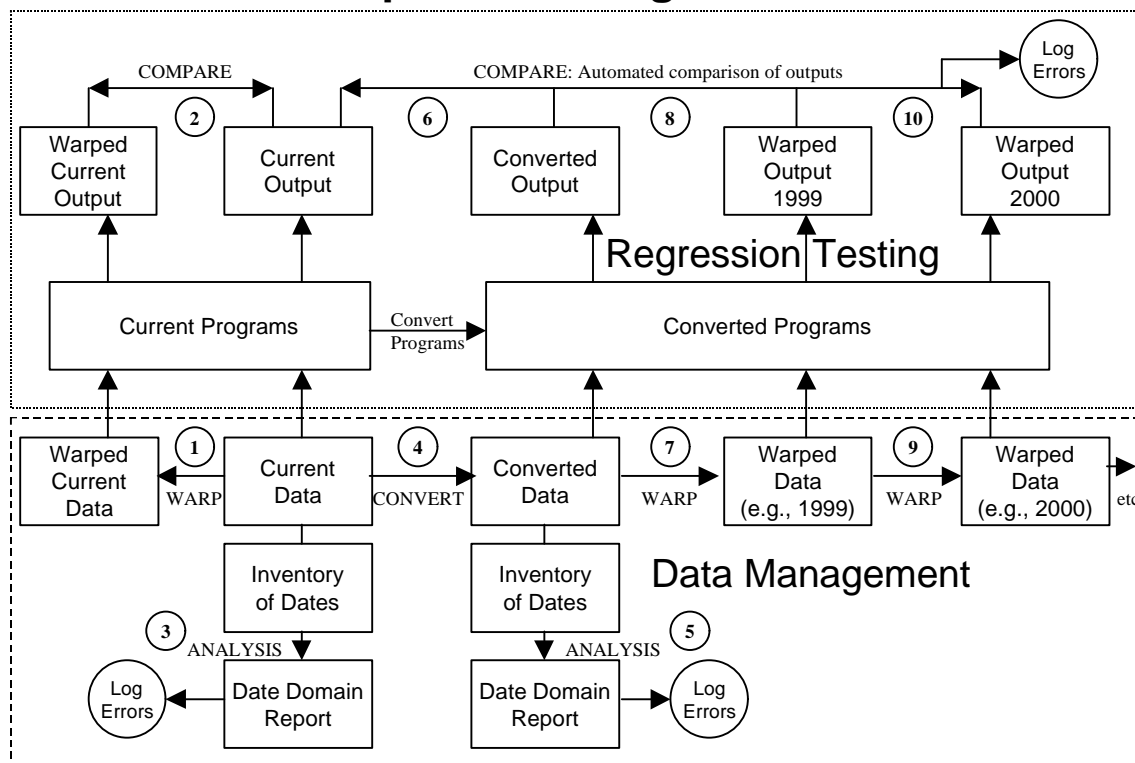
WARP An adjunct function of the Convert command is the ability to age or WARP dates. For example, warping August 25, 1997 forward by three years would yield August 25, 2000. Data Commander permits the warping of some or all dates within a file by the number of years, months, and days that you specify. This is critical for testing Year 2000 compliance in general and for testing the system's ability to handle specific dates (January 1, 2000; February 29, 2000; etc.) in particular.

A final feature of the Convert command is the ability to create test data sets. In most cases, you will not want to run every data file through the system to test for Year 2000 compliance. Rather, you will choose a subset containing representative data. Data Commander facilitates creation of such a test data set.

COMPARE Allows you to automate the comparison of entries in any specified field, not just date fields. You can test whether dates in an output report were correctly expanded and/or warped, or you can test whether depreciation values in the report were correctly preserved. You can also test whether dates in a file were correctly converted before the file is run through a program. Data Commander is flexible enough to permit comparison of dates in different formats and dates that have been warped: All you must do is specify the format and the amount of warping.

You not only want to conduct the necessary tests for Year 2000 compliance, you want to conduct them quickly. This immediately suggests automating as much of the process as possible. Shown below is a more detailed diagram of the testing process (building on the diagram on page 3) with elaboration on those parts amenable to automated testing.

Year 2000 Compliance Using Data Commander™



Step 1: Perhaps your current system is Year 2000 compliant. (You're very fortunate.) Data Commander can help you determine this by warping an existing data set or subset so that it contains values around and after 1 January 2000.

Step 2: Run the warped data set through your current programs to obtain a warped current output. You may see immediately that the warped current output is suffering Year 2000-related problems, or you may need to compare the warped current output to the normal (un-warped) current output. Data Commander can automate this comparison process.

Note: To run a warped data set through your current programs (especially on a mainframe computer), you will very likely need to simulate a future system date using a software tool designed for this purpose (such as HourGlass® or TicToc®).

Most likely, the result of steps 1 and 2 will confirm that your system is not Year 2000 compliant. You now know that you must convert or replace the programs and data files that comprise your system. While compliance plans rightly emphasize the importance of converting or replacing non-compliant programs, many neglect the equal importance of insuring untainted data.

Any organization that has amassed data over a period of years is likely to have some invalid entries (such as 31 April) in its data files. Also, many companies use special entries (such as 9999, 0000, blanks) to represent “never,” “unknown,” etc. Consequently, you should examine data files for invalid dates or special values (also known as embedded business rules) so that they can be corrected, eliminated, or reconfirmed before the files are converted or run through converted programs. Otherwise, bad output caused by data problems unrelated to the conversion will frustrate your Year 2000 compliance effort.

Step 3: Determine if any invalid dates (such as 29 Feb in a non-leap year) or embedded business rules are within your files. Data Commander’s ANALYSIS function will provide a report of invalid dates, exceptional values, blanks, and zeroes. From this report, you can repair or remove any invalid dates before further action.

Depending on the compliance method you have chosen, your next step is to convert programs or data or both. As the name implies, Data Commander operates on data. Many tool vendors specialize in program conversions, but data is equally important and tends to get overlooked.

Step 4: Data Commander’s CONVERT function can change the *format* of any date field and the *value* of any date field entry. Almost any date format is supported. Additionally, if your data conversion is performed using another tool, Data Commander’s COMPARE function can indicate whether the conversion was complete and correct.

Step 5: After converting programs and data, you should have a Year 2000 compliant system. The object of testing is to confirm that assumption. An important step is ANALYSIS of the converted data for invalid dates, exceptional values, etc. You should address any anomalies in the converted data before running it through the converted programs.

Rather than running all your converted data through your converted programs, you can expedite the testing process by selecting a representative subset of production data to serve as the test data set. Again, Data Commander can help. A subsidiary function of the CONVERT command allows you to designate certain dates or types of dates and include them in a newly created test data set, which is then run through the converted programs to obtain a post-conversion output.

Step 6: Now you must COMPARE the post-conversion output to the output from your pre-conversion system. Data Commander can do this not only for date-related fields but also for any fields you specify, such as depreciation amounts. Once this comparison has shown that the converted system functions as well as or better than the current system (probably after some more repairs and re-runs), the added process of testing specifically for Year 2000 compliance begins.

Step 7: You may use the test data set or even a smaller slice of data for the WARP test. It is your decision which dates or range of dates will be used to test whether the system is ready to process future dates correctly. Regardless of your choice, Data Commander allows you to WARP (i.e., move forward by an amount you specify) a set of dates as a function of the CONVERT command. Essentially, you CONVERT some or all dates in the test data set to create a warped test data set containing future dates. You can insure that certain dates (991231, 000101, 000229, etc.) are included in the warped data set since Data Commander allows you to specify the warp amount by the year, month, and day. More simply, if you’re satisfied that the range of dates in your test data set is sufficiently comprehensive (which you can discern from the reports resulting from the ANALYSIS function), you can specify that all date values in the test data set be advanced by a certain number of years.

- Step 8: Once you have obtained “warped” output (from a run of the warped data through the converted programs), you can again use the COMPARE function to check if the warped output correctly advances or preserves the report entries. Within the COMPARE function, you can specify that all date values in the warped output are to be considered correct if they are a certain amount of time (years, months, and days) ahead of (or behind) the corresponding entries in the baseline converted output. Further, you can COMPARE other entries (such as depreciation amounts or purchase prices) to see if they are correctly preserved. You can also compare leap year to non-leap year values by specifying whether 29 Feb should be compared to 28 Feb or 1 Mar (potentially important for reconciling monthly totals).
- Steps 9 & 10: If the particular application or set of programs you are testing does not routinely handle data sets of broad span, you can repeat the warp test, specifying a greater time advance, until your confidence level is met.

In summary, the compliance diagram’s (page 11) top portion (delineated by a dotted line box and including steps 2, 6, 8, and 10) represents Regression Testing – i.e., testing to see how the changed system compares to the existing system. The bottom portion (enclosed by a dashed line) depicts Data Management – i.e., identifying invalid or exceptional values and manipulating data for testing purposes. To the extent that the processes of regression testing and data management can be automated, the process of making your system Year 2000 compliant will take less time and use fewer technicians: This is exactly what Data Commander helps you to do. Further, you can use these capabilities in situations other than Year 2000 testing.

A Conjunction of Events

The Year 2000 represents the largest challenge most computer users will face. The problems’ scope, uniqueness, and immovable deadline lead most consultants and advisers to recommend that Year 2000 compliance efforts not be combined with other changes, such as switching from one hardware or software system to another. While this recommendation has obvious merit, most organizations’ ability to adhere to it is limited. Over the next two years (as with any two year period), companies will buy new equipment, get rid of old equipment, buy software with new capabilities, and upgrade existing software with improved versions. Further, demographic, political, and economic factors beyond an organization’s control will influence the compliance process: experienced employees will leave the organization and new ones will arrive; governments will pass laws and revise tax codes, markets will expand and contract, etc.

Similarly, dates are not the only values affected by Year 2000 compliance efforts. Many output values (interest amounts, depreciation, accrued taxes, etc.) are dependent on date calculations. Consequently, testing of converted systems must verify not only that date formats and values have been changed correctly, but also that date-related fields have been preserved correctly.

One event combining many of the aspects mentioned in the paragraphs above is the European Monetary Union’s adoption of a common currency. On January 1, 1999, the euro becomes a financial instrument, initially as the official exchange rate measurement for many European currencies. On January 1, 2002, the euro will begin displacing bank notes and coins already in circulation in as many as 11 European countries. Consequently, any organization dealing or planning to deal with European entities should consider the currency transition along with the Year 2000 transition.

Year 2000 compliance and the advent of the euro thus represent an enormous dual challenge. Fortunately, Data Commander’s data management capabilities used for data conversion and warping also allow for currency conversion.

Currency Conversions and Data Commander™

Below is a simple depiction of the process that many organizations may want to perform as their business becomes more international and the European currency conventions come into effect. The data file may contain a list of items and prices in a particular currency or currencies. Data Commander will allow conversion of all the currencies to a single currency, and it will allow retention of the original currencies in display with the conversion standard. On the following pages is an example of how Data Commander can perform such a conversion.





euro Conversions

National Currencies

Item	Currency	Amount
A	£	nnn.nn
B	DM	nn.nn
C	¥	nnn.nn
D	\$	nn.nn

Data Commander





All euros conversion

Item		Amount
A		xx.xx
B		x.xx
C		xx.xx
D		x.xx

Data Commander

or

Both National and euros

Item		Amount		Amount
A	£	nnn.nn		xx.xx
B	DM	nn.nn		x.xx
C	¥	nnn.nn		xx.xx
D	\$	nn.nn		x.xx

Example of Date and Currency Conversion

This example uses a hypothetical clothing and accessories company, specializing in Cowboy and Western wear. The subject outstanding invoice file contains four record types (header, company using U.S. dollars, company using French francs, and item), several dates (in a six-character YYMMDD Gregorian format), and several money amounts (some in dollars and some in francs). The file will be converted both for Year 2000 compliance (by expanding the date formats to CCYYMMDD) and for euro compatibility.

Here is the file, now labeled as the Input File. A column count has been added for readability.

Input File

						Column Count
0	1	2	3	4	5	6
1234567890123456789012345678901234567890123456789012345678901234567890						
970910	PRD11	INV2	OUTSTANDING INVOICES BY DATE	00001		
IH970512	ACME	MANUFACTURING CO., INC.	970612	\$000199367		
ID00000024	A435	BLUE SHIRTS SIZE L	970514			
ID00000100	B125	BROWN LOGO BELTS	970514			
ID00000024	C665	CORDUROY PANTS - BROWN	970524			
IH970512	WALLY'S	WESTERN WEAR	970530	\$000056025		
ID00000012	A445	SHARP-TOED BOOTS	970515			
IH970512	SADDLE-UP	SERVICES, INC.	970530	\$000077550		
ID00000040	D445	BUCKSKIN BRITCHES SIZE M	970514			
ID00000006	S112	NAVAJO-STYLE BLANKETS	970521			
IH970513	PIERRE'S	COTE D'AZUR	970525	FF000255000		
ID00000134	D455	SOFTSKIN VESTS	970513			
IH970513	RIVIERA	RAINGEAR	970528	FF001005000		
ID00000010	A657	LONG TRENCHCOATS	970512			

Only the most hardened of computer users will enjoy examining such files, but a brief review of this example file will help you understand how Data Commander works.

The file begins with a header record line. First is a run date (970910), then a report identification number (PRD11INV2), a title (OUTSTANDING INVOICES BY DATE), and a batch number (00001).

The second line contains a vendor record. The record type is identified by the H in column 2. Next comes a date (970512), the vendor name (ACME MANUFACTURING CO., INC.), another date (970612), and a dollar amount (\$000199367).

The third line holds an item record, identified by a D in column 2. Next comes a string of numbers and letters. Without a key to the file format, you wouldn't know whether this string contains one entry or several. In this example, the first eight characters are the field that represents the quantity of the item (00000024), while the last four characters are the item identifier (A435). The item description follows (BLUE SHIRTS SIZE L), and then another date (970514).

Within the first three lines, you see three of the record types contained in this file. Four more vendor records are apparent, but note that WALLY'S WESTERN WEAR and SADDLE-UP SERVICES contain amounts in U.S. dollars, while PIERRE'S COTE D'AZUR and RIVIERA RAINGEAR have amounts in French francs (FF). You can readily pick out the other item records (SHARP-TOED BOOTS, NAVAJO-STYLE BLANKETS, etc.).

As mentioned, Data Commander is used in this example to both expand dates and list euro equivalent (here represented by E\$) values. First, let's look at the Output file that results from the process.

Output File

	0	1	2	Column count	3	4	5	6
7	1234567890123456789012345678901234567890123456789012345678901234567890123456789012345							
	19970910PRD11INV2		OUTSTANDING INVOICES BY DATE		00001			
	IH19970512ACME MANUFACTURING CO., INC.				19970612		\$000199367E\$000223291	
	ID00000024A435BLUE SHIRTS SIZE L						19970514	
	ID00000100B125BROWN LOGO BELTS						19970514	
	ID00000024C665CORDUROY PANTS - BROWN						19970524	
	IH19970512WALLY'S WESTERN WEAR				19970530		\$000056025E\$000062748	
	ID00000012A445SHARP-TOED BOOTS						19970515	
	IH19970512SADDLE-UP SERVICES, INC.				19970530		\$000077550E\$000086856	
	ID00000040D445BUCKSKIN BRITCHES SIZE M						19970514	
	ID00000006S112NAVAJO-STYLE BLANKETS						19970521	
	IH19970513PIERRE'S COTE D'AZUR				19970525		FF000255000E\$001364250	
	ID00000134D455SOFTSKIN VESTS						19970513	
	IH19970513RIVIERA RAINGEAR				19970528		FF001005000E\$005376750	
	ID00000010A657LONG TRENCHCOATS						19970512	

A quick comparison with the Input file shows that each date has been expanded to include century digits: The format has changed from YYMMDD to CCYYMMDD. Also readily apparent is the addition of a euro (E\$) equivalent after each amount.

How was this file converted? Data Commander is platform and language independent: You don't need to know a programming language to use the tool. What Data Commander does require is a set of Directives that explain where the Input file is, what format it's in, and what you want to do with it. The Directives for this example are shown next.

Directives

A line count has been added down the left margin to facilitate the ensuing explanation of the Directives.

```

1  REMARK   *** euro Sample ***   Example of euro Conversion

2  OPTION   LINESPERPAGE 42

3  FILE DCMFSSample "Sample DataCommander euro Conversion" KEY (1 30)
4  RECORD Heading1  LENGTH(54 56)      (COL(1) NOT ="I")
5  DATE      RunDate      FORMAT(YMMMDD (1),   CCYMMMDD (1))
6  FIELD     ReptID       FORMAT(POS(7,13),   POS(9,13))
7  FIELD     ReptTitle    FORMAT(POS(20,30),  POS(22,30))
8  NUMBER    Batch        FORMAT(NUM_D5(50),   NUM_D5V0(52))

9  RECORD InvoiceHdr_US  LENGTH(56 72)    (COL(1 2) ="IH"    &
                                         AND COL(46,47) = " $" )
10 FIELD     IH_ID        FORMAT(POS(1,2),   POS(1,2))
11 DATE      InvDate      FORMAT(YMMMDD (3),   CCYMMMDD (3))
12 FIELD     CustName     FORMAT(POS(9,31),   POS(11,31))
13 DATE      InvDueDate   FORMAT(YMMMDD (40),  CCYMMMDD (42))
14 FIELD     Currency     FORMAT(POS(46,2),   POS(50,2))
15 NUMBER    InvAmount    FORMAT(NUM_D7V2(48), NUM_D7V2(52))
16 FIELD     New_Currency  FORMAT(NULL,      POS(61,2))    &
                                         VALUE("E$")
17 NUMBER    New_InvAmount FORMAT(NULL,      NUM_D7V2(63)) &
                                         VALUE(1.12 * InvAmount)

18 RECORD InvoiceHdr_FF  LENGTH(56 72)    (COL(1 2) ="IH"    &
                                         AND COL(46,47) = "FF" )
19 FIELD     IH_ID        FORMAT(POS(1,2),   POS(1,2))
20 DATE      InvDate      FORMAT(YMMMDD (3),   CCYMMMDD (3))
21 FIELD     CustName     FORMAT(POS(9,31),   POS(11,31))
22 DATE      InvDueDate   FORMAT(YMMMDD (40),  CCYMMMDD (42))
23 FIELD     Currency     FORMAT(POS(46,2),   POS(50,2))
24 NUMBER    InvAmount    FORMAT(NUM_D7V2(48), NUM_D7V2(52))
25 FIELD     New_Currency  FORMAT(NULL,      POS(61,2))    &
                                         VALUE("E$")
26 NUMBER    New_InvAmount FORMAT(NULL,      NUM_D7V2(63)) &
                                         VALUE(5.35 * InvAmount)

27 RECORD ItemDetail    LENGTH(55 57)    (COL(1 2) ="ID")
28 FIELD     ID_ID       FORMAT(POS(1,2),   POS(1,2))
29 NUMBER    ItemQuantity FORMAT(NUM_D8(3),   NUM_D8(3))
30 FIELD     ItemID      FORMAT(POS(11,4),  POS(11,4))
31 FIELD     ItemDesc    FORMAT(POS(15,35), POS(15,35))
32 DATE      ShipDate    FORMAT(YMMMDD (50),  CCYMMMDD (50))

33 EXECUTE CONVERT PREPOST

```

A review of this example Directives file shows how you can use Data Commander.

Line 1 of Directives

A **REMARK** statement is not processed by Data Commander. **REMARK** statements typically appear at the beginning of the Directives (though their order is irrelevant) and are used to clarify or identify the file or process for the user.

Line 2 (Blank lines are ignored by the parser, so they are not numbered in this example.)

The **OPTION** statement specifies any options in effect (in addition to the default options). In this case, the user is specifying that only 42 lines should be printed on each page.

Data Commander prints a report with every use. (See pages 22-23.) The first element of each report is a Directives listing, which is useful for confirming that Data Commander received the intended instructions. At the end of the Directives Listing is a summary of the options in effect, including the default options.

Line 3

The first statement for which the order, or hierarchy, is important, the **FILE** statement provides the type of file, a name for the file (enclosed in quotation marks), and the file key, in this case beginning in column 1 with 30 character length.

Line 4

The indentation of the **RECORD** line is solely for readability, but its order following the **FILE** statement is critical. This **RECORD** statement uniquely identifies the first type of record in the Input file. (Refer to both the Input File and the Directives throughout this explanation.) The type of record is **Heading1**. Its length in the Input (or **PRE**) file is 54 columns, while its length in the Output (or **POST**) file is 56 columns. The change in length is required because this record heading contains a date that will be expanded by two digits during the conversion process. This record is uniquely identified because, unlike all the other records in the Input file, its first column does not contain the letter **I**.

Line 5

Again, the indentations are solely for readability. As a subordinate to the preceding **RECORD** statement, this **DATE** statement identifies the first field—in this case, a date field—in the record. This date is identified as the **RunDate**. Its **PRE** format and start position (**YYMMDD (1)**) and **POST** format and start position (**CCYYMMDD (1)**) are identified within the **FORMAT** clause, indicating that this date will be expanded to include century digits. In this case, 970910 will become 19970910.

Line 6

The **FIELD** statement allows the user to specify fields that may not undergo any change other than position. In this example, the Report Identifier (**ReptID**) field (with entry **PRD11INV2**) must move two columns in the Output file because the preceding date field was expanded by two characters. The **FORMAT** clause for this **FIELD** statement reflects that the field will start in column 7 with length of 13 characters in the Input (**PRE**) file and will start in column 9 retaining its length of 13 characters in the Output (**POST**) file: **FORMAT(POS(7,13), POS(9,13))**.

Line 7

Similar to the previous **FIELD** statement, the **ReptTitle** field will move two columns to the right during the process.

Line 8

The NUMBER statement accommodates the Batch number field. This field must also move two columns during the process, but note the other difference in the FORMAT clause: NUM_D5 indicates a five-digit number in the PRE file, while NUM_D5V0 for the POST file again indicates a five-digit number, this time with no characters after the implied decimal indicator. In this case, the additional specifications (V0) are not to accommodate decimal digits but to prevent Data Commander from suppressing the leading zeroes in the batch number (00001).

Line 9

The second RECORD statement relates to an invoice for a company paying in U.S. dollars. As before, the record is identified by type (InvoiceHdr_US), and the PRE and POST length is specified (56 and 72, respectively). Note that the POST length will increase not only due to date expansion but also because a euro equivalent will be added during the CONVERT process. Finally, the record type is differentiated by column 1 containing I and column 2 containing H and by column 46 being blank and column 47 containing \$. The ampersand (&) at the end of the line allows the statement to be continued onto the next line.

Reports of problems with the Data Commander tool have most often been traced to mistakes in the user's construction of logical conditions, such as those just mentioned for distinguishing record types. Consequently, we urge particular care in the use of parentheses and the conjunctions "AND" "OR" and "NOT".

Line 10

The FIELD that begins this record type is named (IH_ID), and its position (beginning in column 1 with a length of 2 characters) in both the Input (PRE) and Output (POST) files is specified in the FORMAT clause.

Line 11

The first DATE field in this record type is named (InvDate) and its PRE and POST formats and starting positions specified. Again, the date will be expanded to include century digits during the CONVERT process. Also note that the formats for the date (YYMMDD and CCYYMMDD) dictate the date field length. Consequently, only the starting position (3) needs to be specified in the FORMAT clause.

Line 12

The customer name FIELD is identified, and its PRE and POST starting positions and lengths are specified in the FORMAT clause. Note that the starting position in the POST file must move because a preceding date field was expanded by two characters.

Line 13

Another DATE field is identified and specified for expansion.

Line 14

The currency FIELD is identified. Its length (2 characters) is unchanged during the CONVERT, but its starting position must move by four characters (from 46 to 50) because two preceding date fields have expanded.

Line 15

The invoice amount NUMBER field is identified. Again, the starting position for this field must move four columns from 48 to 52 during the CONVERT. Note that the format (NUM_D7V2) for the invoice amount is specified within the FORMAT clause. In this case:

NUM_	amount is a number without a delimiter between thousands
D	display (vice a signed or packed decimal) format
7	number of digits before the decimal indicator
V	decimal indicator is implied rather than shown
2	number of digits after the decimal indicator

Referring to the Input File, note line 2 for ACME MANUFACTURING: The amount on the end of the line is in U.S. dollars and has nine digits. The directives reflect that 7 of the 9 come before the decimal indicator and 2 of the 9 come after. Most likely, the company that owns this file also has an application that prints this amount as \$1993.67.

Line 16

This FIELD statement specifies that a New_Currency field will be added during conversion. Since this field is not present in the Input File, the PRE value in the FORMAT clause is NULL. In the Output File, the New_Currency field will start in column 61 and have a length of two characters. Further, the VALUE of the field will be E\$ to indicate the amount is in euros.

You may specify any value that you choose. For example, you may want ED to represent euro, or DM for Deutsch Marks, IL for Italian Lira, etc. While \$ is a symbol recognized by most applications, some symbols (such as the euro symbol not yet present on many keyboards or systems) may get scrambled between applications.

Line 17

This NUMBER statement specifies the new invoice amount (New_InvAmount) field. Again, since this field is not present in the Input File, the PRE format is NULL. The POST format is specified by NUM_D7V2 (see the explanation under line 15) and the POST field starts in column 63. Finally, the VALUE of the field entry is 1.12 times the invoice amount. The multiplier (1.12) can be adjusted to reflect the exchange rate.

Lines 18-26

Directives for another RECORD type begin on line 18. Note the distinguishing feature for this record type (compared to the one beginning on line 9) is the currency type in columns 46-47. In this case, the amount will be in French Francs (FF). The subsequent directive lines for this record type are identical to those for the similar record using U.S. dollars, with the exception of line 26, which understandably specifies a different multiplier (or exchange rate of 5.35) for the invoice amount.

Line 27

The last RECORD type is ItemDetail, with each clothing or accessory item having its own record. The length of this record type will change from 55 to 57, and it is distinguished by columns 1-2 containing ID.

Line 28

The first FIELD in records of this type is the Item Detail field, which starts in column 1 and has length of 2 characters in both the Input and Output files. (Note that POS(1,2) is equivalent to COL(1 2). The

User's Manual section on RECORD statement condition clauses provides a fuller explanation of how to use COL and POS to specify locations.)

Line 29

This NUMBER statement identifies the `ItemQuantity` field. The FORMAT clause specifies that both the PRE and POST formats are NUM_D8: a number without thousand delimiters, in display format, and 8 digits long (and, by the absence of indicators, without decimal places). The field begins in column 3. Note that the format specification (8 digit display) precludes the need to separately list the field length.

Lines 30-31

FIELD statements on these lines indicate the names, locations, and lengths of the Item ID and Item Description fields. Since no dates precede these fields, their positions do not change during the CONVERT process, nor do their lengths.

Line 32

The `ShipDate` field is identified. As with other dates, `ShipDate` entries will be expanded to include century digits.

Line 33

The EXECUTE statement specifies the process and files: CONVERT from the PRE to the POST format.

At this point, you might want to compare the Input and Output files to see how the Directives worked.

Report File

In addition to producing the Output File during this CONVERT process, Data Commander always produces a Report File (beginning on the next page) regardless of the process (ANALYSIS, CONVERT, or COMPARE). The first element of the Report File allows you to confirm that Data Commander received the Directives the user intended.

Following the listing of Directives statements is a summary of options currently in effect, including those that are default settings (such as SCAN, ERRXREF, and NOMONTHEND). Note the last four options:

<u>Option</u>		<u>Meaning</u>	<u>Example</u>
DATEEDITCHAR (DATE EDIT CHARACTER)	/	Date sub-fields will be separated by a slash.	97/09/10.
NUMEDITCHAR (NUMBER EDIT CHARACTER)	,	Numbers will separate thousands with commas.	19,387,061.
NUMDECCHAR (NUMBER DECIMAL CHARACTER)	.	Numbers will separate decimals with a period.	53,876.28.
CURRSYMBOL (CURRENCY SYMBOL)	\$	Unless otherwise specified, currency is in dollars.	\$3,423.65.

The second element of the Report File is the File Format Summary. As its name implies, this element provides a summary of the record and field formats for any files involved in whichever process was executed. The File Format Summary also provides warnings of field length mismatches, such as the fillers in this example.

The third element is the File Error Summary, though this example contains no errors.

The fourth element of the Report File shows Scan Statistics, such as the time to process, average number of records processed per second, number of records read, etc.

The final element of the Report File is the file Scan Summary, which indicates the number of records of each type, the number of dates within each record that were either exceptions (none was present in this example) or on-calendar, and the high and low values of any DATE or NUMBER fields. Knowing the range of NUMBER values can help decide the precision needed for post-conversion fields.

02/25/1998 10:43:23.55 Data Commander Directives Listing

```
*****
*      1 * REMARK   *** euro Sample ***   Example of euro Conversion
*      2 *
*      3 * OPTION   INPUTPATH   "eurosmpl.txt"
*      4 * OPTION   COMPAREPATH  "eurosmpl.txt"
*      5 * OPTION   OUTPUTPATH   "eurosmpl.txt"
*      6 * OPTION   INPUTFIXED  COMPAREFIXED OUTPUTFIXED
*      7 * OPTION   LINESPERPAGE 42
*      8 *
*      9 * FILE    DCMFSample "Sample DataCommander euro Conversion" KEY (1 30)
*     10 * RECORD  Heading1    LENGTH(54 56)      (COL(1) NOT ="I")
*     11 *      DATE    RunDate    FORMAT(YMMDD (1), CCYMMDD (1))
*     12 *      FIELD   ReptID     FORMAT(POS(7,13), POS(9,13))
*     13 *      FIELD   ReptTitle  FORMAT(POS(20,30), POS(22,30))
*     14 *      NUMBER  Batch      FORMAT(NUM_D5(50), NUM_D5V0(52))
*     15 *
*     16 * RECORD  InvoiceHdr_US LENGTH(56 72)      (COL(1 2) ="IH" &
*     17 *      AND COL(46,47) = " $")
*     18 *      FIELD   IH_ID      FORMAT(POS(1,2), POS(1,2))
*     19 *      DATE    InvDate    FORMAT(YMMDD (3), CCYMMDD (3))
*     20 *      FIELD   CustName   FORMAT(POS(9,31), POS(11,31))
*     21 *      DATE    InvDueDate FORMAT(YMMDD (40), CCYMMDD (42))
*     22 *      FIELD   Currency  FORMAT(POS(46,2), POS(50,2))
*     23 *      NUMBER  InvAmount  FORMAT(NUM_D7V2(48), NUM_D7V2(52))
*     24 *      FIELD   New_Currency FORMAT(NULL, POS(61,2)) &
*     25 *      VALUE("E$")
*     26 *      NUMBER  New_InvAmount FORMAT(NULL, NUM_D7V2(63)) &
*     27 *      VALUE(1.12 * InvAmount)
*     28 *
*     29 * RECORD  InvoiceHdr_FF LENGTH(56 72)      (COL(1 2) ="IH" &
*     30 *      AND COL(46,47) = "FF")
*     31 *      FIELD   IH_ID      FORMAT(POS(1,2), POS(1,2))
*     32 *      DATE    InvDate    FORMAT(YMMDD (3), CCYMMDD (3))
*     33 *      FIELD   CustName   FORMAT(POS(9,31), POS(11,31))
*     34 *      DATE    InvDueDate FORMAT(YMMDD (40), CCYMMDD (42))
*     35 *      FIELD   Currency  FORMAT(POS(46,2), POS(50,2))
*     36 *      NUMBER  InvAmount  FORMAT(NUM_D7V2(48), NUM_D7V2(52))
*     37 *      FIELD   New_Currency FORMAT(NULL, POS(61,2)) &
*     38 *      VALUE("E$")
*     39 *      NUMBER  New_InvAmount FORMAT(NULL, NUM_D7V2(63)) &
*     40 *      VALUE(5.35 * InvAmount)
*     41 *
*     42 * RECORD  ItemDetail  LENGTH(55 57)      (COL(1 2) ="ID")
*     43 *      FIELD   ID_ID      FORMAT(POS(1,2), POS(1,2))
*     44 *      NUMBER  ItemQuantity FORMAT(NUM_D8(3), NUM_D8(3))
*     45 *      FIELD   ItemID     FORMAT(POS(11,4), POS(11,4))
*     46 *      FIELD   ItemDesc   FORMAT(POS(15,35), POS(15,35))
*     47 *      DATE    ShipDate  FORMAT(YMMDD (50), CCYMMDD (50))
*     48 *
*     49 * EXECUTE CONVERT PREPOST
*     50 *
*      *** OPTIONS CURRENTLY IN EFFECT ***
*      SCAN
*      ERRXREF
*      NOMONTHEND
*      OUTPUT
*      INPUTFIXED
*      COMPAREFIXED
*      OUTPUTFIXED
*      CODESET          EBCDIC
*      MAXINPUT         9,999,999,999
*      MAXERRORS        250
*      MININPUTCOL      1
*      MAXINPUTCOL      79
*      LINESPERPAGE     42
*      DATEEDITCHAR    /
*      NUMEDITCHAR     ,
*      NUMDECCHAR      .
*      CURRSYMBOL      $
*****
```


02/25/1998 10:43:23.55 Data Commander File Format Summary

```

*****
Record / Date/Field/Number Name Pre Format Length Start End Scal/Wnd Post Format Length Start End Scal/Wnd
*****
*
* Heading1
* RunDate YMMDD 6 1 6 CCYYMMDD 8 1 8
* ReptID 13 7 19 13 9 21
* ReptTitle 30 20 49 30 22 51
* Batch NUM_D5 5 50 54 NUM_D5V0 5 52 56
*
* InvoiceHdr_US
* IH_ID 2 1 2
* InvDate YMMDD 6 3 8 CCYYMMDD 8 3 10
* CustName 31 9 39 31 11 41
* InvDueDate YMMDD 6 40 45 CCYYMMDD 8 42 49
* Currency 2 46 47 2 50 51
* InvAmount NUM_D7V2 9 48 56 NUM_D7V2 9 52 60
* New_Currency NULL 2 61 62
* New_InvAmount NULL NUM_D7V2 9 63 71
* Filler#1 0 57 56 1 72 72
* *** WARNING *** PRE- AND POST- FIELD/FILLER LENGTHS DO NOT MATCH
* *** WARNING *** PRE- FIELDS/FILLER EMPTY/NULL
*
* InvoiceHdr_FF
* IH_ID 2 1 2
* InvDate YMMDD 6 3 8 CCYYMMDD 8 3 10
* CustName 31 9 39 31 11 41
* InvDueDate YMMDD 6 40 45 CCYYMMDD 8 42 49
* Currency 2 46 47 2 50 51
* InvAmount NUM_D7V2 9 48 56 NUM_D7V2 9 52 60
* New_Currency NULL 2 61 62
* New_InvAmount NULL NUM_D7V2 9 63 71
* Filler#2 0 57 56 1 72 72
* *** WARNING *** PRE- AND POST- FIELD/FILLER LENGTHS DO NOT MATCH
* *** WARNING *** PRE- FIELDS/FILLER EMPTY/NULL
*
* ItemDetail
* ID_ID 2 1 2
* ItemQuantity NUM_D8 8 3 10 NUM_D8 8 3 10
* ItemID 4 11 14 4 11 14
* ItemDesc 35 15 49 35 15 49
* ShipDate YMMDD 6 50 55 CCYYMMDD 8 50 57
*****

```

02/25/1998 10:43:23.55 Data Commander File Error Summary

Note: No Errors in this case

```

*****
Scan Statistics - Start 02/25/1998 10:43:24. Stop 02/25/1998 10:43:24. Elapsed = .17 Secs; Avg Recds / Sec = 82.35
*** 14 Records Read from the Input File
*** 0 Errors encountered during input scan process
*** 14 Total Records Written to Output eurosmpo.txt (00000000872 BYTES)
*****

```

02/25/1998 10:43:23.55 Data Commander File Scan Summary

```

*****
Record Date/Number Total Exceptions On-Cal/Legit Low/High Blank Zero Other
*****
* Heading1 1
* RunDate 1 1997/09/10
* Batch 1 1997/09/10 1 1
*
* InvoiceHdr_US 3
* InvDate 3 1997/05/12 1997/05/12
* InvDueDate 3 1997/05/30 1997/06/12
* InvAmount 3 560.25 1,993.67 627.48 2,232.91
* New_InvAmount 3
*
* InvoiceHdr_FF 2
* InvDate 2 1997/05/13 1997/05/13
* InvDueDate 2 1997/05/25 1997/05/28
* InvAmount 2 2,550.00 10,050.00
* New_InvAmount 2 13,642.50 53,767.50
*
* ItemDetail 8
* ItemQuantity 8 6 134
* ShipDate 8 1997/05/12 1997/05/24
*****

```

Summary

Year 2000 testing is different from other computer system testing because of the need to test future dates.

Factors affecting successful completion of testing:

- Complexity and number of programs and data files
- Time available
- Soundness of plan
- Effectiveness of management
- Availability of skilled personnel
- Quality of testing tools

One of the more controllable factors affecting successful compliance is the selection of effective and efficient testing tools. Blackstone & Cullen's Data Commander software tool can help with almost every aspect of the testing process and with data analysis and conversion. Further, Data Commander is useful whenever data management and regression testing is needed, not just for Year 2000 compliance.

Year 2000 compliance efforts will not occur in isolation from other events. The most significant coincident event for many computer users will be the advent of a European common currency, the euro. The same data management capabilities that make Data Commander useful for Year 2000 compliance also make it useful for accommodating currency conversions.

No one tool does everything needed by everyone, but Data Commander is a powerful and flexible tool that will help any organization cope with the foreseeable future's two biggest computer system challenges.

Remote Testing Technology

Using TestWorks as Your Quality Agent
A Special Application of TestWorks Technology

By

Edward Miller
Software Research, Inc.

Software Research, Inc

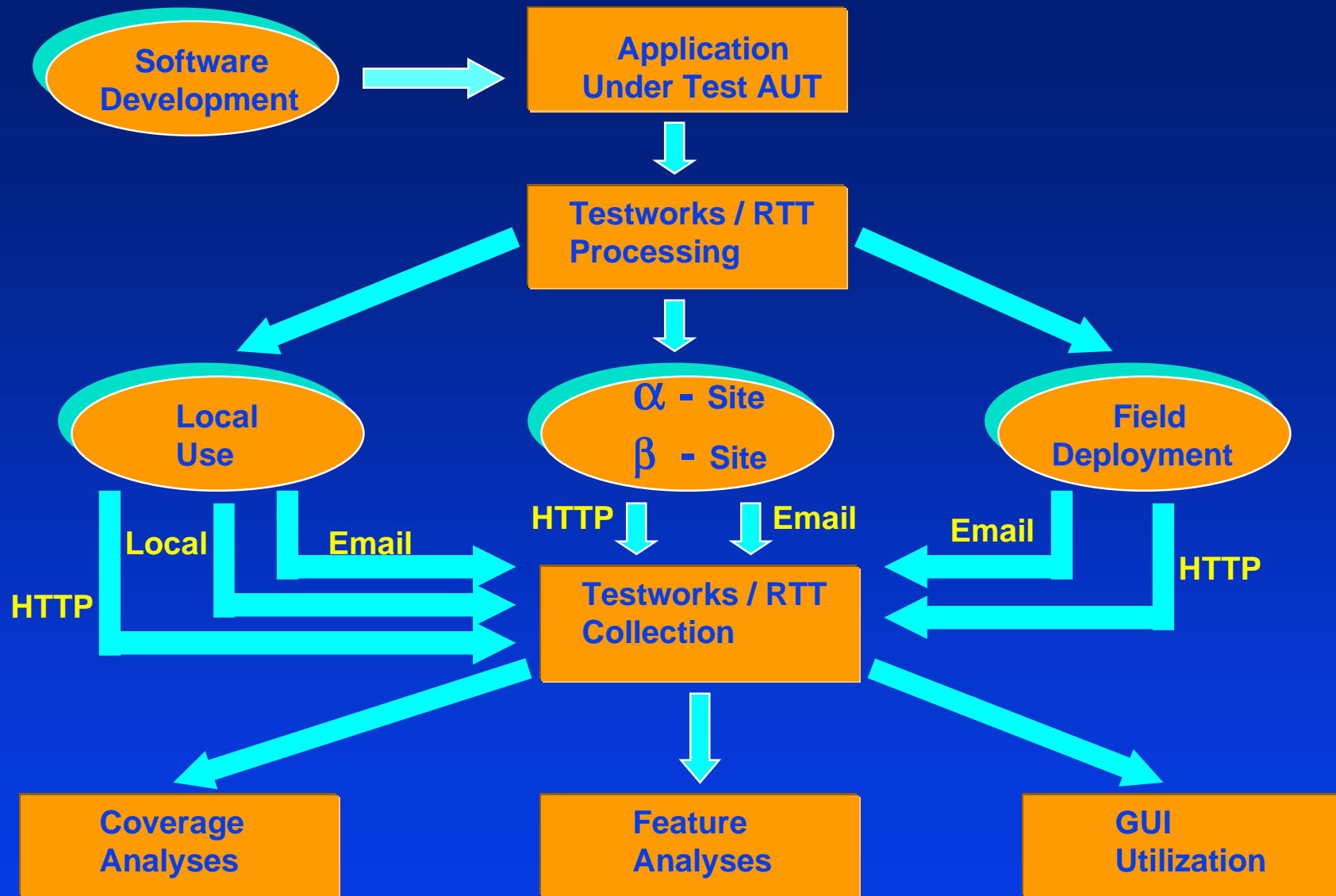


RTT Concept

- **Preparing the Application Under Test (AUT)**
- **User Operation of AUT**
- **Collection of Data from Field**
- **Reduction of Data**
- **Feedback to AUT Developers**



Remote Testing Technology



RTT Benefits / Applications

- Very Powerful Data Collection
- Very Accurate Information
- Highly Adjustable to Focus on Specific Areas
- Automated Alpha - Testing
- Automated Beta - Testing
- True Field - Based User Testing



User Information Collectable

- **Module Coverage**
- **Branch Coverage Within Module**
- **CallPair Coverage Between Modules**
- **Object-Mode GUI Usage**
- **TrueTime-Mode Keyboard/Mouse Usage**
- **User-Specified Feature Execution**



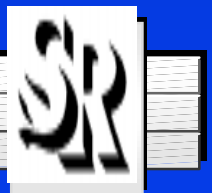
Data Collection Modes

- Complete Data
- Last N Events / Hits
- Last N Seconds Activity
- Rotating Buffer



Data Transmission Modes

- **Local Collection**
 - ★ **Efficient**
 - ★ **Fast**
- **Email Submission**
 - ★ **Slower**
 - ★ **Visible**
- **Web Transmission**
 - ★ **Quick**
 - ★ **Invisible**



GUI Recording Capability

- TrueTime Mode
- Object Mode
- Mixed TrueTime and Object Mode



Code Instrumentation Capability

- Branch Level
- CallPair Level
- Per Module
- Per Set of Modules
- Local De-Instrumentation



Feature Test Capability

- **Passive Formatted Comments**
- **Very-General Capability**
 - ★ **User Features**
 - ★ **Desired Sequences**
 - ★ **Hidden Features**
 - ★ **Critical Checkpoints**



Analysis Capabilities

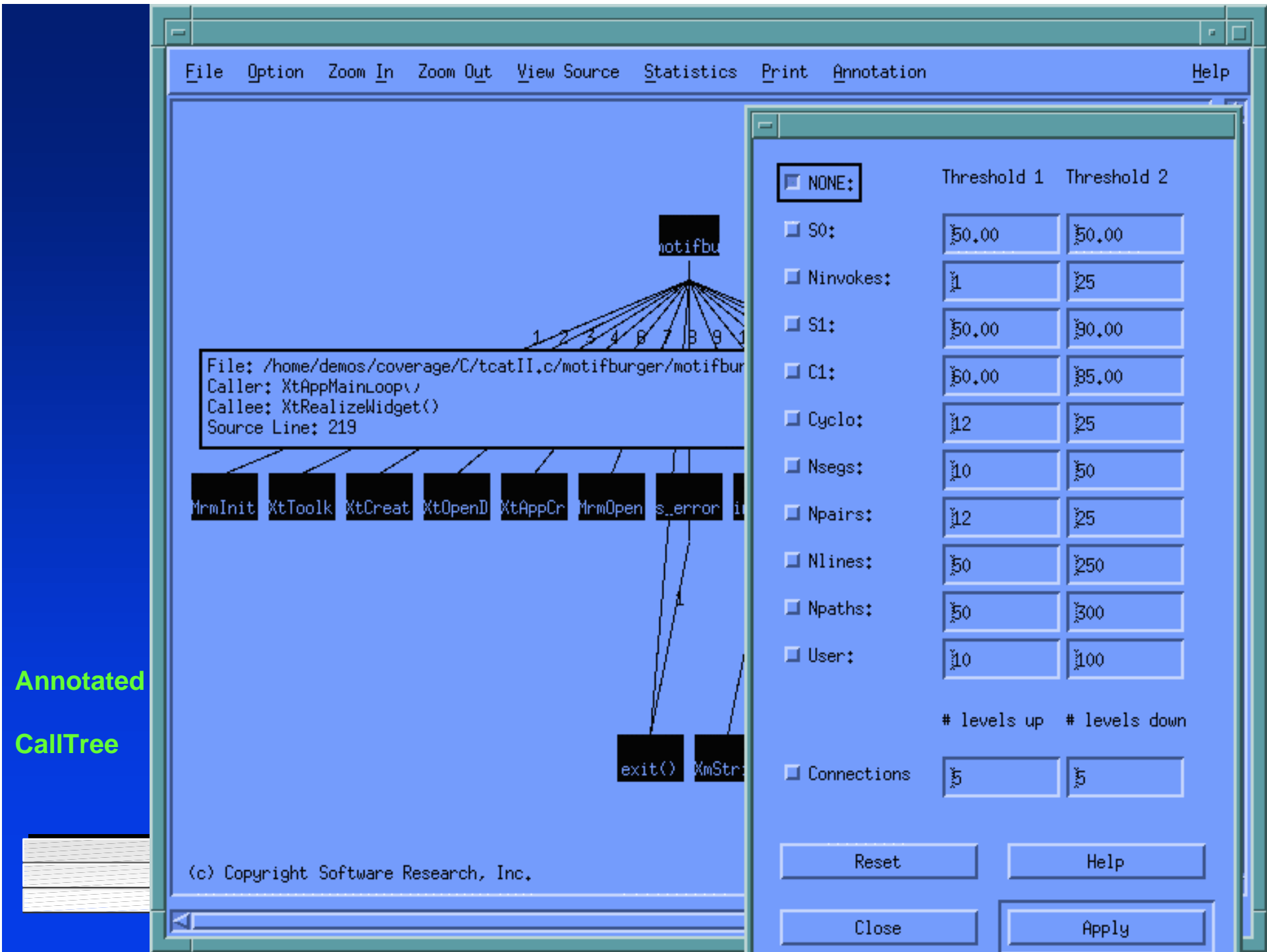
- Low-Level Code Execution
- Object-Mode GUI Interaction
- Feature Test Mode



Analysis Code Execution

- TraceFile
- Coverage
- Digraph of Function
- CallTree of Subsystem





**Annotated
CallTree**

File Option Zoom In Zoom Out View Source Statistics Print Annotation Help

File: /home/demos/coverage/C/tcatII.c/motifburger/motifbur
Caller: XtAppMainLoop()
Callee: XtRealizeWidget()
Source Line: 219

	Threshold 1	Threshold 2
<input checked="" type="checkbox"/> NONE:		
<input type="checkbox"/> S0:	50.00	50.00
<input type="checkbox"/> Ninvokes:	1	25
<input type="checkbox"/> S1:	50.00	90.00
<input type="checkbox"/> C1:	50.00	85.00
<input type="checkbox"/> Cyclo:	12	25
<input type="checkbox"/> Nsegs:	10	50
<input type="checkbox"/> Npairs:	12	25
<input type="checkbox"/> Nlines:	50	250
<input type="checkbox"/> Npaths:	50	300
<input type="checkbox"/> User:	10	100
	# levels up	# levels down
<input type="checkbox"/> Connections	5	5

Reset Help

Close Apply

(c) Copyright Software Research, Inc.

Annotated Digraph

The screenshot displays a software window with a menu bar (File, Options, Zoom In, Zoom Out, View Source, Statistics, Print, Annotation, Help) and a main area containing a digraph. The digraph consists of nodes 0, 1, 2, -3, and 4. Node 0 is at the top, with a downward arrow labeled '1' pointing to node 1. Node 1 has a downward arrow labeled '3' pointing to node 2. Node 2 has a downward arrow labeled '4' pointing to node -3. Node -3 has two curved arrows pointing to node 4. A text box at the bottom of the digraph area contains the following information:

```
File: /home/demos/coverage/C/tcatII.c/motifburger/motifburger.c
Function: 2
Segment: 0 Type: IF
Expression: <MrmOpenHierarchy(db_filename_num,db_filename_num)
Result: 1 Start line: 199 End: 200
```

Overlaid on the right side of the window is a configuration dialog box. It features a list of options with checkboxes: NONE (checked), Nhits, N%, Nlines, User, and Highlight. To the right of these options are two columns of input fields labeled 'Threshold 1' and 'Threshold 2'. Below the input fields is a 'Path file:' label and a text input field. At the bottom of the dialog are four buttons: Reset, Help, Close, and Apply.

	Threshold 1	Threshold 2
<input checked="" type="checkbox"/> NONE:		
<input type="checkbox"/> Nhits:	1	10
<input type="checkbox"/> N%:	10.00	90.00
<input type="checkbox"/> Nlines:	5	25
<input type="checkbox"/> User:	10	100
<input type="checkbox"/> Highlight:		
Path file:	...	

(c) Copyright Software Research, Inc.

Xcover Report

File Action Options View Source

PROJECT: <TCAT> TRACEFILE: <Trace,trc> ARCHIVE: <Archive>

	Current		Archive		Hits		Counts		C1 Cov.	
	Files	Modules	Files	Modules	Segs	CPs	Segs	CPs	Cur.	Arch.
PROJECT TOTAL	1	18	1	18	16	19	101	97	15.84	48.00
<init_application(>>					5	6	5	6	100.00	100.00
<s_error(>>					0	0	1	1	0.00	0.00
<set_something(>>					0	0	1	1	0.00	100.00
<get_something(>>					0	0	1	1	0.00	100.00
Segment 1					0	1				
Callpair 1										
<set_boolean(>>										
<update_drink_display(>>										
Segment 1										
Callpair 1										
Callpair 2										
<reset_values(>>					0	0	3	4	0.00	100.00
<clear_order(>>					0	0	1	1	0.00	100.00
<activate_proc(>>					0	0	35	35	0.00	37.00
<toggle_proc(>>					0	0	1	0	0.00	0.00
<list_proc(>>					0	0	1	2	0.00	0.00
<scale_proc(>>					0	0	1	0	0.00	100.00
<show_hide_proc(>>					0	0	3	3	0.00	66.00
<show_label_proc(>>					0	0	7	5	0.00	0.00
Segment 1					0	0				
Segment 2					0	0				
Segment 3					0	0				

Action

```

297  * If we need to retrieve more than one,
298  * into one arglist and we should make o
299  * more efficient).
300  */
301
302  static void get_something(w, resource, v
303      Widget w;
304      char *resource, *value;
305  {
306      Arg al[1];
307
308      XtSetArg(al[0], resource, value);
309      XtGetValues(w, al, 1);
310  }
311
312
313  /*
314  * Keep our boolean array current with t
315  */
316
317  static void set_boolean(i, state)
318      int i;
319      int state;
320  {

```

File: /home/demo/coverage/C/tcatII.c/motifburger/motifbu
Function: get_something()
Segment: 1 Type: NULL
Expression: (1)
Result: 0 Start line: 308 End: 308

WIN Cover Report

COVER for Windows - [trace.trc]

File View Window Help

Project Name: Scribble Trace File: C:\Program Files\Software Research\Coverage\TCAT\Scribble-VC5.
 Update Archive Archive File: C:\Program Files\Software Research\Coverage\TCAT\Scribble-VC5.

Current Archive		Hits Records		Counts		C1 Coverage %		S1 Coverage %	
Files :	103	103		Segs	CPs	Segs	CPs	Cur.	Cum.
Functions :	102	102						Cur.	Cum.
Project Totals :		842	104	176	189	29.55	68.18	31.75	87.83

C:\PROGRA~1\SOFTWA~1\COVERAGE\TCAT\SCRIBB~1.0\SCRIBBLE

CScribbleApp::OnAppAbout(void)	Function Totals :	0	0	1	1	0.00	100.00	0.00	100.00
	Segment 1	0	[2]						
	Callpair 1		0	[2]					
CAboutDlg::GetMessageMap(CAFX_MSGMAP*)	Function Totals :	0	0	1	0	0.00	100.00	0.00	0.00
	Segment 1	0	[443]						
CAboutDlg::_GetBaseMessageMap(CAFX_MSGMAP*)	Function Totals :	0	0	1	0	0.00	100.00	0.00	0.00
	Segment 1	0	[52]						
CAboutDlg::DoDataExchange(void,CDataExchange*)	Function Totals :	0	0	1	1	0.00	100.00	0.00	100.00
	Segment 1	0	[4]						
	Callpair 1			0	[4]				
CAboutDlg::CAboutDlg(void)	Function Totals :	0	0	1	0	0.00	100.00	0.00	0.00
	Segment 1	0	[2]						
CScribbleApp::InitInstance(int)	Function Totals :	5	14	9	15	55.56	55.56	93.33	93.33

For Help, press F1

NUM

Feature Test Mode

- Raw Event Log
- Coverage Counts



GUI Interaction

- **Raw Log (Recording)**
- **Playback on Application**



RTT Licensing Options

- **Site License Required**
- **Special Software Bridges Needed**
- **Required 1-month Consulting Option**



Contact Information:

Software Research, Inc.
625 Third Street
San Francisco, CA 94107 - 1997 USA

Phone: [1] (415) 957 - 1441

FAX: [1] (415) 957 - 0730

Email: sales@soft.com

WebSite: <http://www.soft.com>

Software Research, Inc



Year 2000 Functional Testing

Gordon Tredgold
The Testing Consultancy
tredgold@testing-consultancy.com



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Introduction

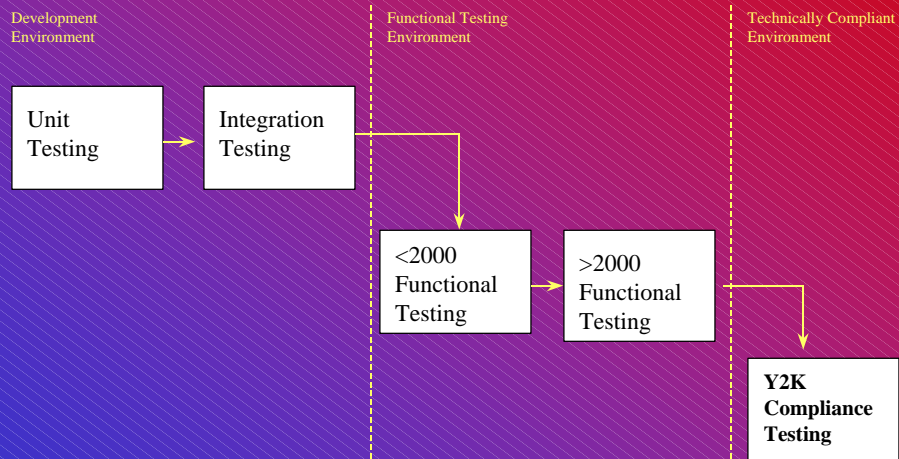
- Overview of Year 2000 Testing Approach
Functional Testing
Testing Timescales

Case Study of Testing Pilot



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Overall Y2K Testing Approach



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Test Types

- **Unit Tests**
ensure all new and amended code work as planned
- **Integration Test**
ensure that all changed modules within a subsystem work together
- **<2000 Functional Testing**
ensure the application meets the user requirements in a below 2000 env.
On completion the application can go live



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Test Types (cont.)

- **>2000 Functional Testing**
ensure the application meets the user requirements
looking forward from 19xx into 20xx, running
over 31/12/99, looking forward from 2000, and
looking back from 20xx to 19xx

Y2K Compliance Testing

ensure that the compliant functionality can work
on compliant technical infrastructure



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Functional Test Design

- Set up test calendar to run from
15th December to 1st March

This allows us to include

- Year end processing
- Month end processing (3 month ends)
- Weekly and Daily Processing



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Test Design

- Use black box testing to design tests to prove the system meets its business requirements

Use white box testing to design tests for date specific processing



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Test Design (cont.)

- Ensure that all transactions are run below and above year end where possible
- Ensure any critical date processing is included
- Ensure all functionally different data types are



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Quick design approach

- Use a RAD/JAD approach to design the tests, use workshops with the Users
- Identify all on-line txns from Txn Logs
- Identify all batch jobs from batch schedules
- Identify all data types by analysing data base type tables, e.g. Account Type



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Functional Testing Time Scales

- Time box approach
- 4 weeks design
- 4 weeks build
- 7 weeks execution <2000
- 7 weeks execution >2000
- Total = 22 elapsed weeks



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Timescales/Resource requirements

- 22 weeks elapsed to test a system
Approximately 6 months elapsed
Any team starting now Nov could test on average
2 systems by end 1999
With a team of 8 people this gives us
approximately 4 man years of effort to test a



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Prioritise

- Using the estimation metric and the number of systems requiring testing, it is easy to calculate the testing capacity required.

For most companies, especially large blue chip clients, the required capacity will exceed the available capacity and therefore the systems will need to be prioritised



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Prioritise (cont.)

- The systems need to be systems to be prioritised to ensure that risks are minimised
e.g... prioritise using the following scale
 - 1 - Critical Systems - Infrastructure
 - 2 - Critical to a business sector

This could also be used to prioritise the txns to be tested within an application



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Case Study

- Approach piloted on a pension system
The functional testing was actually to be performed 3 times
 - Prior to remediation (create baseline)
 - Prior to implementation
 - and above >2000

<2000 tests were created to run Dec 97- Mar 98
then aged for the >2000 test run Dec 99 - Mar 00



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Case Study

- Automated test tools were used
 - Mercury Winrunner was used for Capture/Replay the scripts
 - Data Aged using File Aid Data Ager
 - Date Simulated using Xchange
- For >2000 tests aged scripts, input data, expected results and actual results from baseline test
- All components were aged by the same amount



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Outcome

- The pilot was successful - the majority of tests
 - teething problems with test tools
 - learning curve with tools
 - understanding result differences
 - increased timescales for the first test run
- On completion we had an automated reusable test bed for the application



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Summary

- Take an aggressive/time boxed approach
 - Use Rad/Jad workshops to rapidly design the tests
 - Make tests reusable
 - Use tools where possible/appropriate
 - Outcome - a reusable test model that provides good test coverage



The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION

Results of the ESSI-project OMP/CAST

Experiences and results of an experiment
with
Computer Aided Software Testing
in a
Graphical and DB environment

Presentation at QWE '98
By Luc Van hamme, Jef Rymenants and Jo Wevers
OM Partners N.V. Belgium
Brussels, 9-13 November 1998

Overview



- > **Context : the company and the products**
- > **The OMP/CAST project**
- > **The experiment with Computer Aided Testing**
- > **Experiences and Results**
 - practical experiences
 - quantification of the results
- > **Preventing to (re)invent the wheel : some advice**
- > **Conclusions**

OM Partners n.v.

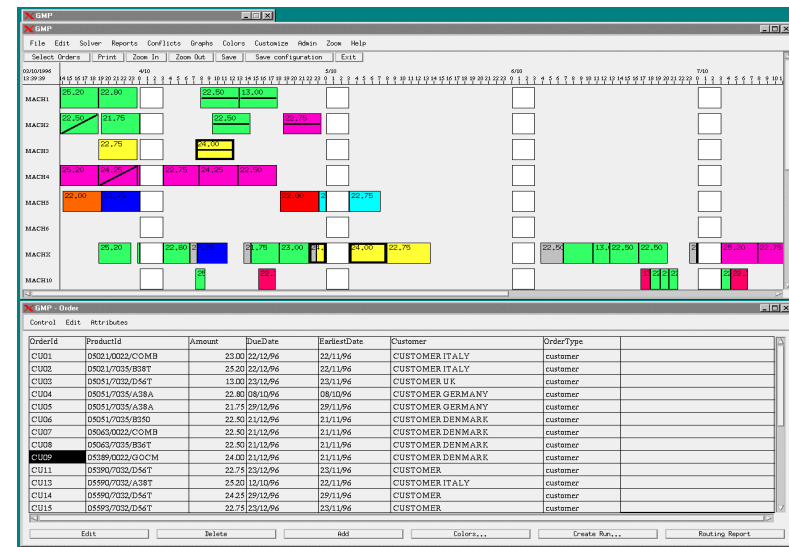
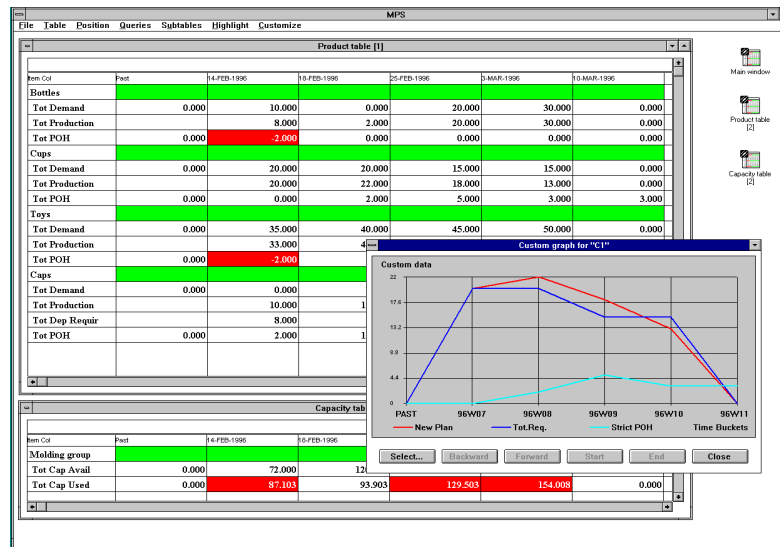
**Software and consulting company
active in developing and implementing
interactive and intelligent planning systems
in the area of operations management**

> **Some figures**

- net sales (group) : 200.000.000,- BEF (5.000.000 ECU)
- company growth : + 20 % per year
- employees (group) : 50, most of them with advanced university degrees
- customer base : over 300

> **Research and Development : 35 % of net sales (auto-financed)**

Product Specifics



> Graphical windowing environments

> Multi-user systems

- relational databases
- client server architecture

> Optimization and solver modules

- based on techniques of operations research and/or artificial intelligence

The OMP/CAST project (1)



- > **OM Partners / Computer Aided Software Testing**

- > **OMP/CAST is co-funded by the European Commission in the context of its ESSI Process Improvements Program**

- > **Objectives of OMP/CAST**
 - **select test software : regression testing, coverage**
 - **evaluate feasibility of automating test effort**
 - **compare automatic testing with manual testing**
 - **implement formal procedures based on automatic testing for the whole organization**

- > **Motivation**
 - **stable software**
 - . **daily operation of our customers' production relies more and more on our software**
 - **consistent and systematic testing of existing functionality for new releases and builds**

The OMP/CAST project (2)

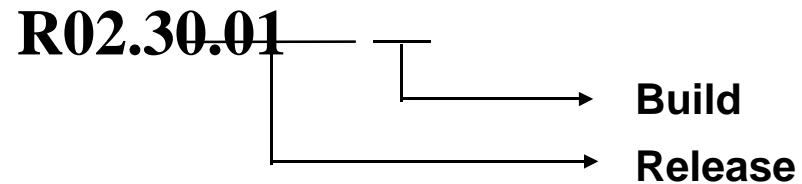


- > **Literature study on automated software testing**
- > **Establish metrics to be used during the evaluation period**
- > **Selection and installation of software tools**
- > **Creating test sets and using computer aids for automatic testing**
 - **robot testing**
 - **coverage tests**
 - **memory checking**
- > **Procedures**

The experiment - Principle



- > **Concentrate on testing old functionality with new releases and builds**



- . **new release : approximately every 3 months**
- . **new build : bug fixes only**

- > **Focus on effort needed for :**
 - **training**
 - **set up environment**
 - **maintenance**

The experiment - Set up (1)



- > **Bug reporting system**
 - log bug reports and trace progress
 - baseline for comparison with automated testing
 - logging of time spent in different tasks

- > **Tests also performed by consultants**
 - consultants know the different customer applications better
 - . consultants are responsible for modeling and implementation
 - company and customer base too small for full time testing group

- > **Product MPS was chosen for testing**
 - stable user interface
 - important ongoing development effort

- > **Tools selected :**
 - SQA Robot (regression testing)
 - Rational PureCoverage (coverage testing)
 - Rational Purify (memory checking)
 - Rational Quantify (performance analysis)

- > **Installation of testing environment**

The experiment - Set up (2)



> List of features to be tested

- software manual is one reference
- look at typical customer applications
- make a short scenario on paper

> Record test scripts

- on older software version
 - . simulate several test runs in a short term period
- on real live customer data

> Recording principle

- start record functionality
- insert test cases
 - . menu
 - . data
 - . graphical comparison
 - . timers
- edit recorded scripts
 - . scripts can be edited in a Basic like language
 - . insert comments
 - . delete/add wait states and actions

The experiment - set up (3)



> Play back and test evaluation

- play back the recorded scripts with the new release
- log viewer and built in graphical and text comparators make evaluation easy

```
SQA Robot - MPSTEST
File Edit Record Debug Insert Tools Options Admin Window Help
Main
' set the dataset on which a testrun should be performed
' if a run on all datasets has to be performed, do not specify a GoTo dataset here
' and comment the "GoTo EndLabel" command at the end of the code for each dataset
' GoTo Mpstest
' GoTo Balta
' *****
Mpstest:
SpecDataDirectory = "mpstest"
' PrepareEnvironment (SpecDataDirectory)
' following not needed if PrepareEnvironment is run
' (in that case, the data dir is already cleaned up)
' the following can be used when running SQA several times
' without having to reinstall the db-settings-...
CleanUpDataDirectory
CallProcedure "MPSSTART"
CallProcedure "EDTOPRN"
CallProcedure "END"
Sub Main
' *****
' Start test run on mps database (small test db)
' selects IEST2 bucketscheme + 16-2-96:00:00 startdate
' dumps the error/message/warning sheets in
' INIWIN_ERROR-MESSAGE-WARNING.TXT
' *****
Dim Result As Integer
Dim ApplicationString As String
' Initially Recorded: 05/07/98 15:44:26
' Test Procedure Name: Start mps db
CallProcedure "SETDIRS"
' Add the necessary options for the exe
ApplicationString = Exe & " /database=mps_cast"
StartApplication ApplicationString
Window SetContext, "Caption=Start Timehorizon", ""
ComboBox Click, "ObjectIndex=1", "Coords=166,14"
ComboListBox ScrollLineUp, "ObjectIndex=1", ""
ComboListBox Click, "ObjectIndex=1", "Text=TEST2"
EditBox Left_Drag, "ObjectIndex=1", "Coords=64,10,-43,11"
InputKeys "16-2-96"
EditBox Left_Drag, "ObjectIndex=2", "Coords=61,9,-14,5"
InputKeys "0"
```

The experiment - set up (4)



> Example of a log view and comparator

The screenshot displays two overlapping software windows from the SQA Robot suite.

SQA Test Log Viewer - MPSTEST (Left Window):

- Shows a list of test events with columns for Log Event, Result, Date, and Time.
- Key entries include:
 - Call Procedure (SETDIRS) - Pass
 - Test Case (BALT118C) - Region Image Compare - Pass
 - Test Case (BALT118D) - Region Image Compare - Pass
 - Test Case (BALT119E) - Region Image Compare - Pass
 - Test Case (BALT119A) - Region Image Compare - Pass
 - Test Case (BALT119B) - Region Image Compare - Pass
 - Test Case (BALT120A) - Region Image Compare - Pass
 - Test Case (BALT121) - Region Image Compare - Fail (Incompt)
 - Test Case (BALT121B) - Region Image Compare - Fail
 - Test Case (BALT121C) - Region Image Compare - Fail
- Bottom status bar: Description: Image Comparison Failed; Additional Information: Line: 83 RegionTC (CompareImage, "CaseID=BALT121A,Coords=5,477,1006,886")

SQA Text Comparator - BALMEN04 (Right Window):

- Master: BALMEN04.GRM
- Table view showing filters and bucket names:

Table	Filters	Bucket names
GRANADA	GRANADA	Day long
GRANADA	GRANADA	Day week
GRANADA	GRANADA	Resource reservat

SQA Image Comparator - BALT121A (Bottom Window):

- Shows a grid of image comparison results for different dates in June 1998.

Date	1-JUN-1998	8-JUN-1998	15-JUN-1998	22-JUN-1998	29-JUN-1998
Pixel count	1152000	1152000	1152000	1152000	1152000
Difference	0	0	0	0	0

Experiences and results (1)



- > **Test scripts are recorded in a Basic like language**
 - **editing possibilities are powerful but require basic programming skills of the tester**

- > **The user interface is easy to use but guarantees no “plug and play”**
 - **understanding the basics of different modelling alternatives is necessary**

- > **Use of user defined routines for preparation of data, start up and exit, ... saves time**

- > **The testing software is (very) sensitive.**
 - **e.g. changing resolution or number of colours causes graphical test cases to fail**

- > **The built-in image comparator and text comparator are powerful tools**
 - **graphical as well as data differences are easily detected**

- > **The integrated debug features are very useful for the maintenance of scripts**

Experiences and results (2)



- > **Maintenance caused the following troubles and effort**
 - **new release used different names for window titles and the robot was not able to recognize the window context**
 - **menu identification by name causes problems when testing other language versions**
 - **inconsistency in the (customer) data caused problems**
 - **in order to get the right test comparison the context of the test has to be reproduced**
 - . **editing long scripts becomes very time consuming and difficult**
 - **our software typically uses more than 10 graphical windows in a session which makes recognition and recreation of the test context extra difficult**

Testing effort figures



Robot testing	
total time spent until now : approx 50 days	
Training	11%
Set up / Getting started	23%
Recording	25%
Maintenance	29%
Test run and analysis results	6%
Coordination	7%
Total	100%

Robot testing for new build		
effort (hours) for testing of typical set of 100 actions		
	Automatic	Manual
Training	0	0
Set up / Getting started	0,5	0,5
Recording	0	0
Maintenance	0	0
Test run and analysis results	0,5	5
Coordination	0	0
Total	1	5,5

Robot testing for new release		
effort (hours) for testing of typical set of 100 actions		
	Automatic	Manual
Training	0	0
Set up / Getting started	0,5	0,5
Recording	0	0
Maintenance	8	1
Test run and analysis results	0,5	5
Coordination	0	0
Total	9	6,5

Test result figures



Number of errors detected			
Release	Regression testing	Other testing	Both
2.28	2	62	2
2.29	3	24	3
2.30	0	10	1
Total	5%	90%	6%

Errors undetected automatically	
Action not covered in script	81%
Major error interrupted automatic testing	6%
Relation with data change in new functional	13%

Coverage



- > Coverage test using PureCoverage
- > MPS contains approx 100.000 lines of specific code
- > Extensive manual testing on product OMP
 - estimate of maximum coverage reachable
 - . simpler user interface
 - . same programming techniques as MPS
 - 40.000 lines of code
 - 51% of lines hit
 - 68% of functions hit

> Maximum coverage \neq 100%

Coverage test Robot Scripts			
	OMP	MPS	
Functions hit	68%	59%	
Lines hit	51%	47%	

- unused functionality in underlying layers

homemade tools

error exits are very difficult to test

- . memory allocation
- . database errors

Advice : Recording test scripts



- > **Carefully plan the tests**
 - **list all features to be tested**
 - **write a scenario with a typical data set**
 - . **keep in mind ongoing developments and anticipate**
 - **plan big but start small**
 - . **gradually gain experience in building better test scripts**

- > **Take your time**
 - **creating test scripts is an investment that only pays if you do it thoroughly**
 - **edit and adjust the recorded scripts to make them easy to maintain**

- > **Check your Data**
 - **check for data inconsistency before recording**

Advice : Maintenance



- > **Maintenance starts at the design**
 - use several small scripts instead of long scripts
 - . they are easier to maintain
 - start scripts from a point that can be reproduced easily

- > **Don't underestimate the effort for maintenance**

- > **Apply good programming practices to your scripts**
 - add comments
 - use understandable names for your scripts and testcases
 - introduce visual structure
 -

- > **Use language independent references to menus and windows**

- > **Avoid tester input in your test scripts**

- > **Use the integrated debug functionality for debugging**

Advice : Playback and test evaluation



- > **Group your test procedure in 1 shell script (and get a coffee during playback)**

- > **Avoid interaction with other applications**
 - **A pop up mail message can destroy your playback session**

- > **Use the same computer for recording and playback**
 - **Test software has built in features for image and text comparison. Use them !**
 - **difference in graphical resolution may affect your graphical test cases**
 - **difference in calculation speed can affect synchronization of different actions in the script**

- > **Enhance performance by using memory checking**

Conclusions



- > **Automatic testing is no miracle solution but can be a good enhancement for testing existing functionality**

- > **Using CAST is a long term investment. The library of test scripts grows which makes it possible to perform tests on new builds and releases with a rather low effort.**

- > **CAST is a step towards more formal testing procedures and we expect an increase in stability of the released software**

- > **Roll out in the organization**
 - **other products**
 - . **date problem**
 - **developers as well as consultants**
 - **maintenance and training effort is the main threshold**
 - **procedures**

Results of the ESSI PIE Project OMP/CAST

Luc Van hamme, Jef Rymenants, Jo Wevers
OM Partners n.v.
Brasschaat, Belgium

1 INTRODUCTION

The OMP/CAST project, short for “OM Partners/Computer Aided Software Testing”, focuses on the testing stage of the software development cycle at OM Partners n.v.

The main business of OM Partners is consulting and software development in the area of decision support systems, and of short, medium and long term production planning and logistics.

Typically, our software makes use of graphical user interfaces, relational databases, time consuming algorithms, multi-language interfaces and is available on several platforms.

Our products are of considerable importance for the daily operation of the business of our customers. Thanks to computer aided testing, we expect to increase the stability of the released software even more, especially after modifications to existing functionality or data structures have been introduced.

In this paper, we will mainly report on the experience gained during the implementation of an automated software testing environment. We will present quantitative results as well.

2 THE OMP/CAST PROJECT

The OMP/CAST project is funded by the EC, in the context of its ESSI (European Systems and Software Initiative) Process Improvements Experiments (PIE) program.

The objective of OMP/CAST is the introduction of formal procedures and software tools for software testing. The nature of our products (graphical user interfaces, relational databases, multi-language interfaces) make this project extra challenging.

The OMP/CAST project runs from May 1st, 1997 to October 30th, 1998.

The major activities to be performed during the OMP/CAST project are the following:

- Literature study and market survey of testing tools and practices.
- Definition and implementation of a measurement method.
Two basic systems were implemented: one to trace time spent on different activities, and one to log and trace software errors and how they were detected.
- Selection and installation of software tools, including training of a core team and gaining practical experience on test examples.

- The experiment.
It consists of setting up the test environment, training all project members and testing subsequent releases of our software, while maintaining the test environment. Special attention is given to measuring the effort spent on each of these tasks, and compare it to manual testing.
- Evaluation of results and procedures.
- Dissemination of results within our company as well as in the international community.

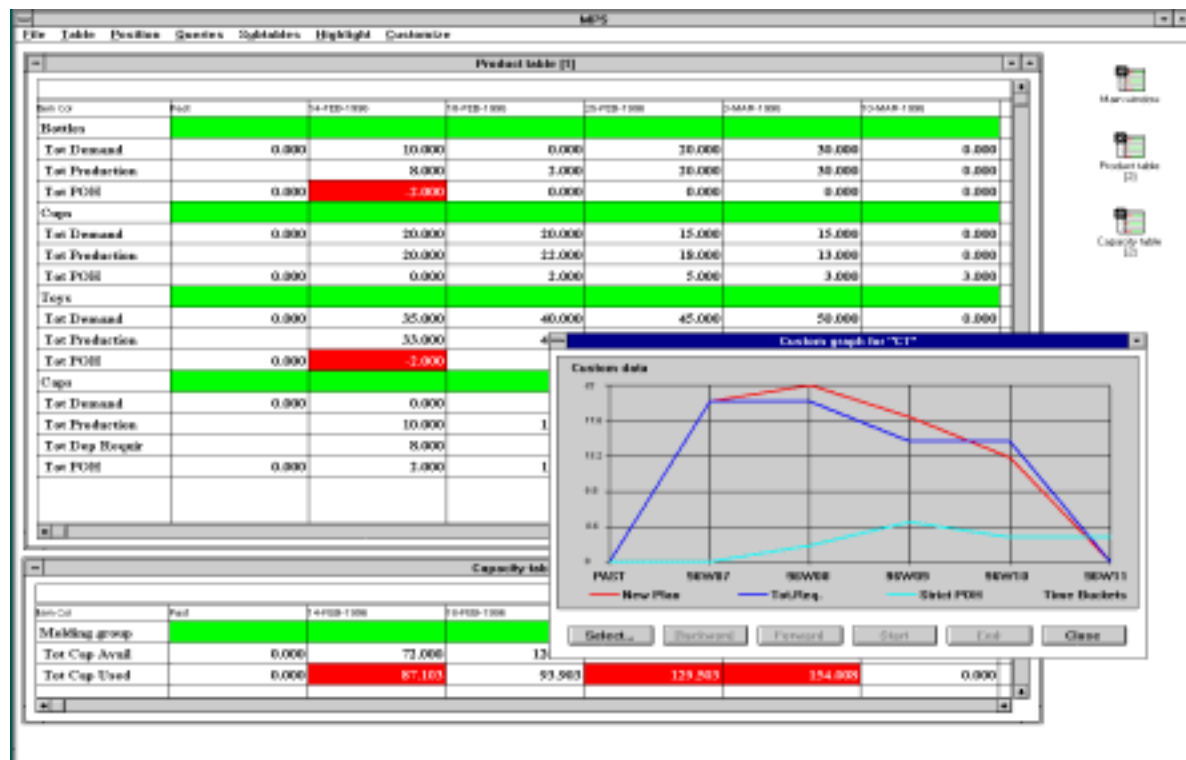
3 IMPLEMENTING AN AUTOMATED SOFTWARE TESTING ENVIRONMENT

3.1 Context

3.1.1 The OMP/MPS software

The OMP/CAST experiment is performed on the development process of the OMP/MPS package, which is the baseline project.

OMP/MPS (Master Production Scheduling) is the newest planning system of OMP Partners n.v., covering medium-term production planning. OMP/MPS is a standard product, that is used in different industries. It is a highly interactive tool, in which the user typically works in approximately 5 windows at a time, each with a spreadsheet-like look and feel. The figure below gives an idea of the user interface of OMP/MPS.



Because of the mix of classical dialog boxes and graphical objects in our products, regression tools (which replay recorded user actions) can only be useful if they support both identification by controls and by low level graphical objects.

Like most of our products, OMP/MPS can be used in multi-user environments, and therefore its data are stored in a relational database. When comparing results with reference sets, database contents should be compared as well.

Finally, OMP/MPS contains several algorithms that propose planning solutions to the user. These algorithms are based on linear programming, constraint logic programming (AI) and specific heuristics. Since these algorithms are time consuming, it is important to watch out for important increases in solution time.

All the properties mentioned here are representative for most of our software products. OMP/MPS is also in a development stage in which the user interface is fairly stable. This is why the development process of OMP/MPS constitutes a good baseline project.

3.1.2 Release policy

The release policy at OM Partners consists of introducing gradual changes in new releases rather than bringing out releases and concepts that completely replace existing functionality. This allows for a fairly high release frequency, of the order of one release every three months.

Release numbers look like

Rxx.yy.zz

in which xx.yy represents the “major and minor release number”, and zz represents the “build number”. A new (major or minor) release contains new functionality, a new build number contains fixes of software errors only. Thus, when the build number is increased, the risk of introducing new software errors is small, resulting in increasing stability of a particular software release. There are on average 3 to 4 builds per release.

3.1.3 Motivation

The main motivation for introducing automated software testing, is that new functionality is typically very well tested by both software developers and consultants, whereas existing functionality is poorly tested after addition or modification of functionality or data structures. Therefore, there is a non-negligible risk that existing functionality at customer sites is disturbed when installing new releases.

3.2 Set up of the experiment

3.2.1 Measurement

In order to correctly evaluate every effort involved in using automated testing on the one hand, and the number and type of errors discovered on the other hand, a formal measurement system was put into place. The system is based on spreadsheets and a database.

The following parameters are recorded in this measurement system:

- Time spent on a particular task, by a particular person
- When errors are detected:
 - Error description
 - Error severity
 - Was the error detected in house or at a customer site?
 - Was the error detected by automated testing or by other methods?

For traceability purposes, the further life of an error is logged as well.

3.2.2 *Initial set up*

After an initial study, we have decided to base our automated software testing system on the following components:

- Regression testing. This component consists of a tool that records user actions on a reference version of the software, and plays back these actions on newer versions of the software. The recorded user actions are stored in an editable test script. During the recording phase, reference data are stored for comparison in the playback phase. The reference data can be screen images, files or database contents. For regression testing, we have used SQA Robot of Rational Software.
- Coverage analysis. This component measures the fraction of lines of code that has been executed during a session. It gives an idea of how well testing covers the software under test. We have used Rational's PureCoverage.
- Memory checking. Almost all the code of our software is written in C. This programming language has poor protection against overwriting memory (i.e. the contents of other variables). Such errors can cause the software to crash immediately, or, even worse, to behave strangely or crash at a later point. Using a memory checking tool during playback, one can detect errors that would otherwise go unnoticed. For this component, we have used Rational's Purify.

3.2.3 *Organization*

After gaining some experience using small examples, we have introduced an environment that is easy to maintain, and that allows testers to easily introduce new test sets.

This environment is based on a directory structure per release, since the contents of test scripts will differ between releases. Because it is crucial to start each playback session from a well defined starting situation, the directory of a release contains test scripts, reference data, and (compressed) databases to start from.

The test scripts are organized in a modular way, and are called from one main script. In this way, new scripts can easily be added or temporarily deactivated, while maintaining the possibility of running all scripts at once. There is also one script containing information such as the software release under test, and relevant directory names. Another script initializes the environment, that is to say, the database is uncompressed and files are copied to a work directory.

There is no direct way to compare database contents from the regression tool. However, it is possible to call other programs. We have therefore written a utility that dumps the database.

Time stamps at the record level are filtered out, since they would disturb any comparison of the data.

The use of automated testing is mostly performed by consultants, not by developers. This is because we wanted tests to use real customer data. Due to the nature of their work, our consultants know best how these data are used by the customers.

Before recording a new test set, testers first write out a rough scenario, listing which functionality will be tested, in what order. Then, an initial database situation is defined and stored. A script is recorded, and a line, calling the new script, is added to the main script. The script is edited to add comments, wait states and possibly some extra actions.

When preparing for testing a new release, test set maintenance consists of converting the data to possible database design modifications and of adapting the scripts to modifications in the user interface. This can be modifications to window titles, to menu item texts, or to the sequence of mouse click actions due to modified dialogs with the user. This work is a mixture of script editing, script playback and partial re-recording.

4 RESULTS OF THE EXPERIMENT

In this chapter, our results are presented. First, a list of remarks and considerations is given, summarizing our experiences gained while using the automated testing environment. Second, we provide quantitative results of the measurements that we performed. Finally, some advice is supplied for those wanting to start with automated software testing.

4.1 Lessons learnt

During the many tests that we ran, we have gathered a list of notes and remarks, which we summarize here.

- Like in most regression tools, SQA Robot test sessions are stored in scripts. These scripts are in a Basic like language, with powerful editing possibilities. Editing scripts requires the tester to master basic programming skills. This is particularly true when manipulating wait states, timers or system calls.

Fortunately, an integrated debugger is available, providing possibilities such as stepping through the code, setting breakpoints and viewing the contents of variables.

- In order to easily read and maintain scripts created by other people, a programming style guide has been written. Amongst other things, it contains guidelines on indentation, and naming of variables and scripts.
- The user interface of the regression tool is very easy to use. However, this guarantees no simple “record and playback”. Knowledge of different options and modeling alternatives is essential. Examples are the aforementioned wait states, but also options concerning the identification of user interface controls.

Therefore, a substantial training period is required for each person participating in the automated testing effort. In our organization, in which there are no full-time testers, the number of individuals involved is high with respect to the time spent on testing. In order to make this group work consistently in the same way, and to decrease the initial training effort, we have established a list of options and choices with their values to be used.

- A lot of time can be saved by carefully preparing a standard testing environment like the one described in the previous chapter. Special attention should be paid to the preparation of data, such that scripts can start from a well defined state.
Furthermore, it has proven to be handy to centralize information such as directory and release number information in one script.
Also, introducing modularity reduces the amount of time needed to create new test sets.
- It is not sufficient to test whether a set of test scripts successfully terminates. One should introduce as many comparison points as possible. This can be done on output files, database contents and images (screen captures, possibly containing masked regions). The image comparator provided with SQA Robot is very powerful. It can highlight image differences, or blink (toggle) between two images.
However, since bitmaps are compared, changing the resolution or number of colours of the screen, will cause false failures. One should therefore agree upon a resolution and number of colours to be used when running the test scripts
- Menu identification can be done by the menu text or by its identifier. The former method has the disadvantage that it is language dependent, which does not allow to automatically test multi-language environments. The latter method causes severe maintenance problems, since changing the menu composition changes all identifiers. We have opted for identification by menu text, because addition of new menu items in new releases is common in our situation.
- Similarly, window identification can be done by window title or by window identifier. Again, using window identifiers caused trouble when the user interface evolved. We therefore decided to use window titles. So far, this has caused an important maintenance effort once, when we decided to change the systematic of window title texts. However, we do not expect this to occur frequently.
- New releases sometimes imply modifications to the database model and to customization parameters. Before running the test scripts, the old data have to be converted to the new situation. This should be done with great care, in order to avoid unnecessary debugging of scripts or the application. It is, however, an excellent test bench of data conversion operations at customer sites.
- When recording and playing back a session, one should avoid as much as possible the appearance of unexpected windows, such as warning messages for the arrival of new mail.
- Because of the maintenance issues listed above, we consider it necessary to record many short sessions rather than a few long ones.
Indeed, when a test scripts fails because of a maintenance problem rather than a software problem, debugging and modifying the test session requires replaying the session from a well-defined point, which is usually the start of the script. With long test scripts running for hours, this becomes very tedious and discouraging for the tester.

4.2 Quantitative results

In the previous section, we have highlighted the major observations that we made during the process of recording, maintaining and playing back automated tests sets. In this section, we will give some quantitative results, obtained from monitoring these operations.

One should be aware of the fact that quantitative results depend on the software under test, and on the commercial environment in which the organization operates. Ours is an environment with a few hundreds of customers, and software being implemented by our own consultants, who assist customers in the process of integrating planning tools in their specific planning and production environment.

4.2.1 Regression testing

In order to gather statistics on several releases in a reasonably short time period, test sets were recorded on an old release. These tests were then maintained and applied on the following builds and releases. By doing so, comparison with manual testing was possible.

We have collected data during a period corresponding to 50 person days, carefully tracking the amount of time spent on each of the individual tasks. This resulted in the following table:

Task	Effort
Training	11 %
Set up/Getting started	23 %
Recording	25 %
Maintenance	29 %
Test run and results analysis	6 %
Coordination	7 %

Table 1 Effort for each of the automated testing tasks

“Training” consists of an introductory session, in which the trainee is explained the basics of automated testing, the basics of the tools used, the procedures and testing environment that we created. He or she then goes through the manuals and learns the techniques by actually using (a copy of) the testing environment.

“Set up and getting started” is the time spent on finding out what was the best way to use the tools and their options for our particular situation, as well as determining what the test environment and procedures should look like.

Obviously, the percentage of time spent on training and getting started should decrease with time. However, the training percentage will not tend to zero, since new members of the test team will always have to be trained.

When analyzing these data, one notices two important facts that are very pronounced. The first one is that maintenance of test sets takes as much time as recording these sets. This is due to the changing user interface as functionality is added to the software. Usually, user interface modifications are not limited to extensions of the window menus, but imply addition or reorganization of dialog boxes, or even the way output looks.

The second fact is that running tests and analyzing the results takes only little time with respect to the other tasks. This is because of the ease with which data can be compared to the reference sets. Testers only have to watch failures, and for each of these failures the tools emphasize deviations from reference sets. When the failure is caused by a software error, a bug report is logged by the tester. Otherwise, further maintenance of the test script follows.

Some care should be taken in interpreting the table above. The fraction of time that maintenance takes will strongly depend on the particular software being developed, and more specifically on the amount of change to the user interface between releases. Also, if we were to bring out more builds (containing bug fixes only), the percentage of test script maintenance would drop.

In one of our tests, we have tried to evaluate the effect of the regression tool on productivity. We have therefore recorded a short test scenario, and then applied it to a new build and to a new release. This has been done with the tool, but also “manually”. The manual test consisted

of a person executing the written scenario with the OMP/MPS application, and comparing screens and files with reference print-outs and reference files. The table below shows the results, expressed in hours.

Task	Testing a new build		Testing a new release	
	Automatic	Manual	Automatic	Manual
Set up/Getting started	0.5	0.5	0.5	0.5
Maintenance	0	0	8	1
Test run and results analysis	0.5	5	0.5	5
Total	1	5.5	9	6.5

Table 2 Comparison between automatic and manual testing efforts

As was to be expected, running and analyzing the test takes only a small amount of time when done automatically, since it only consists of starting the test run and checking the results later on. However, when new releases are tested, the maintenance effort becomes important, due to the changed user interface. Obviously, the exact figure depends on how much the application’s user interface modified, but it is not hard to understand that maintaining a scenario text takes far less time than maintaining an automated test environment. The final balance between manual and automatic testing depends on the number of builds per release.

So far, we have reported on the effort involved in automated testing. Table 3 gives an idea on what the outcome on software quality is.

Release	Regression testing	Other testing	Both
2.28	2	62	2
2.29	3	24	3
2.30	0	10	1
Total	5 %	90 %	6 %

Table 3 Number of software errors found

Some comment is necessary in order to correctly interpret this table. The column “other testing” contains those errors found at customer sites and by consultants while preparing customizations in-house. This represents an amount of hours much larger than those spent on regression testing. Also, the column “regression testing” only represents errors in functionality that existed in release 2.27. The “other testing” column, however, also contains errors introduced to new tasks. Some of these errors would be detected while recording new sets, corresponding to new functionality.

What this table shows is that the regression testing method detects roughly 5 % of errors that would otherwise have remained undetected for a longer period of time. We are still recording new tests scripts. While the number of scripts grows, we expect the percentage of errors found by automated testing to increase.

4.2.2 Regression testing

Finally, we present some data on coverage of the automated testing. Coverage is defined here to be the percentage of lines or functions of an application that have actually been executed during the test. The idea is to try to test all lines of code of the application under test. The problem of measuring coverage is twofold.

First, measuring the number of lines or functions hit by the test is only an indirect measure for the number of “situations” that can occur when using the software. For instance, feeding a zero to a particular routine could very well cause the software to traverse the same number of lines as feeding a one to the routine. The resulting outcome could be very wrong in one case and correct in the other case. Yet the line coverage is the same in both cases. Since it is difficult to measure this kind of effects, one has to accept line and function coverage as an approximation of this kind of “functionality coverage”.

Second, one can never reach 100 % coverage. This is because applications use tools, in our case mostly written in-house, of which only part of the functionality is actually used. Therefore, we have tried to estimate what could be the maximum coverage for OMP/MPS. This has been done by measuring the coverage in an optimization package, OMP, based on linear programming techniques. OMP has been written in our company. It uses similar programming techniques and tools as OMP/MPS. However, the OMP user interface is much more limited, making it easier to test every functionality. After verification that no routines were missed by the test, we consider the coverage obtained with OMP to be the maximum reachable with OMP/MPS.

	OMP	MPS
Routines hit	68 %	59 %
Lines hit	51 %	47 %

Table 4 Coverage obtained during all tests

The numbers represent the cumulative result of all tests that have been recorded. One can see that the result is good. Further tests will focus on covering the missing routines.

4.3 Advice

We have now gained experience on using automated software testing. Below, we present some advice for those who want to start with similar tools. It is based on our own experience with using automated software testing. It should be interpreted as such, i.e. it is valid for environments similar to ours.

- Don't use automated software testing during the first development phase of an application.
One of the main conclusions is that maintenance of the test sets, due to modifications of the application, is one of the major efforts involved in automated software testing. If the development process is still in a phase where the user interface is under construction, playback of test sessions will always result in false failure reports.
- Carefully plan the tests.
An initial effort should be invested in creating an environment in which test sets can easily be created and modified. Reproducibility is important, so make sure that there is a version of the test set corresponding to each release of the application.
When starting the tests, one should first list all the features to be tested. Then a scenario with a typical data set should be written. Anticipate ongoing developments. With the scenario, recording can start.
- Plan big but start small.
Experience in building better tests scripts will be gained gradually. It is easier to adapt few and small test sets to new insights.
- Maintenance starts at the design.

Once experience has been obtained, one can start extending the test sets. Test sets should consist of many small scripts, not few large ones. These scripts can be called from one main script, which makes it easy to start a new playback session.

Scripts should start from a point that can be easily reproduced.

- Testers should take their time.
Creating scripts is an investment that only pays off if it is done thoroughly.
- Apply good programming practices to your scripts.
Scripts are small programs!
- Enhance performance by using memory checking tools.
Such tools allow detection of errors that go unnoticed in the output.
- Write out procedures.
The experience gained will be more easily accessible for newcomers.

5 CONCLUSIONS AND FUTURE USAGE

In this paper, we have presented a testimony of the introduction of automated software testing. A description has been given of how the environment was set up. We have listed a series of considerations when applying regression techniques, and have shown measurements which compare automated testing to manual testing. We have concluded with some advice, which will be useful for other development organizations.

We have also indicated under which circumstances automated testing tools can lead to an increase of productivity.

In our company, we will now enter a phase in which the other development teams can use the new experiences, and apply them to their applications. New challenges will certainly turn up, as we use the testing techniques with software that relies heavily on the “current date”.

In the OMP/MPS testing team, we will build further on the current impetus. One of the new possibilities that we see is using the regression tools to reproduce software error reports, even if the software errors were not found by the automated software testing environment.



Vision and Tools

Boudewijn Schokker

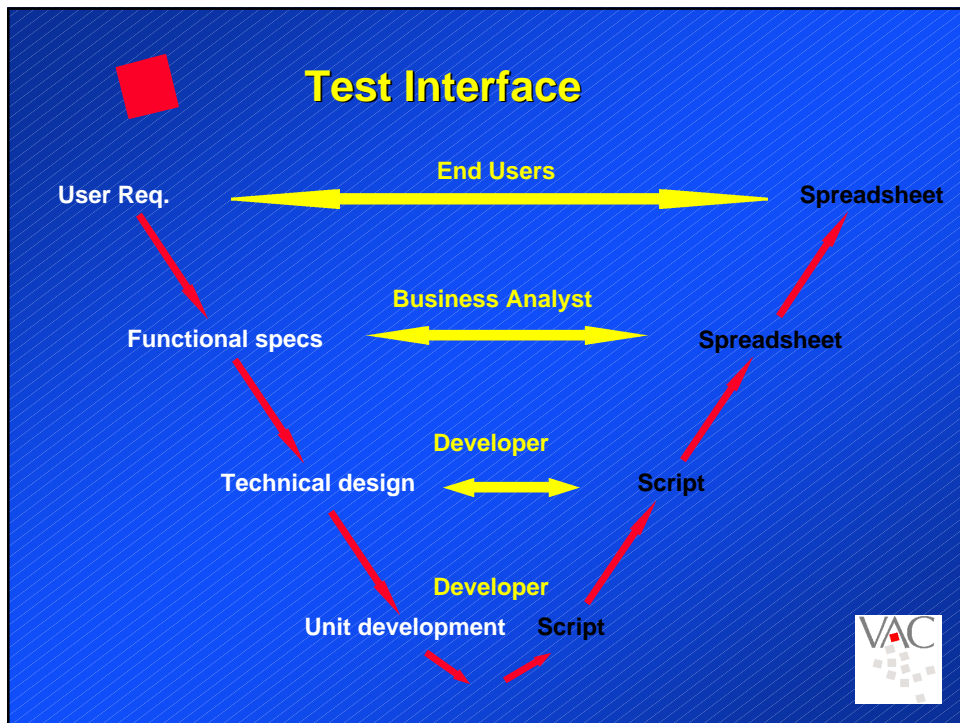
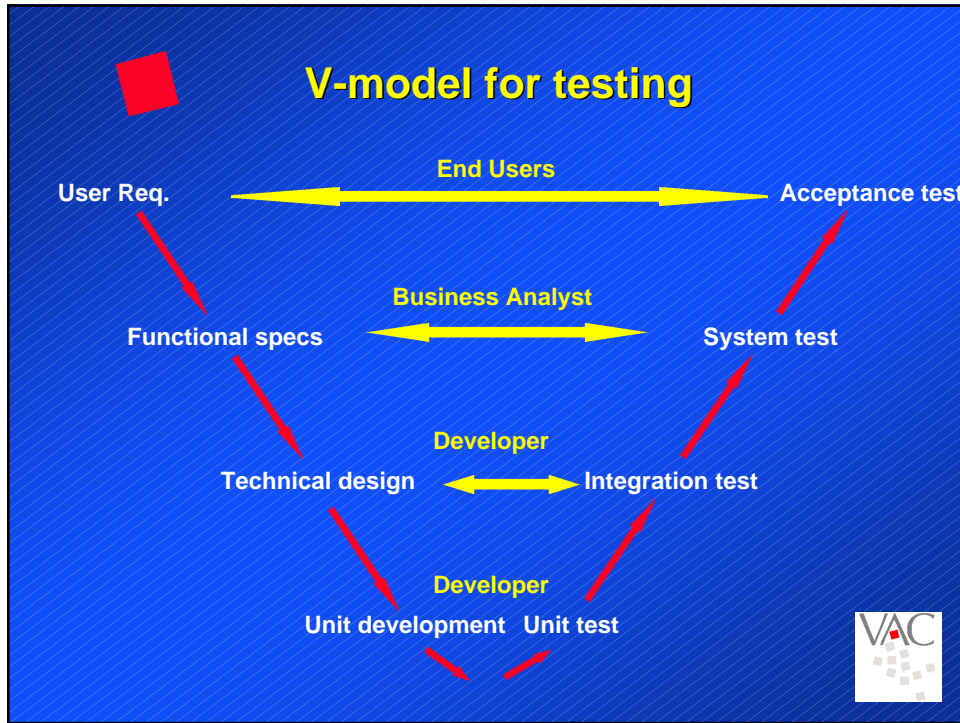
VAC Software Engineering
the Netherlands

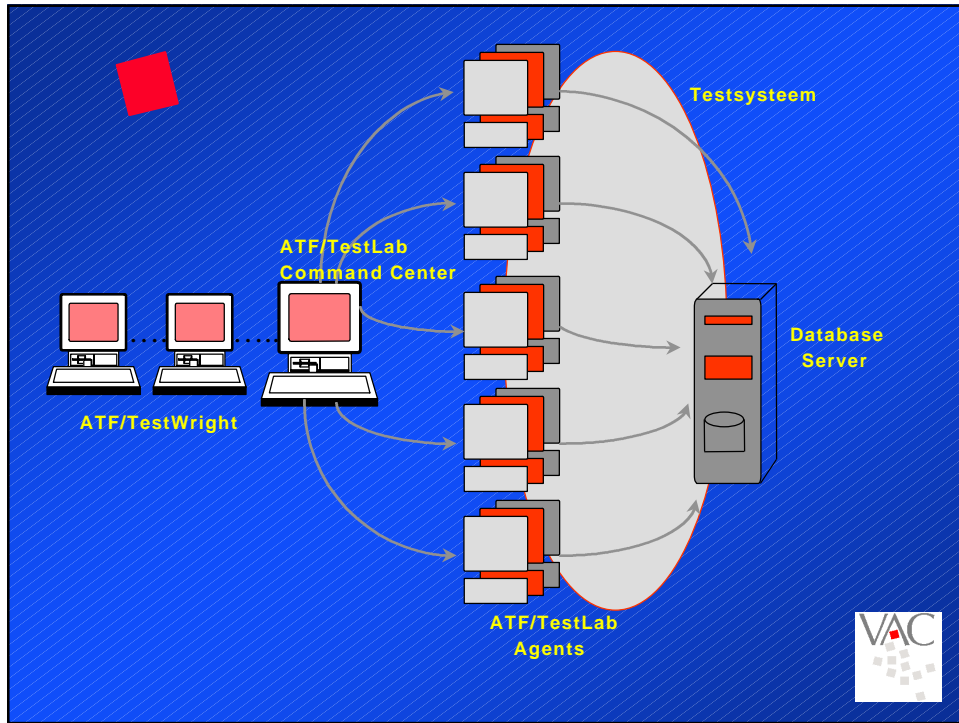


Visions:

- Efficiency improvement by matching the skills of the tester and the test interface
- Quality enhancement by integrating system development process supporting tools







Demo !!!!

The VAC logo is located in the bottom-right corner of the slide.

Current and intended status of system development

Strategy

- Classify the current situation
- Identify the goals
- Determine how to achieve the goals

Guidance

- Use e.g. Capability & Maturity Model (CMM)
- Determine areas of interest and supporting tools



Where are we?

Process status

- Optimized
- Controlled
- Defined
- Repetitive
- Chaotic

Achieved

- 2 in the world
- 3 in Holland
- ??



CMM chaotic => repetitive

Areas of interest

- Configuration management
- Quality Assurance
- Subcontract management
- Project planning
- Project tracking
- Requirements planning

Tools: Version control, Workflow management,



CMM repetitive => defined

Areas of interest

- Reviews
- Coordination between groups
- Software product engineering
- Structured testing
- Integrated software management
- Training programs
- Process definition
- Process focus

Tools: GUI tests, Database tests,





Demo !!!!



Conclusions

- Tailoring of Test Interface enhances acceptance of Test tools
- Data driven approach increases maintainability, and flexibility of testware
- Process Improvement model sets priorities on tooling
- Structured process improvement requires tool integration



WebSite Validation Technology

By

Edward Miller
Software Research, Inc.

Software Research, Inc



WebSite Quality Factors

- Time / Change
- Structure
- Content
- Accuracy and Consistency
- Response Time and Latency
- Performance



WebSite Architectural Aspects

- HTML
- Java, JavaScript
- Cgi-Bin (Perl, etc.)
- Database Access
- Multi-Media



Assuring WebSite Quality Automatically

- **Browser Independent**
- **No Buffering, Caching**
- **Fonts and Preferences**
- **Object Mode**
- **Tables and Forms**
- **Frames**



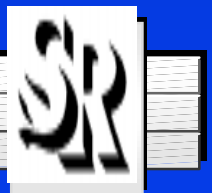
Basic WebSite Validation Steps

- Page Quality
- Table, Form Quality
- Page Relationships
- Performance, Response Time



WebSite Content Validation

- **Identify Segments**
- **Baseline of Content**
 - ★ **Segment Fragment**
 - ★ **Segment**
 - ★ **Multiple Segments**
 - ★ **All Segments**



E-Commerce WebSite Validation

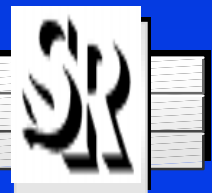
- **Stable Inputs**
- **Database Access**
- **Response Files**
- **Validation Check**
- **Typical Uses**



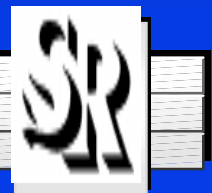
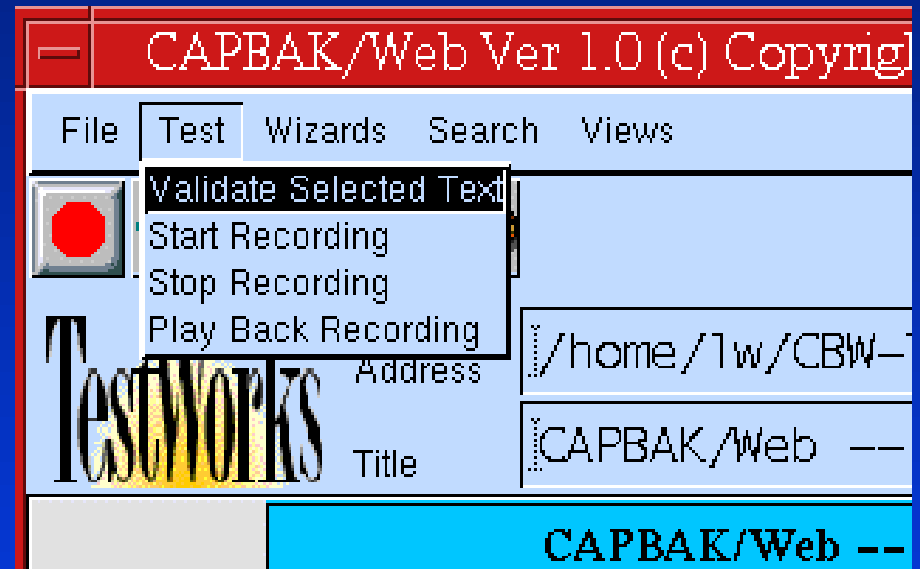
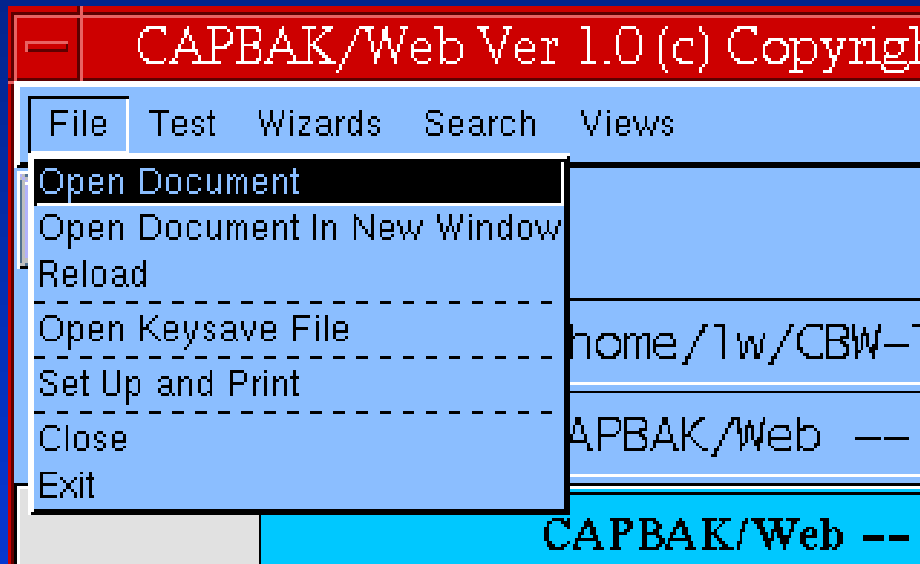
CAPBAK/Web Browser

The screenshot shows the CAPBAK/Web Browser interface. The title bar reads "CAPBAK/Web Ver 1.0 (c) Copyright 1998 SR/Inc. (04/17/98)". The menu bar includes "File", "Test", "Wizards", "Search", "Views", and "Help". Below the menu bar are navigation icons: a red stop button, a blue back arrow, a grey forward arrow, a yellow home button, and an eye icon. The address bar contains the URL "/home/lw/Thot/solaris/bin/help.html" and the title bar shows "TestWorks/Web -- OnLine Help". The main content area features the TestWorks logo and a yellow banner with the text "CAPBAK/Web Ver. 1.0" and "Capture/Replay, Script Creation, Site Validation & Verification, Performance Analysis and Regression Testing of WebSites Copyright 1998 by Software Research, Inc.". Below the banner is a table with a "Click To Learn About Subject" column and a grid of links.

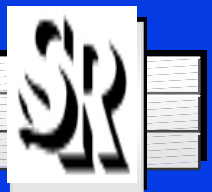
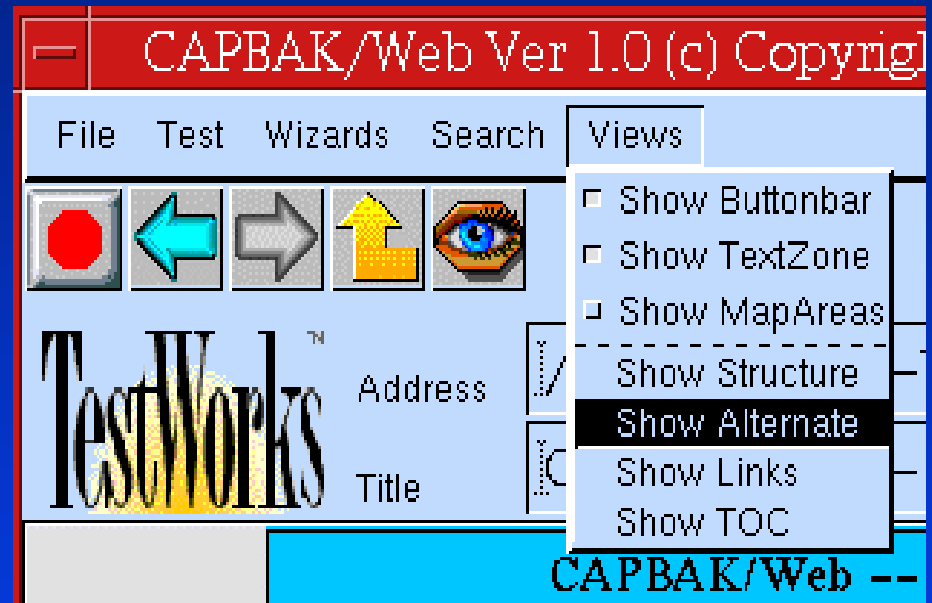
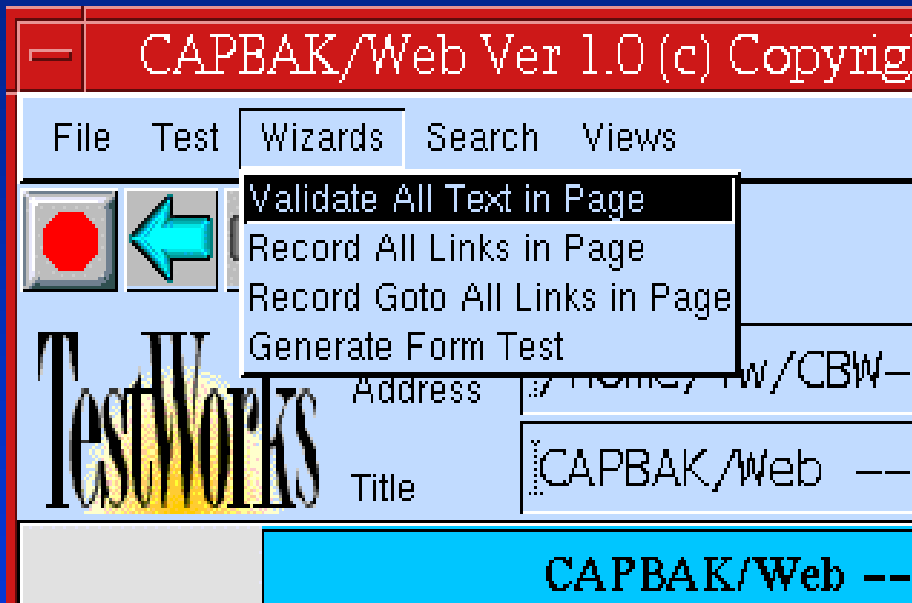
Click To Learn About Subject	FAQ	Terminology	Script Language
	Link-Check Wizard	Link-List Wizard	Form Wizard
	ASCII Browser Display	Link Display	Structure Display
	Command Line Environment Variables	Application Strategies	Reports
Hints On How to Use HELP			



CAPBAK/Web File, Test Pulldowns



CAPBAK/Web Wizards, View Pulldowns



CAPBAK/Web Link Check Wizard Output

```
void default () {  
  
/* Produced by CAPBAK/Web Ver. 1.0 (04/04/98) Record All Links wizard  
(c) Copyright 1998 by Software Research, Inc. */  
  
/* Mon Apr 6 12:37:46 1998 */  
  
WT_GotoLink("/home/lw/CBW-Test-Area/example1/index.html");  
WT_FollowLink(L49,"","#target");  
WT_BackCheck("/home/lw/CBW-Test-Area/example1/index.html");  
WT_FollowLink(L59,"","#notdefined");  
WT_BackCheck("/home/lw/CBW-Test-Area/example1/index.html");  
WT_FollowLink(L67,"","#./example1.out.html");  
WT_BackCheck("/home/lw/CBW-Test-Area/example1/index.html");  
WT_FollowLink(L75,"","#./example1.notoutside.html");  
WT_BackCheck("/home/lw/CBW-Test-Area/example1/index.html");  
  
/* Mon Apr 6 12:37:46 1998 */  
}
```



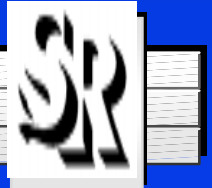
CAPBAK/Web Page Link List Wizard Output

```
void default () {  
  
  /* Produced by CAPBAK/Web Ver. 1.0 (04/04/98) Goto All Links wizard  
  (c) Copyright 1998 by Software Research, Inc. */  
  
  /* Mon Apr 6 12:38:22 1998 */  
  
  WT_GotoLink("/home/lw/CBW-Test-Area/example1/index.html");  
  WT_FollowLink(L49, " ", "#target");  
  WT_GotoLink("/home/lw/CBW-Test-Area/example1/index.html");  
  WT_FollowLink(L59, " ", "#notdefined");  
  WT_GotoLink("/home/lw/CBW-Test-Area/example1/index.html");  
  WT_FollowLink(L67, " ", "./example1.out.html");  
  WT_GotoLink("/home/lw/CBW-Test-Area/example1/index.html");  
  WT_FollowLink(L75, " ", "./example1.notoutside.html");  
  WT_GotoLink("/home/lw/CBW-Test-Area/example1/index.html");  
  
  /* Mon Apr 6 12:38:22 1998 */  
}
```

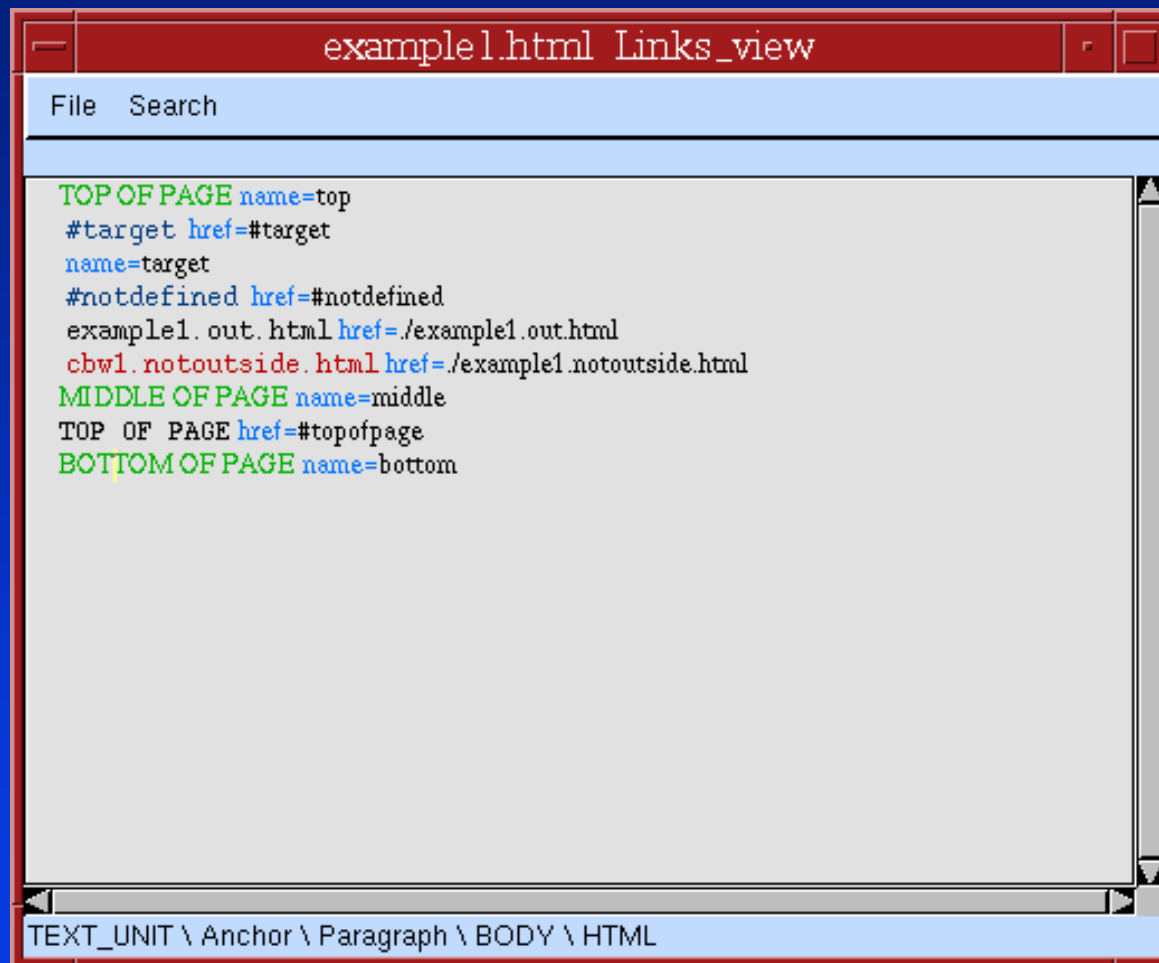


CAPBAK/Web Form Wizard Output

```
void default () {  
  
  /* Produced by CAPBAK/Web Ver. 1.0 (04/04/98) Form wizard  
  (c) Copyright 1998 by Software Research, Inc. */  
  
  /* Mon Apr 6 12:38:39 1998 */  
  
  WT_GotoLink("/home/lw/CBW-Test-Area/example1/index.html");  
  WT_SelectOneRadio(L104,"check","now",FALSE);  
  WT_SelectOneRadio(L110,"check","next",FALSE);  
  WT_SelectOneRadio(L116,"check","look",FALSE);  
  WT_SelectOneRadio(L122,"check","no",FALSE);  
  WT_SelectCheckbox(L134,"concerned","Yes",FALSE);  
  WT_SelectCheckbox(L140,"info","Yes",FALSE);  
  WT_SelectCheckbox(L146,"evaluate","Yes",FALSE);  
  WT_SelectCheckbox(L152,"send","Yes",FALSE);  
  WT_FormTextInput(L162,"TestWorks");  
  WT_FormTextInput(L171,"TestWorks");  
  WT_FormTextInput(L180,"TestWorks");  
  WT_FormTextInput(L189,"TestWorks");  
  WT_SubmitForm(L200,"SUBMIT FORM");  
  WT_ResetForm(L196,"RESET FORM");  
  
  /* Mon Apr 6 12:38:39 1998 */  
}
```



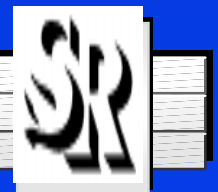
CAPBAK/Web Link Display



The screenshot shows a browser window titled "example1.html Links_view". The main content area displays the following text:

```
TOP OF PAGE name=top
#target href=#target
name=target
#notdefined href=#notdefined
example1.out.html href=./example1.out.html
cbwl.notoutside.html href=./example1.notoutside.html
MIDDLE OF PAGE name=middle
TOP OF PAGE href=#topofpage
BOTTOM OF PAGE name=bottom
```

The status bar at the bottom of the window displays the path: TEXT_UNIT \ Anchor \ Paragraph \ BODY \ HTML.



Netscape: CAPBAK/Web -- Example 1 (example1.html)

File Edit View Go Communicator Help

Back Forward Reload Home Search Guide Print Security Stop

Bookmarks Netsite: http://www.testworks.com/Products/Web/CAPBAK/example1.html

CAPBAK/Web -- Example 1 (example1.html)


This example HTML file is designed to be used with CAPBAK/Web. The goal of this page is: (i) To show of what CAPBAK/Web can do; (ii) to exercise CAPBAK/Web; (iii) to act as a teaching and demonstration device.

Here is an Anchor Link to the document to an anchor named [#target_](#).

Here is a broken Anchor Link inside this document [#notdefined](#) that is NOT defined and thus should fail.

This is an HREF Link to [example1.out.html](#) which exists and but is trivial.
This HREF Link is to [cbwl_notoutside.html](#) which does not exist and should fail.

Here is an image link that exists: [NEW!](#)

Here is an image link one that does not exist: 

Please select options from the FORM shown below:

PLEASE CHECK ONE:

- Buying Now
- Buying Next Month
- Just Looking
- Not Really Interested

INTEREST AREAS:

- Yes I am concerned with my WebSite Quality
- Yes I want more information about CAPBAK/Web
- Yes I might want to evaluate CAPBAK/Web in my environment
- Yes, send me information about CAPBAK/Web by Email. Here is my Name and Email address and Phone Number:

Name:

Phone:

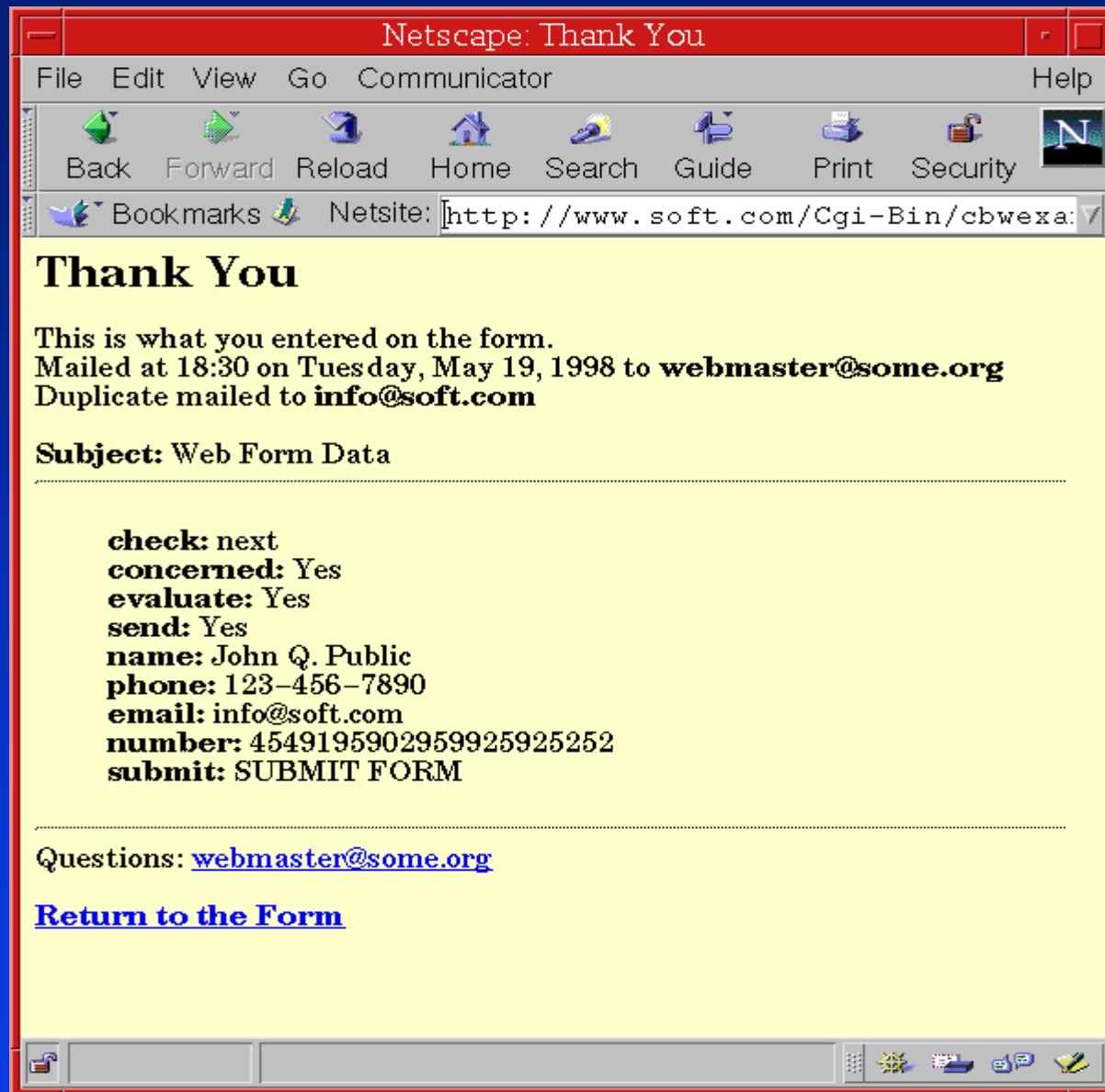
E-Mail:

Serial Number:

This reference connects to the [TOP OF PAGE](#)

100%





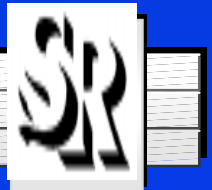
CAPBAK/Web Structure View

```
example1.html Structure_view
File Search
HTML
HEAD
  META http-equiv=Content-Type content=text/html; charset=iso-8859-1
  META name=Author content=web2
  META name=GENERATOR content=Mozilla/4.04 [en] (Win95; I) [Netscape]
  TITLE
    CAPBAK/Web -- Example 1 (example1.html)
BODY
  CENTER
    TABLE border=1 cols=1 width=75% bgcolor=#33CCFF
      TBODY
        TR
          TD
            CENTER
              B
                FONT size=+2
                  CAPBAK/Web -- Example 1
                  BR
                    (
                      TT
                        example1.html
                    )
                P
                  A name=top
                    TOP OF PAGE
                BR
                P
                  This example HTML file is designed to be used with CAPBAK/Web. The goal of this
                P
                  CENTER
                    Here is an Anchor Link to the document to an anchor named
                    A href=#target
                      TT
                        #target
                    BR
                    A name=target
                      Here is a broken Anchor Link inside this document
                    A href=#notdefined
```



CAPBAK/Web Sample TestScript

```
void default () {  
  
  /* Recording by CAPBAK/Web Ver. 1.0 (04/17/98)  
  (c) Copyright 1998 by Software Research, Inc. */  
  
  /* Mon May 18 12:27:34 1998  
  */  
  
  WT_GotoLink("/home/lw/www.soft.com/htdocs/index.html");  
  WT_FollowLink(L58,"Picture_Unit","./contents.html");  
  WT_FollowLink(L52,"Software Research, Inc. HOME PAGE","./index.html");  
  WT_FollowLink(L60,"Picture_Unit","./Corporate/index.html");  
  WT_FollowLink(L290,"Picture_Unit","../QualWeek/index.html");  
  WT_FollowLink(L175,"Picture_Unit","../Distributors/index.html");  
  WT_FollowLink(L592,"Picture_Unit","../Info/index.html");  
  WT_FollowLink(L766,"Picture_Unit","../Products/index.html");  
  WT_FollowLink(L92,"T-SCOPE","./Coverage/tscope.html");  
  WT_FollowLink(L80,"Take a tour of TestWorks for UNIX: TestWorking  
  MotifBurger","../../AppNotes/Motifburger/index.html");  
  WT_FollowLink(L174,"Picture_Unit","fig11.gif");  
  WT_BackCheck("/home/lw/www.soft.com/htdocs/AppNotes/Motifburger/index.html");  
  WT_FollowLink(L372,"Picture_Unit","../../Support/index.html");  
  
  /* Mon May 18 12:29:12 1998  
  */  
}
```



Contact Information:

Software Research, Inc.
625 Third Street
San Francisco, CA 94107 - 1997 USA

Phone: [1] (415) 957 - 1441

FAX: [1] (415) 957 - 0730

Email: sales@soft.com

WebSite: <http://www.soft.com>

Software Research, Inc



SIM GROUP Ltd.

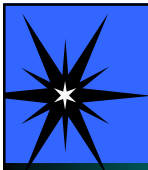


Test Environments on DEMAND

10/14/98

© 1998, SIM Group Ltd.

1



Agenda

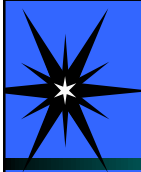
- Types of Software Testing
- Definition of a Test Environment
- What Testing needs from Test Environments
- The **QUEST** Methodology for Test Environments
- Additional Considerations & Benefits

10/14/98

© 1998, SIM Group Ltd.

2





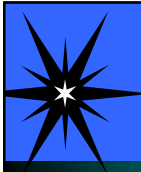
What we all want from testing

- Maximum coverage of test conditions
- Minimum effort and time to test
- A fast and easy way to test
- Minimal problems following release and during live running

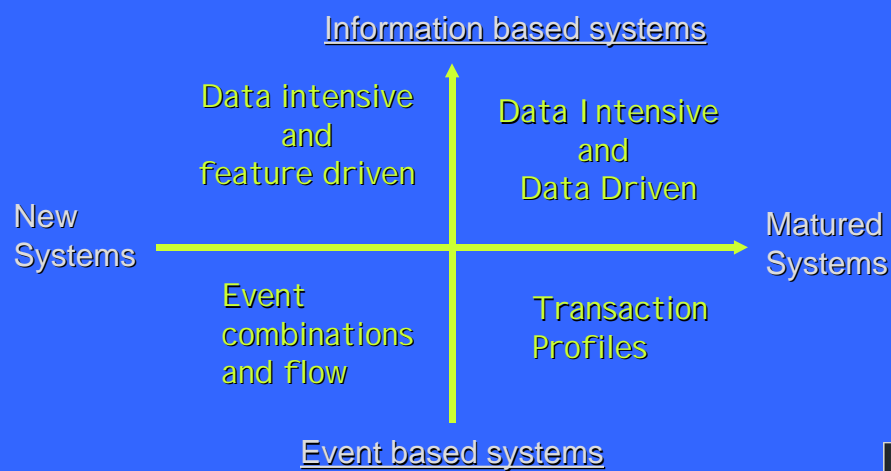
10/14/98

© 1998, SIM Group Ltd.

3



Different systems require different types of testing



10/14/98

© 1998, SIM Group Ltd.

4





Based on experience

- Predicting (and testing) all possible uses of the system
- Thorough regression testing
- Fully understanding the operational context of the system
- Using real test cases makes it easier to spot problems

10/14/98

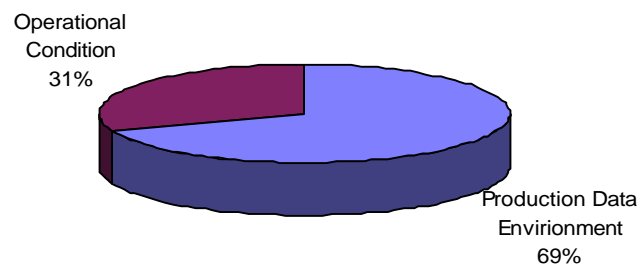
© 1998, SIM Group Ltd.

5



Information based systems ...

Production Fault Analysis

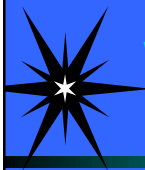


10/14/98

© 1998, SIM Group Ltd.

6





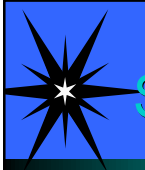
What in the test environment prevents finding these problems?

- Data values and combinations of data
- Technical configuration is different
 - Placement and location of Databases & files
 - Sizes, volumes and performance tuning
- Processing control differences
- Connectivity and interfaces
- Currency of objects

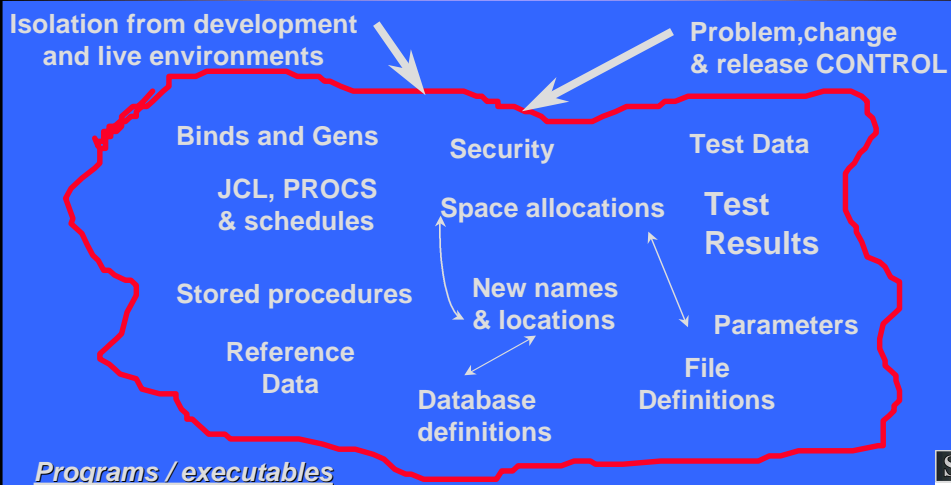
10/14/98

© 1998, SIM Group Ltd.

7



Scope of Test Environments

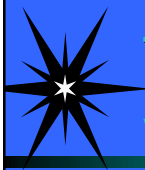


10/14/98

© 1998, SIM Group Ltd.

8





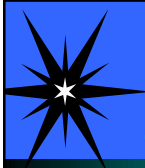
Test Environments - what are they?

- DB meta data
- DB
- Files
- Data
- Control parameters
- Testing infrastructure
 - Tools
 - Test Data
- Process control data
 - JCL
 - Schedules
 - Batch control statements
 - Stored Procedures
 - Process Automation
- Executables

10/14/98

© 1998, SIM Group Ltd.

9



What testing needs from test environments

- Need it now
- Must work straight away
- Contents must be reliable and have integrity
- Optimum Data (low volume and high coverage)
- Special Test Conditions

10/14/98

© 1998, SIM Group Ltd.

10





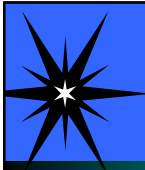
The **QUEST** Methodology for Test Environments

- Responsive to project and testers needs
- Applies to any platform
- Tools and utilities to support the process
- Complements other testing tools
- Enables stability for automated testing
- Saves lengthy set up time

10/14/98

© 1998, SIM Group Ltd.

11



QUEST methodology Step 1

- Hands off analysis and definition of environment contents and object details
 - Files and databases
 - Procedures and process control
 - Sizes and locations
 - Catalog and Directory

10/14/98

© 1998, SIM Group Ltd.

12





QUEST methodology Step 2

- Can create many environments from one master
- Transformation for individual test environment needs
 - Sizing
 - Names
 - File Sharing & separation
 - Location of objects

10/14/98

© 1998, SIM Group Ltd.

13



QUEST Environment maintenance

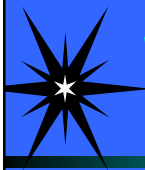
- Ability to edit most fields for objects
- Can size as required
- Locate/place as required
- Can insert, amend, delete
- Share between applications
- Refresh and Rename
- Many fastpath inputs

10/14/98

© 1998, SIM Group Ltd.

14





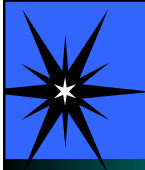
What we've done.... and next steps

- Registered a system
- Built the information tables within **QUEST**
- Transformed the master definition to give the new environment definition
- Used maintenance to customise as required
- Now we are ready to Create the objects
- The 'Populate' steps come after....

10/14/98

© 1998, SIM Group Ltd.

15



QUEST Creation Step 3

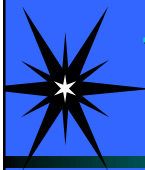
- All file entries
- Database definitions
- Database objects
- Test procedures and processes
- Batch control
- Generation procedures submitted

10/14/98

© 1998, SIM Group Ltd.

16





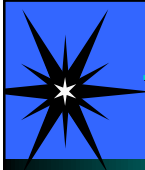
Test Data (The **QUEST** way) Step 4

- Analyse testing requirements and define test conditions required to extract
- Identify records to extract
- Define relationships for referential integrity
- Extract all data
- Transform data
- Populate in data in Test Environment

10/14/98

© 1998, SIM Group Ltd.

17



Transforming Test Data

- Scrambling of Test Data
 - De-personalisation of customer details
 - Protecting sensitive information
 - Global changes for testing requirements
- Customisation of test cases / conditions
- Date Advancement
 - Move dates forward to retain date integrity
- Currency conversion for

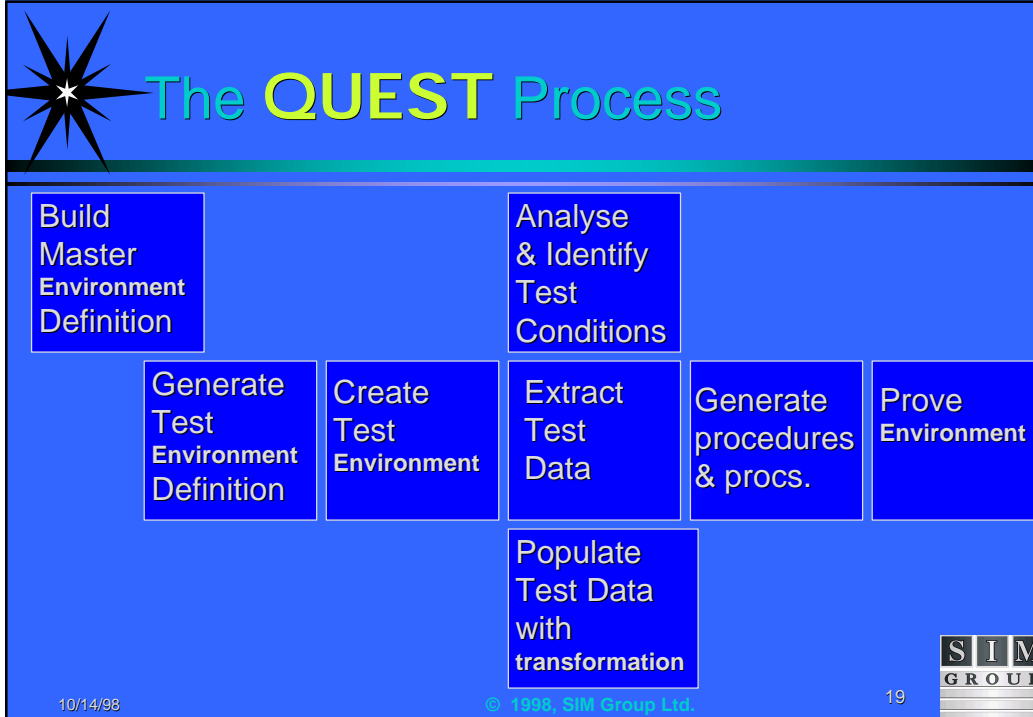


10/14/98

© 1998, SIM Group Ltd.

18

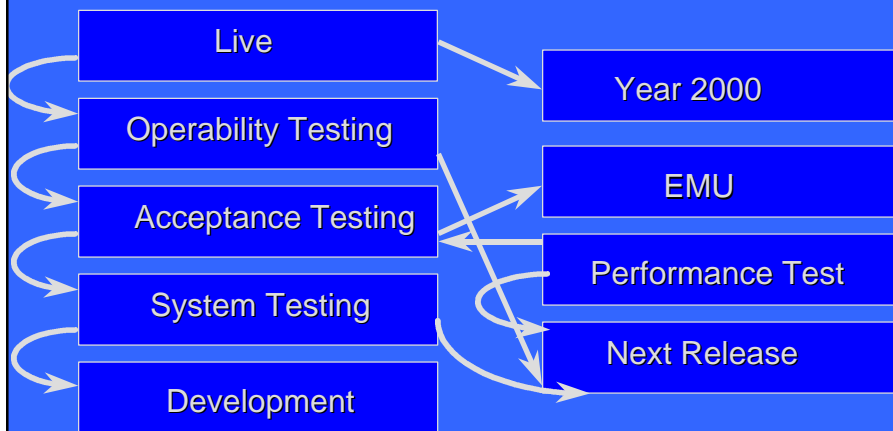




-
- Additional Benefits of QUEST**
- Maintenance of environments
 - Management
 - Control
 - Auditability
 - Efficient utilisation of resources
 - Test Assets
- 10/14/98 © 1998, SIM Group Ltd. 20
-



Managing Environments





Test Environment Services

1. Overview

Background

Successful and effective testing requires reliable test environments. If the testing environment is up to date and the coverage of data is correct then the impact of the environment itself on the testing process is reduced.

Test environments can be created in a variety of ways, for example:

- a full copy of production environment(s)
- sizing a production environment down by careful use of selection criteria
- starting from scratch.

SIM have a wide experience in this field and can assist you to choose the most suitable method for your particular environment, regardless of the platform.

Problems with Test Environments

Test environments are generally unreliable for a number of reasons;

- Set up takes a long time and is accorded a low priority by technicians;
- Set up can take longer than the actual testing;
- Maintenance is difficult, expensive and time-consuming;
- It is often easier to do a little testing and implement;
- Management of test environments is usually minimal;
- They become out of date and the data becomes inconsistent;
- Coverage of data is often insufficient to the objectives of the testing;
- System 'experts' move on and a knowledge gap is created;
- Up to 65% of recorded production problems can be attributed to 'poor' testing.

SIM can help you to resolve these issues with the appropriate elements of our package of services devoted to the creation of test environments.



Test Environment requirements

The main requirement of testing is that as much of the code as possible is exercised. Since data is the basis of any good test environment the most important factor is that the coverage of data is sufficient to the needs of the testing being undertaken. If, along with good coverage, the test data is of manageable volumes and retains referential integrity, the subsequent testing results will be much more reliable. It would be better still if the test data could be made available quickly, is easily maintained or refreshed, can be repeated as necessary and is portable between different systems.

The main benefit of having robust test environments is that testing becomes more complete, more flexible and more responsive to business needs. Almost inevitably, because of this flexibility, even more testing is possible.

SIM can assist in the provision of robust test environments dependant on your needs. We are able to advise on, define and create the processes necessary to provide the data.

Test Data Solutions

The test data that constitutes your test environment can, depending on the requirements, be provided either by taking copies of existing data or by extracting coherent sets of data. Data is best collected in natural portions – that is relating to an identifier (region, branch, product type, etc.) and it should, where possible, relate to specific test conditions. This meets both the requirement to get best possible coverage and the need for manageable volumes of data. Data collection processes must be responsive to project and testing schedules, may need to be invoked many times during a project and will certainly need to be both controlled and maintained.

SIM will be able to advise on, define and create reusable processes for data collection and management which are, where possible, generic. These processes will be available for reuse with the test environment for which they are created and may, because they are generic, be applicable to other test environments.

Test environments – an asset

Good test environments lead to better, more complete and more flexible testing. These test environments and the processes used to create them will become key components of your overall testing strategy. As such they should be treated as assets in the same way that any business software and data would be.

SIM can advise on management of test assets and also provide tools to assist in this process.



2. Objectives of the SIM Group

The SIM Group specialise in software testing and uses a flexible approach to assist customers to solve testing issues effectively. Our consultants and technical staff will work in partnership with your IT professionals, in the most appropriate and effective way, to provide test environment creation processes.

We are committed to the success of your test environment project and our wide range of experience and skills enable us to use 'best practice' to ensure that success.

A range of other consultancy and testing skills are also available.

3. Methodology

SIM will achieve the above by means of our proven methodology which includes the following;

- Develop an understanding of the data requirements for each test environment by means of discussions, workshops, reference to users, developers and support staff and by the analysis of the available documentation, objects and data.
- Agree with all interested parties the scope of the test environment – including objects and data requirements.
- Create the data objects using the most appropriate method.
- Populate the data objects with appropriate levels of data using the most appropriate method.
- Develop processes and procedures to support the test environments as test assets.
- Provide access to the QUEST toolset which will automate many of the more repetitive tasks involved in the creation of mainframe test environments.

4. Deliverables

SIM methods and processes provide;

- Test Environment Analysis documentation
- Best methods for Object Creation and Data Provision
- Automation of Object Creation and Data Provision
- Test Asset Processes & Procedures.



5. Benefits

The SIM Test Environment Creation service will benefit your company by providing;

- A comprehensive documented understanding of how your systems are built.
- A known and agreed basis for the creation of test environments.
- A defined and documented process for the creation of test environment objects.
- A defined and documented process for the population of test environment objects.
- Automation applied to the appropriate processes.
- The appropriate tools to manage test environment(s).
- Transfer of the necessary skills to ensure that test environment(s) are managed and maintained.

Are you “Euro-Ready”?

Mercury Interactive’s solutions for EMU
conversion projects



Ido Sarig
Director, European Marketing
Mercury Interactive Corp.

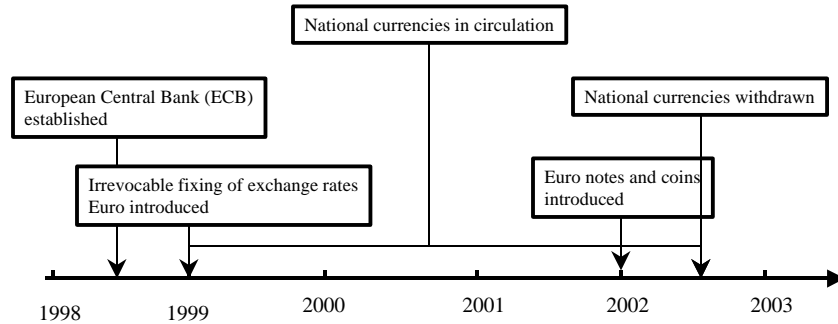


What is the Euro?

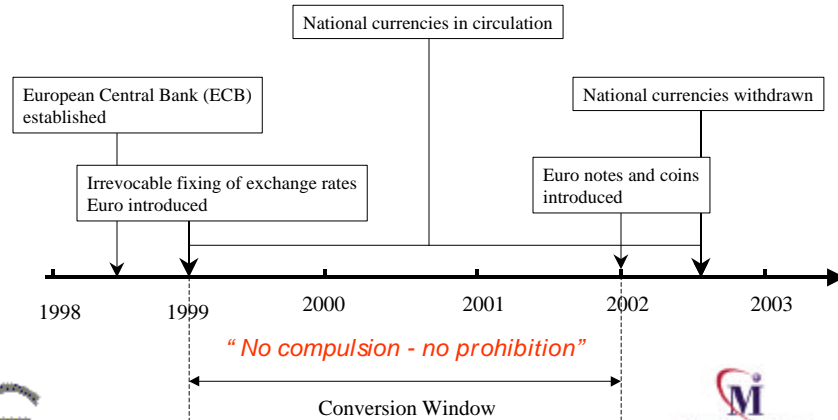
- ◆ Europe’s new single currency - will replace national currencies by 2002
- ◆ Will create the world’s 2nd largest economy
- ◆ Will Make Europe more competitive, more efficient



The Euro Timeline



The Euro Timeline



The Euro

- ◆ Euro is not just a math problem – it is a business problem
- ◆ Business drivers
 - New competition, changed businesses
 - New markets, new products, and services
 - New business processes to leverage new opportunities
- ◆ EMU projects are more difficult than Y2K conversions
 - Money is harder to find than dates
 - Automation of conversion is more difficult - few tools available
 - Testing is more complex
- ◆ Most projects will be done twice or three times by 2002!!



IT Impact of Converting to Euro

- ◆ Business Issues:
 - “Euro” can be a competitive differentiator
- ◆ Making applications “Euro-Ready”
 - **Triangulation** - implement algorithm to convert national currency 1 to national currency 2 via the Euro, with 6-digit precision
 - **Rounding problems** - account for rounding differences in systems that track orders through amount-matching
 - **Decimalization** - add support for decimal point, in Spain, Italy, Belgium
 - **Thresholds** - What is Ff 99.95 in Euro?
 - **Dual Display** - re-arrange screen layout, reports
- ◆ Phase dependant business logic



Just another currency?

- ◆ The Euro's definitely **not** just another currency:
 - Use special algorithms to convert to/from Euro
 - Direct or cross rates are not allowed between 2 National currencies
 - The required precision is more than most financial systems support today.
 - Need to maintain books and report in Euros



Scale of the Euro Problem

- ◆ The euro will cost corporations two to six times more than will Y2K conversions
 - Andrew Dailey, Gartner Group
- ◆ It will cost software developers about \$300 billion to convert their software
 - Dennis Keeling, Ovum
- ◆ IT departments are already strained for resources due to Y2K problem



Introducing TestSuite Euro



TestSuite Euro: - Single testing solution for all Euro projects (legacy and C/S)

TestDirector *Defines and Organizes the testing process*

WinRunner Euro *Powerful regression testing tool enhanced for Euro testing*



Quick and easy test data generation tool



TestDirector

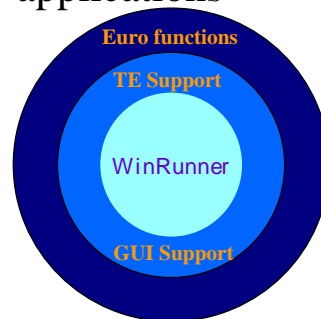
- ◆ A well defined test plan is a must - before testing begins
- ◆ Begin with test planning using TestDirector NOW
- ◆ Use TestDirector to prioritize tasks



WinRunner Euro: Automated Testing of Converted Code

Enhanced version of WinRunner accelerates development of “Euro-Ready” applications

- ★ WinRunner
 - Automatically records test baseline
 - Supports all common Terminal Emulators and GUI development environments
- ★ WinRunner Euro Adds:
 - Automatic verification for Triangulation results
 - Support for phase-dependant conversion rules



Verifying Triangulation

Old Application: Airtours		
Ticket Price:	Ff	1,200
Total:	Ff	2,400

New Application: Airtours		
Ticket Price:	DM 356.70	EUR 181.08
Total:	DM 713.40	EUR 362.16

Conversion Table	
Ff:	6.626901
DM:	1.969848

Record WinRunner Script on existing application

Run WinRunner Script on new application

WinRunner verifies correct implementation of triangulation



WinRunner Euro

- ◆ Leverage a single testing solution across the enterprise
 - Euro, Y2K, C/S, Web, LR
 - Provides lowest cost of ownership
- ◆ Only tool with specific Euro functionality
- ◆ Use it for Euro today - leverage for the future



TestBytes - Eliminates production data requirements

Sensitive Production Data
(Names, Bank Accounts)



TestBytes



Realistic, unclassified
Test Data

- ◆ Rapidly generate realistic test data without accessing sensitive production data
 - Solves Privacy & Security issues
 - Allows outsourcing of Euro testing to 3rd party



Summary



- ◆ January 1, 1999 is a non-negotiable date
 - Euro conversion **MUST** be completed by then!
- ◆ 50%-80% of the effort will be spent on testing
- ◆ Integrating automated testing tools into the process saves scarce human resources, as well as time and money
- ◆ TestSuite Euro - the only comprehensive testing solution

This is the difference between success and failure!



Demo



EMU Conversion – Test Reality Before Reality Tests You...

Ido Sarig
Director of European Marketing
Mercury Interactive Corp
2 Hayotzrim
Or Yehuda 60218
Israel
Tel: +972 3 538 8846
Fax: +972 3 533 1617
Email: ido@mercury.co.il

Readying information systems for the EMU entails not only the conversion of a large number of applications in a limited amount of time but also the verification that the systems still function properly. Software testing is therefore a vital part of this conversion effort, which is at least as momentous as the Year 2000 project and incorporates a wider range of issues. This paper discusses how to employ automated testing tools to ensure thorough testing, thus avoiding potentially serious financial pitfalls.

Introduction

The introduction of a single European currency, the euro, demands more from IT departments than simply adding new currency fields and ensuring that currency conversion is performed correctly. Failure to tackle all the issues involved, such as triangulation, rounding differences and threshold value conversion, can result in significant problems when the new currency arrives. Thus, as with Year 2000 compliance, euro-converted information systems must be rigorously tested to ensure that they function at least as well as they did before.

As with the millennium project, which has already strained the resources of IT departments, the tight deadlines and the scale of euro conversion rule out the option of manual testing. Automated testing tools have already demonstrated their value to Year 2000 testing teams, enabling them to meet the deadline while ensuring that business processes will operate correctly when the millennium arrives. Coupled with a well-designed testing methodology, euro-specific automated tools save scarce human resources and significantly accelerate the testing process, as well as minimizing the number of new errors unintentionally introduced during conversion.

EMU-Specific IT Issues

Triangulation, Rounding Problems and Data Pollution

During the transition period, organizations will be receiving and producing financial information in both the national currency units and the euro. They may find themselves in the situation where, because not all systems have been switched to the euro, information systems working in national currency units will have to communicate with systems working in euro units.

The EC is requiring that currency conversion, which will most frequently be between the national currency and the euro, is carried out to six-digit precision, using an algorithm more sophisticated than those found in most financial systems. Conversion between national currencies, which will occur, though less frequently, must be calculated in a triangular fashion, via the euro, stipulates the EC.

Conversion calculated to such precision unavoidably creates rounding differences, which, if ignored, can cause financial systems serious problems. Many systems have a built-in capability to match transactions according to their amounts or to crosscheck to ensure calculations have been performed correctly. Even minute differences, with barely any financial significance, will result in a non-matching of amounts. Testing tools identify rounding differences and validate that modification measures, such as truncation, taken to avoid such problems, have been applied consistently throughout the application. They also verify that information systems are not suffering from data pollution, when euro amounts are accidentally combined with amounts in national currency units.

Threshold Values

Threshold or limiting values are used in many organizations to define the actions of a system. For example, junior employees may only be empowered to perform transactions up to a certain amount. If these threshold values are not converted to euros then this can have the undesirable result of junior employees authorizing transactions up to EUR 10,000 instead of FF 10,000. Simple conversion of threshold amounts to euros may result in a figure such as EUR 457.90, and it may be more appropriate to change this to EUR 460.00, in order to create an amount that's simpler to remember. This is also relevant for the issue of 'psychological pricing': Conversion may change a FF99.95 price into EUR 6.73, and businesses may wish to change this to EUR 6.95, in line with in-house pricing policy. If, as is often the case, the threshold value is hard-coded and scattered throughout the system rather than kept in a separate file, extensive searching and thorough testing are required to make sure all such values have been properly replaced.

Decimalization

In countries with low-value currency units, such as Spain, Italy and Belgium, financial systems have no provision for decimal points in monetary units. The euro is subdivided into 100 cents, and thus another task for the testing tools is to verify that in these countries, data types have been modified so that they are capable of holding floating point amounts and displays have been converted accordingly.

Regression Testing

As well as the euro-specific issues mentioned above, more generally, it is essential to test that the new applications work as well as their predecessors, and that no new errors have been introduced during the conversion process. Thus, the converted applications should be subjected to all the tests satisfied by the old versions, with test data updated to incorporate the new currency formats. Since the introduction of the euro is imminent, it is too late to wait to begin the testing process until all currency-related code has been modified. The most effective test plan is to carry out testing concurrent with assessment, code inspection and conversion.

Record and Capture of Business Processes

Using an automated testing tool, which will support most common Terminal Emulators and GUI development environments, organizations can use a simple record-and-capture paradigm to record their existing business processes using local currencies to form a test baseline —at the same time as their software engineers convert the system. The record and capture process, which can be carried out by non-skilled personnel, simply records the actions of real business users as they are using the system. The recordings are then automatically turned into test cases or scripts and stored in a repository. Thus, all the test scripts necessary can be created *now*, without the need to wait until the system has been euro-converted, saving valuable time. Once conversion is complete, these test scripts will be used for testing the converted application.

Excluding Fields and Filtering

Once the system conversion is concluded and, simultaneously, all of the critical business process have been recorded and test scripts created, the testing can begin. Using an object-based testing tool greatly accelerates the testing process. This is due to the fact that object-based testing tools view the graphical user interface (GUI) as a set of labeled fields that contain data that may or may not be changed. Through the automatic creation of an intelligent object map of the screen which captures the logical purpose of different fields, such tools understand logically what a currency field is.

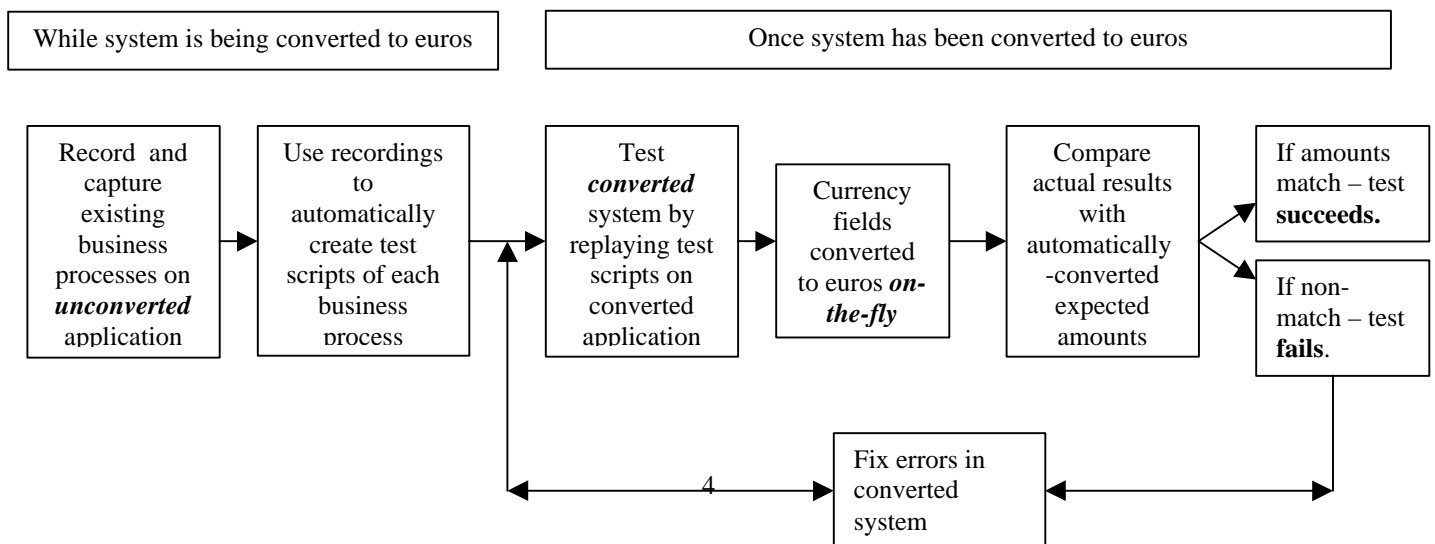
However, by default, the testing tool assumes that all fields with numerical values — except for date fields, which are recognizable by their format — are currency fields. Thus, at the outset the tester must specify which fields are not in fact currency fields but are, say, catalog numbers which should be excluded. The tester can also, for example, stipulate that any number under a particular threshold value, say 100, should not be treated as currency, or even filter an entire area of a screen out of the test.

Once the non-currency fields have been excluded, the actual updating of the test script to convert currency fields to euro amounts is done on the fly during the test itself, through an automatic find-and-replace, saving business users and testers precious time.

Running the Test

The test begins with the developer highlighting the currency fields that require verification. The developer also specifies the original base currency, say the French franc, and the new base currency, i.e. the euro. The test script is then run on the converted business application to simulate the actions of a real user. During the execution of the test, the testing tool verifies that all relevant amounts have been converted accurately and that the expected amounts are returned in the correct currency. A discrepancy between the expected and actual amounts results in test failure. In the report generated after the test is completed, the testing tool identifies where exactly the test broke down, enabling developers to easily see and fix errors.

Diagram 1: Automated Euro Testing Process



Screen Layout Modification

One obvious euro conversion issue requiring testing is change to the layout of computer screens, whether within organizations or on ATMs, cash registers and other money-handling equipment. Screens will need to be modified twice: once to add euro fields alongside national currency fields, and a second time, at the end of the transition period, to remove national currency fields (as is shown in figure 2). Adapting to changes in screen layout poses no problem for object-based testing tools. With the intelligent object map of the screen which has been created, such tools can automatically account for currency changes even if the screen location of one or more fields may have changed. These object-based features eliminate the need for test script re-work, and enable the reuse of the same script recorded on the original application. As a result, they decrease the burden on the tester and improve test accuracy.

Diagram 2: Using GUI Map to Cope with Screen Layout Modification

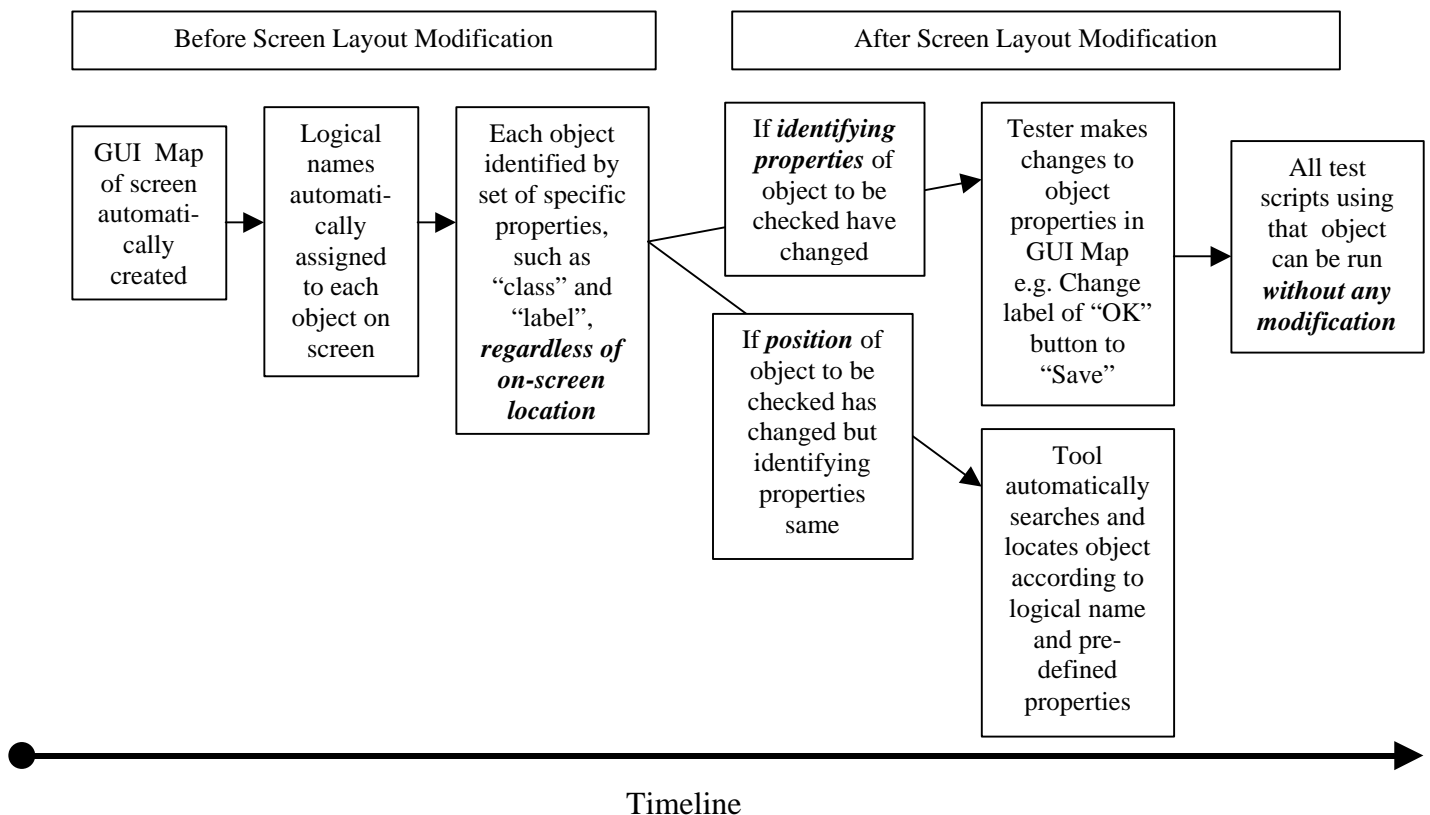


Diagram 3: Runtime Identification of Object in Test Script Using GUI Map

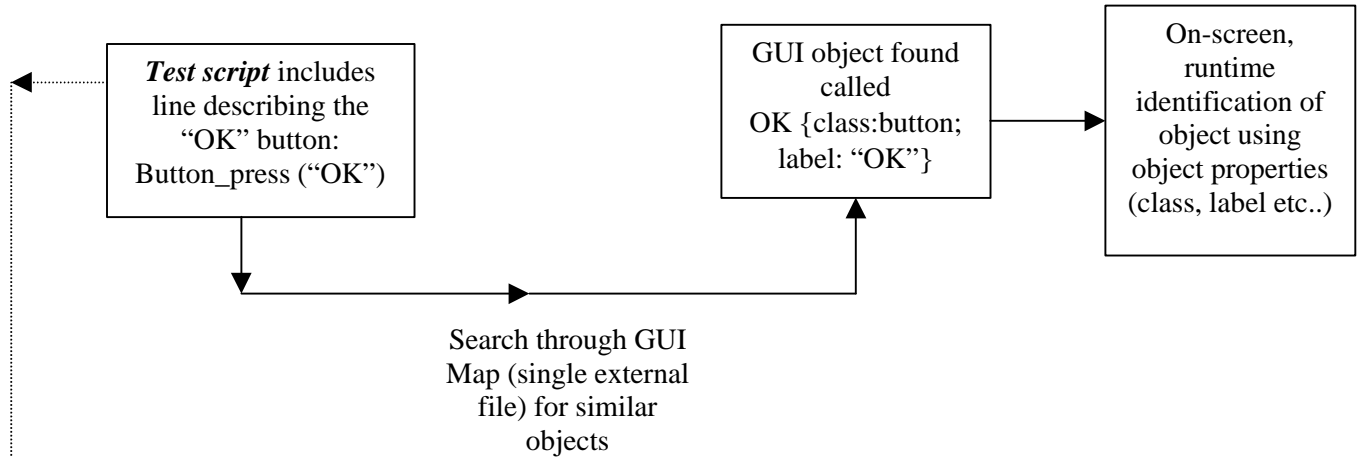


Figure 1: Test Script Showing Flight Application Business Process

```

    win_activate ["Flight Reservation"];
    set_window ["Flight Reservation", 10];
    edit_set ["Date of Flight:", "02/27/98"];
    list_select_item ["Fly From:", "San Francisco"]; # Item Number 3;
    set_window ["Flight Reservation_1", 10];
    button_press ["OK"];
    set_window ["Flight Reservation", 10];
    edit_set ["Date of Flight:", "02/27/99"];
    list_select_item ["Fly From:", "San Francisco"]; # Item Number 3;
    list_select_item ["Fly To:", "Los Angeles"]; # Item Number 1;
    edit_set ["Name:", "jonathan"];
    obj_mouse_click ["FLIGHT", 45, 27, LEFT];
    set_window ["Flights Table", 10];
    list_select_item ["Flight", "2004 SFO 10:33 AM LAX 11:17 AM USA $121.60"]; # Item Nu
    -> button_press ["OK"];
    set_window ["Flight Reservation", 10];
    edit_set ["Name:", "jonathan"];
    obj_type ["Name:", "<kReturn>"];
    win_move ["Flight Reservation", 240, 52];
    set_window ["Flight Reservation", 10];
    obj_mouse_click ["Button", 14, 6, LEFT];
    edit_set ["Date of Flight:", "02/27/99"];
    list_select_item ["Fly From:", "Los Angeles"]; # Item Number 1;
  
```

Figure 2: Screens Before and After Euro Conversion

Old Application: Airtours		
Ticket Price:		Ff 1,200
Total:		Ff 2,400

New Application: Airtours			
Ticket Price:	DM 356.70	EUR 181.08	
Total:	DM 713.40	EUR 362.16	

Data Driven Testing

Because state-of-the-art testing tools understand the data entered by the users and the data returned by the application in response, they can perform data-driven testing. Testers record a business process just once and then create a test script, parameterizing currency fields and highlighting those that require verification. The test can then be played back with a wide variety of data from an external data file. In the case of the euro, the data could vary in terms of both currencies and prices, reflecting the real-life actions of many users.

As an example of such a scenario, suppose an organization is converting a flight reservation system. When the user input data is captured from the pre-converted application, the ticket price field may contain “BEF 10,000” for a particular flight. To test converted code, testers may want to execute a series of tests for a wide variety of flights in different price ranges. This data is stored in an external data file and used by the testing tool during playback of the test script. The euro testing tool automatically verifies that conversion and triangulation has been correctly implemented between expected amounts in various currencies.

This process can be further improved by having the external data file generated automatically from existing data in the database. Tools that perform such automatic data generation connect directly to the database or test data files and can generate an unlimited number of rows of data with a structure identical to the original data. This is especially useful when the data involved is of a sensitive nature, such as that processed by banks. Many banks are, in fact, prohibited from outsourcing their testing efforts with actual test data containing genuine account numbers and names. An automatic test data generation tool solves this issue by creating realistic “pseudo-data” based on the real data but with the real information scrambled. Such pseudo-data generating tools have become even more useful to euro testing teams following the EU regulations that came into effect in October 1998 (EU Directive 95/46/EC) placing stricter controls on personal data.

Phase Dependent Business Logic

One method which organizations may decide to adopt to deal with the two phases of the EMU is to implement phase dependent business logic into their systems so that only one round of conversion and testing is required. Phase dependent conversion means that applications will automatically “know” whether to deal in euros and national currencies or solely in euros

depending on the date of the transaction. Screen layouts, printed reports and other documents will be altered accordingly. This is another feature of the system that necessitates rigorous testing. Testing tools which support such phase dependent logic enable testers to enter data with different dates in order to verify that the application is responding correctly.

Test Management

A vital part of any software testing project is a well-structured test plan, which takes into account deadlines, number of test developers, hardware requirements and other issues. This is especially relevant for euro conversion, which, as with the Year 2000 project, must meet strict deadlines. Thus the integration of a testing tool and a project management tool can assist in the smooth execution of tests and analysis of results.

Test script archiving and off-hour testing

First, the test management tool archives all test scripts in a central repository so that every test developer has access to every test case and the corresponding test results. Second, during the conversion effort, there will inevitably be contention over hardware, since the production system, software development and software testing all run on the same hardware. In addition, the euro conversion project may be competing for resources with a Year 2000 conversion project.

The ability to run automated tests during off hours can help eliminate competition. Intelligent test management tools can queue tests to run during low access periods, such as midnight to 5 a.m.. The tests can be scheduled to run concurrently or sequentially, and on different machines, if necessary. Results are automatically saved in the test management database, with any errors flagged so that developers can quickly scan the results the following morning. Defect reports can be created to identify within the business process the exact location of failure of the test.

Conclusion

As demonstrated above, preparing information systems for the euro is a complex task, at the very least equal in size to the Year 2000 conversion effort. To tackle euro conversion manually would drain both human and financial resources and would, in all likelihood, result in failure to meet the impending deadlines. Automated testing is not longer an unfamiliar concept to IT departments, many of which are using such tools to avoid millennium-related catastrophes. Euro testing tools not only take into account the current shortage of skilled personnel due to millennium projects, but speed up the euro testing process, allowing for the creation of test scripts while information systems are still being converted, and help to avoid the introduction of new errors. At this late stage, euro-specific automated testing tools are really the only choice.

THE EURO CONVERSION

MYTH VERSUS REALITY!

A Special Panel Session

Quality Week Europe

Conference Day #2

12 November 1998 @ 1600

Chaired by Thomas Drake, Coastal Research & Technology, Inc.

Panelists

John Corden, Cyrano
Patrick O'Beirne, Systems Modelling Ltd., Consultant
Jens Pas, ps_testware
Graham Titterington, Ovum

The Euro Conversion – Myth vs. Reality!

This special panel session is designed to provide a forum for discussing the Euro conversion with a decided focus on the technical, economic, cultural, and liability concerns posed by the Euro conversion. Questions that will be presented and discussed include the following:

- What are the real-world challenges and experiences of those currently working the Euro problem?
- Who and what will be impacted by the Euro conversion?
- What are the facts about the Euro conversion versus what is myth?
- Is the Euro conversion the ultimate millennium challenge for Europe?
- Are the business and technical challenges posed by the Euro conversion more difficult to resolve from those surrounding the Year 2000 problem and what difference, if any, can quality make?

The Euro conversion would appear to pose not only economic challenges but also a number of technical and business obstacles and issues, and perhaps most importantly a degree of risk involving historic proportions. The core period for the Euro conversion is marked by transition over a three-year period beginning on 1 January 1999 and the numerous technical and managerial challenges posed by the Euro conversion appear very similar to the Year 2000 problem.

There are a number of common issues relating to applications and package software, source code, service vendors, testing, program management, information technology resource support, domain expertise, and time! But what is unique about the Euro conversion? Perhaps the most critical aspect of the conversion lies in the area of requirements management.

Every business process appears related to the Euro issue and involves far more than just a technical upgrade and modification of existing business and enterprise systems. More importantly are the strategic decisions that effect how businesses conduct their enterprises each and every day. The Euro poses perhaps the greatest challenge in these areas and perhaps the greatest impact of the Euro conversion will occur at the retail level in the buying and selling of goods and services each and every day.

There are also technical risks and obstacles associated with the Euro conversion. Consider the problem of possible data pollution and corruption, conversion errors, and display problems. The data migration paths posed by the Euro conversion encompass a whole host of challenges for testing, quality assurance, and configuration management including database conversions, numeric translations, and modified pricing structures.

And what is the definition of Euro conversion compliance and how does one test for it? Some are even saying that the Euro conversion is actually more complex and has greater impact than even the Year 2000 problem. Why? It will implicitly shift and even perhaps radically alter the day to day lives of the people affected and have a major and lasting impact on all the business and institutions who invest and trade and engage in daily economic transactions within Europe and from without Europe vis-à-vis Europe.

The primary intent of this panel session is having a facilitated discussion among the panel members and the audience on the impact, change, and reality posed by the Euro conversion and explore the question of what does it all mean from a technical, managerial, and information technology perspective.

Reducing the Risk in Information Systems Implementation of the Euro through the use of Automated Software Testing Tools

By John Corden

Introduction

As part of European Monetary Union, the new Euro currency unit will be introduced on 1st January 1999. This presents a substantial challenge for all organizations who trade in or into Europe and are dependent on Information Systems (IS). Business applications will need major modification to handle the Euro. Major changes inevitably bring substantial risk, and organizations will need to trade off business benefit against this risk when determining their implementation schedule. This White Paper outlines how automated software testing tools can be used to minimize risk in Euro IS implementation.

Note: all conversion rates quoted are for illustrative purposes only and may not necessarily represent a realistic estimate of the actual conversion rates to be set on 1st January 1999.

Business Benefits

It is not the purpose of this White Paper to discuss the merits of adopting the Euro. However, it is essential to acknowledge that there are substantial benefits for organizations based within, or trading extensively with, the eleven Participating Countries¹, to adopt the Euro as soon as possible. As a result, suppliers to organizations who unilaterally adopt the Euro could come under considerable pressure to contract and trade in the Euro, even though they are not legally obliged to² and may not even reside in a participating country. So failure to adopt the Euro might have a negative effect on competitiveness.

Transitional Period

The technical challenge, and hence the risk, occurs not so much from the introduction of the Euro, but from the transitional period, which ends on 31st December 2001. During the transition, all organizations that wish to operate with the Euro will need to run dual currencies, i.e. their National Currency Unit (NCU) and the Euro. The main point to remember is that the Euro is the Base Currency while the NCU is simply an alternative measure of the Base Currency at a fixed conversion rate, just as pounds and kilograms are alternative measures of weight with a constant conversion between the two.

Thus a French organization (i.e. an organization in a participating country) could sell at either Euro or Franc prices, but the values must be **exactly** equivalent according to the fixed Euro/FF conversion rate. This situation is comparatively straightforward, because the format of the two values is the same. However, in countries like Italy and Spain, where NCU values are generally whole numbers, the systems will not only need to be enhanced to handle dual currencies, but will also need to accept, store and output decimal values.

Article 235 of the Maastricht Treaty

Organizations in participating countries **must** apply the Article 235 Regulations of the EC Treaty which:

- Specify that conversion rates will be expressed as one Euro = n NCU to six significant figures (e.g. Euro/£ = 0.775735 or Euro/French Franc = 7.36945)
- Prohibit inverse rates (e.g. £/Euro = 1.28910 is **not** allowed)
- Specify that for conversions into the Euro, calculations should be rounded up or down to the nearest Cent
- Specify that for conversions into NCU calculations should be rounded up or down to the nearest unit or sub-unit (e.g. for French Francs, round to the nearest Centime; for Italian Lira, round to the nearest Lira.)
- Specify that for values of 0.5, calculations should be rounded up

¹ Austria, Belgium, the Netherlands, Finland, France, Germany, Ireland, Italy, Luxembourg, Portugal, Spain

² Article 109L(4), Regulation 1: the principle of no compulsion, no prohibition

The problem is even greater if an organization wants to convert one participating NCU into another (e.g. Deutsche Marks (DM) into French Francs (FF)). Because Article 235 mandates use of "Triangulation", this means that it is necessary to convert DM into Euros first, then Euros into FF.

E.g. if the Euro/DM conversion rate is 1.93933 and the Euro/FF conversion rate is 7.36945 then to convert DM1,000 to FF the calculation is:

$$\begin{aligned} \text{DM1,000} \div 1.93933 &= \text{EU515.64} \\ \text{EU515.64} \times 7.36945 &= \text{FF3,799.98} \end{aligned}$$

The Information Systems Impact

Consider the following quotation in French Francs (FF), issued by one French company to another.

FF.Quotation

Quantity Description	FF.Unit Price	FF.Net Price
100 Ring Binder	9.90	990.00
100 500 Sheets A4 copier paper	29.50	2950.00
FF.Total		3940.00

If the company wished to issue the same quotation in Euros (EU), according to Article 235 the FF.Unit Price, FF.Net Price and FF.Total Values would each be calculated at the fixed Euro/FF conversion, rounded to the nearest Cent.

So, if the exchange rate were 7.75735, the correct calculations would be:

FF				Euro
9.90	÷	7.75735	=	1.276209
				Rounded to 2 decimals =
				1.28
990.00	÷	7.75735	=	127.6209
				Rounded to 2 decimals =
				127.62
29.50	÷	7.75735	=	3.802845
				Rounded to 2 decimals =
				3.80
2950.00	÷	7.75735	=	380.2845
				Rounded to 2 decimals =
				380.28
3940.00	÷	7.75735	=	507.9054
				Rounded to 2 decimals =
				507.91

But if the **program** producing the quotation calculated the Euro Total by summing the Euro Net Price column, it might well reach a different answer:

EU.Quotation

Quantity Description	EU.Unit Price	EU.Net Price
100 Ring Binder	1.28	127.62
100 500 Sheets A4 copier paper	3.80	380.28
EU.Total		507.90

The point here is that the CORRECT total, according to Article 235, is EU507.91.

This is obviously a very simple example. The situation would become even more complicated if the French company needed to issue the quotation in Deutsche Marks, because of the need for Triangulation.

Any programmer who has ever coded an invoice VAT calculation will know that, depending on exactly how programming variables are specified (integer, fixed decimal, floating point etc.), you may get slightly different results if you calculate VAT line by line or calculate it once on the net total. Because such inconsistency would cause confusion between trading partners, Article 235 states that all calculations must be performed in Base Currency and then converted immediately before output. This, together with the relatively short duration of the Transitional Period, suggests that most organizations will strive to minimize changes by simply overlaying a conversion filter on existing output.

Business Risk

Whenever business processes and information systems require extensive modification, there are substantial risks that either the modifications will fail to operate correctly, will de-stabilize other processes, or will introduce undesirable side-effects.

Taking the example of the French company's quotations system, imagine that a programmer is asked to modify the application to give the option of producing quotations in Euros. A number of scenarios might result:

- (i) The programmer might implement the specification correctly, and in all circumstances the FF and Euro quotations produced would be exactly equivalent according to Article 235.
- (ii) The programmer might make a programming error that means that, in some circumstances, the FF and Euro quotations might not be exactly equivalent.
- (iii) The programmer might not correctly interpret the specification or the provisions of Article 235, and in some or all circumstances the quotations are not equivalent.
- (iv) In modifying the program, the programmer might de-stabilize other functions of the system which are not directly related to calculation of the Euro equivalent values

Of course, there may be any combination of scenarios ii, iii & iv.

If incorrect values are calculated or applications are de-stabilized, the organization would, at best, suffer disruption and inconvenience. At worst, it could lose competitiveness, lose money or become liable for damages. Suppose, for example, that a stockbroker incorrectly quotes an equity value in Euros and a customer acts on the information and loses money?

Testing to Reduce Risk

The answer, of course, is to test applications to ensure that, in all circumstances, they correctly process and output Euro values equivalent to NCU and/or perform conversion between NCU, according to Article 235. And, in addition, we must ensure that unwanted side-effects haven't been introduced.

Conventional software testing, whether automated or not, works as follows:

- (i) Specify conditions
- (ii) Specify expected results
- (iii) Execute test
- (iv) Compare test results with expected results
- (v) Correct as necessary, and go back to step *iii*

This can be narrowly focused on the modifications, or broadly focused to encompass all functions to ensure that side-effects haven't been introduced. Also, if modifications are applied incrementally (i.e. modify, test, correct errors, test, correct errors...), then regression testing might be used to ensure that error correction doesn't upset correct behavior observed and tested at an earlier stage.

So, in our example above, if we were to modify the original, single currency quotation system so that we could, optionally, produce alternative currency quotations, we would need to:

- (i) Specify the conditions

Quotation required in Euros for:
100 x Ring Binders
100 x 500 Sheets of A4 Copier Paper

- (ii) Specify the expected results

Ring Binder unit price = EU1.28
100 x Ring Binders = EU127.62
500 Sheets of A4 Copier Paper unit price = EU3.80
100 x 500 Sheets of A4 Copier Paper = EU380.28
Total = EU507.91

- (iii) Execute the test
- (iv) Compare test results with expected results
- (v) If the results are incorrect, make corrections and go back to *iii*

This could be done manually for a simple system but, for anything even slightly more complex, the task would become onerous and error prone. Imagine, for example, a complex business process such as a tax assessment, with multiple inputs, rules, thresholds and rates.

In all situations, what is needed is a tool that can:

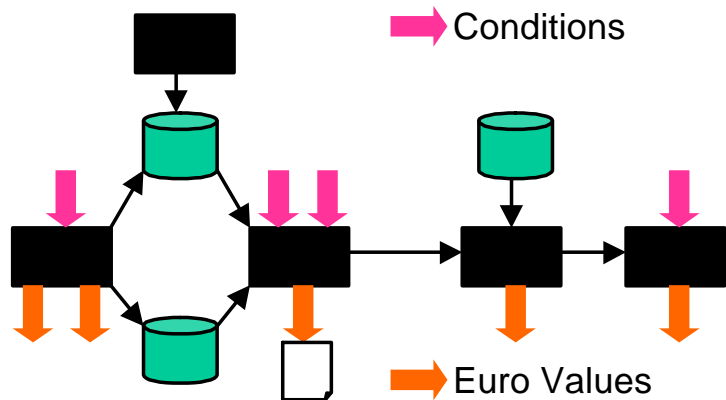
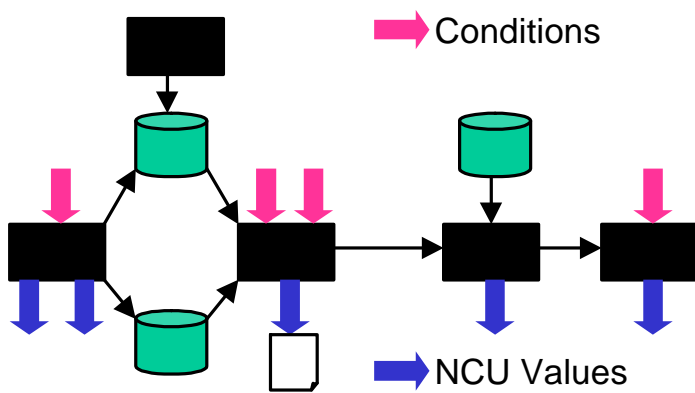
- capture NCU values output by a system for a given set of conditions when working in currency A
- automatically convert them according to Article 235 to produce the expected results in currency B
- execute the test automatically, comparing the test results with the expected results and reporting differences as errors

For example, going back to our quotation system, the tool would need to record the conditions from the FF quotation, and each of the NCU amounts: FF.Unit Price, FF.Net Price and Total. Then, when the Euro test is run, it must substitute the correctly calculated Euro amounts. In doing so, the tool must also take account of changes in numeric format (use of commas or periods, number of decimal places, currency symbol, etc.).

Because the tool is most unlikely to be able to distinguish monetary values from other numeric fields, operator intervention is required. The simplest way to achieve this is to allow the operator to highlight monetary fields while recording the test.

Also, during the recording the operator should capture other crucial elements of application behavior. These can then be checked during test execution to ensure that the modification process has not introduced undesirable side-effects.

The test tool should allow tests to run 'lights out', i.e. once initiated, they will run without operator intervention, processing transactions at the maximum possible system throughput while maintaining input/output synchronization. It should also be possible to build control structures into the test to enable a range of conditions to be tested.



Example of Euro Compliance Testing:

Quotation - QU05		Currency = FF	07-Jul-1998
Stock Code	Quantity Description	FF.Unit Price	FF.Net Price
RB100	100 Ring Binder	9.90	990.00
CP500A4	100 500 Sheets A4 copier paper	29.50	2950.00
Total			3940.00

Accept <enter>, Modify <PF2>, Reject <PF3>_

FF	9.90	÷	0.775735	=	EU	1.28		FF	9.90	
FF	29.50	÷	0.775735	=	EU	3.80		FF	29.50	
FF	990.00	÷	0.775735	=	EU	127.62		FF	990.00	
FF	2950.00	÷	0.775735	=	EU	380.28		FF	2950.00	
FF	3940.00	÷	0.775735	=	EU	507.91		FF	3940.00	

Convert to Euro

Capture NCU Values

Execute Test

Quotation - QU05		Currency = EU	08-Jul-1998
Quantity Description	EU.Unit Price	EU.Net Price	
100 Ring Binder	1.28	127.62	
100 500 Sheets A4 copier paper	3.80	380.28	
Total			507.90

Accept <enter>, Modify <PF2>, Reject <PF3>_

Report Results

```

Start - Euro Compliance Test Report.
=====
Transaction = Quotation - QU05
Currency = EU
Test Date = 08-Jul-1998

07-Jul-98 12:23:54 EURO QUOTATION Product Code = RB100
07-Jul-98 12:23:54 EURO QUOTATION PASS Euro.Unit Price for Ring Binders
07-Jul-98 12:23:54 EURO QUOTATION PASS Euro.Net Price for 100 * Ring Binders

07-Jul-98 12:23:54 EURO QUOTATION Product Code = CP500A4
07-Jul-98 12:23:54 EURO QUOTATION PASS Euro.Unit Price for 500 Sheets A4 copier paper
07-Jul-98 12:23:55 EURO QUOTATION PASS Euro.Net Price for 100 * 500 Sheets A4 copier paper

07-Jul-98 12:23:56 EURO QUOTATION *****FAIL***** Total

Expected value was:      EU507.91
Value displayed was:    EU507.90

End - Euro Compliance Test Report.
=====
    
```

Conclusions

The Euro deadline is 1st January 1999. The specification is that all monetary values expressed in both NCUs and Euros must be **exactly** equivalent according to Article 235.

IS departments and contractors historically have a poor track record for delivering mission-critical systems on time, and there is nearly always some latitude in the specification. But with the Euro, there is no contingency for either late delivery or failure to meet the specification. Any failure in mission-critical IS can cripple an organization, so it is absolutely essential that all reasonable steps are taken to ensure that any changes necessary to achieve Euro compliance are thoroughly tested.

In this context, the purpose of testing is to find out whether the application behaves in the same way, independent of the currency, given a defined set of circumstances. To validate this, the user can either:

- Laboriously work out the predicted set of results for every set of circumstances (if the answer in NCUs is A, then the answer in Euros should be B). Then run tests and compare predicted with actual results manually.

or

- Use an automated testing tool to capture A and automatically calculate the predicted result B, run a test and automatically compare the predicted with the actual result, reporting variances.

Given the wide extent and range of tests necessary, particularly if regression testing is employed, an organization can make enormous savings by using an automated testing tool which incorporates the necessary functions to convert NCUs to Euros (and vice versa) and automatically checks results for correct values. If it can also create a substantive audit trail that **proves** that the user has run adequate tests in a scientific and systematic manner, it can provide the levels of confidence essential for a business to make the transition into the Euro.

CYRANO have a 12-year track record in developing and deploying test tools and methodologies that help major organizations prove their mission critical IS.

CYRANO MillenniumTest is used by hundreds of users to test mission-critical systems for Year 2000 compliance.

CYRANO EuroTest, which is 100% compatible with CYRANO MillenniumTest, and will be released during the summer of 1998, provides a comprehensive solution to Euro compliance testing.

Euro Myths and Realities

Patrick O'Beirne
pobeirne@sysmod.ie



Systems Modelling Ltd. Suite 2, Villa Alba
Tara Hill, Gorey, Co. Wexford, Ireland
Tel: +353 55 22294 Fax: +353 55 22297

Gartner Group

- “Billions of lines of code to be changed
Estimated \$1.10 to fix each line of code
character to font sets, printer
drivers, keyboards, ATMs, cash registers
We're talking about taking all of this
information and changing it all, it's crazy”

Giga Group

- “Almost every system falls over when it has to do triangulation, the computer starts to choke
all systems will have to be able to account for six significant decimal places
sound like nit-picky little problems but each one of them will kill you
if a company converts its history, the numbers become almost meaningless”

Businessweek

- GETTING READY FOR THE EURO BUG
- A scary fact: That transition period, designed to coddle consumers, poses all sorts of frightening conundrums for computer systems
The challenges range from the mundane ... coming up with a computer key for the
to inserting the most abstruse mathematics into
- <http://www.businessweek.com/premium/13/b3571164.htm>

Manage the risk

“Managing the risk in euro currency conversions”

- Patrick O’Beirne, Cutter IT Journal June 1998
 - Most processing currency-independent
- Look for input , exchange & output functions

<http://www.iol.ie/sysmod/euroriskarticle.htm>

Quality Week Europe, November 1998

Panel Discussion : Euro Myths and Realities

* What are the real world challenges and experiences of those working the Euro problem?

Trying to get business managers to decide what they want

* Is the Euro conversion the ultimate Millennium challenge for Europe and what is the impact?

So is Y2K; so is e-commerce; so is the global economic environment.

* Are the business and technical challenges for the Euro conversion more difficult to resolve than those surrounding the Year 2000 problem?

Some are, some are not; see following.

* What are the myths and the realities surrounding the Euro Conversion?

See the following.

Patrick O'Beirne provides consultancy on Year 2000 issues for the desktop and euro conversion of business systems. He may be contacted at Systems Modelling Ltd., Tel. +353-55-22294 email pobeirne@sysmod.ie
URL: <http://www.iol.ie/sysmod/emu.htm>

Euro myths and realities

"We will comply with EU regulations whenever they come out"

These are characteristic statements from software suppliers on the euro, taken from an article in a recent buyer's guide to accounts software:

- "There is still no definitive interpretation of EMU compliance for accounting software"
- "Our development plan is ... subject to the directives from Brussels becoming available in time"
- "The EU has not settled the issue, but we ... will be ready when they are ready"

The facts are, the rules have been known since 1996 and embodied in Regulations of July 1997 based on Article 235 of the Maastricht treaty. Articles 4 and 5 of those regulations contain the conversion and rounding rules. There will never be a specification for euro-compliance from Brussels. That will be a matter for the marketplace. For example, there is no law that you must use the euro symbol in documents; in practice, people will want to.

"We have multi-currency accounting, we don't need to do anything"

Very few current 'multi-currency' packages meet the criteria in Articles 4 & 5 of the Regulations, as they were designed for the management of floating rates, not a six-digit fixed rate regime. The euro needs more than most multi-currency packages provide. Multi-base-currency packages have been proposed, and may suit some but one size does not fit all. And of course, even if you don't have to change your accounting, you may have to change your business.

"We must be ready on 1 Jan. 1999"

"We can afford to wait until 30 June 2002"

Both are potentially true, but may be misleading in your specific industry. There may be industry codes to practice to change on 1 Jan. 2001. "No compulsion, No prohibition" on the use of the euro only means that partners to a contract are free to contract in either national currency or euro terms. This is of course subject to normal commercial practice and pressures.

"Triangulation is mandatory in currency conversion"

The famous 'Triangulation' rule in Article 4 is that to convert from one participating currency to another, you must convert via the euro, and round the euro amount to not less than 3 decimal places. Other methods are allowed only if they produce the same result; and that is the let-out clause that software companies will be using. Triangulation is not a matter of government enforcement, national or local, but a matter of data integrity, audit trail, and maybe dispute resolution/litigation. Unless there is a dispute the method of calculation is not relevant. And it is only relevant where an alternative method, if used, reaches a different result. In Brussels there are certainly no plans to setup a "Triangulation Enforcement Agency".

It also depends on the purpose. As non-EMU currencies such as the dollar float against the euro, it makes no real difference how US trading companies do it as most commercial companies use an artificially fixed rate for a period and reconcile the differences at the end of an accounting period with the actual receipts from the bank.

"All amounts must be stored to six decimal places"

Computerworld 3 August 1998: "The euro requires that you keep six figures to the right of the decimal point, so fields designed to accommodate prices such as 5.99 will have to be expanded to 5.990000. "

Article 5 states that you must round to nearest cent for amounts to be paid; therefore rounding does not apply to other numbers such as unit prices such as a phone charge of .00416p/sec. Prices can be to any number of places you like, including none. Only the conversion RATES are to 6 significant figures, which only for Irish Pounds is six decimal places.

TESTING THE EURO

Complex simplicity

The Euro-project regards the introduction of simplicity. We remove all the different currencies, forget about foreign exchange rates and agree that one monetary currency has to be used in Europe¹. How simple life can be. This debate will not discuss the economical advantages and consequences of the introduction of this standard money-platform. We also know that this simplification requires quite a complex IT-modification. What we, IT-people, seem to forget is that also the business is undergoing a complex change.

Focus on IT

As with the Y2K-project, people heavily focus on the IT-side of the story. Indeed, there is a lot to do. Both Euro and Y2K require a lot of modification and adaptation work. Which of the two is more complex is not the issue here. It all depends on how you look at it.

Focussing on IT is not the only similarity between these two projects. Another one is that they are considered as “Conversion”-exercises. Not the functionality changes, the software only has to perform “business as usual” under different circumstances.

The assumption behind this logic is that business does not change. For Euro, nothing is less true. Business does change and not just to a small extent. The introduction of the Euro is more than “just a currency addition” (which, by the way, we as Europeans are very familiar with). The introduction of the Euro requires Business Process Re-engineering. After all, if we choose to introduce the Euro and business does not alter, why bother to change to Euro. Current processes will get a new or stream-lined life and new opportunities arise.

Business Process Thinking

The wrong assumption that we are dealing with a simple “conversion”-exercise has as a consequence that the planned testing, if there is any, is of the regression type.

1

²” (BPT). BPT™ starts by stating that the reason why we are busy with an IT-project is to make more profit. This is, or at least should be, the prime objective of any IT-activity. Thus, the final “acceptance test” should be the measurement of a profit increase due to the new IT-solution that was implemented, even if it is “only” an addition of a currency. BPT™ then will decompose this simple, yet powerful acceptance criterion into sub-goals, which are the objectives of well-known business processes (e.g. “We must be able to send out”). These processes are then split up into different functions. These functions represent the functionality of the IT-solution.

This way of working causes test criteria to be based upon business rules (e.g. “I can write a letter”) rather than on technical criteria (e.g. “I can write a Word document”). This way of working prevents us from relying on assumptions such as

² BPT: Business Process Thinking: Technique applied to analyse a IT-functionality from a business process point of view instead as of from a classical functionality point of view.

the structure of the available functionality or the stability of the business requirements.

When reading the above, one might say that the applied arguments are generic to any IT-project. This is indeed the fact. The Euro-project is a “normal” IT-project. It requires the same approach as any professional software (and hardware) development. The Euro is not more complicated, nor simpler than any other IT-work.

The Euro opportunity

What is different, however, is that we are all involved. The Euro- (and Y2K-) project is an issue that we are all tackling. So there must be an opportunity of economies of scale. We should encourage the exchange of experiences in these projects in order to improve effectiveness and efficiency. This way the IT-cost will considerably decrease. One of these experiences is structured testing. We know that testing costs a lot of money or, to put it more correctly, exposes how much money we really spend achieving a specific quality level. Testing is considered as a necessary evil. As such, little is invested in it. Sharing Euro-test experience could mean that we share test effort. We outsource our testing to external teams or we adopt a standard testing approach. We hire “test capacity”. We decrease our cost, both in testing, but also in post-implementation support (the software will be of higher quality).

Shared experience is also a leverage to get to a higher level of Software Engineering. We should look at the Euro-project as the momentum to detach ourselves from our legacy of craftsmanship and to become real software engineers.

We must keep a relative perspective however. Although the Euro is an opportunity for IT to mature, the issue itself remains a business matter.

Euro characteristics(1)

Graham Titterington

- Euro is a business issue with IT implications
- The Euro project is about achieving business benefit
- EMU is being introduced in 3 phases
- Many new applications will be needed

The Euro Conversion - Myth vs reality



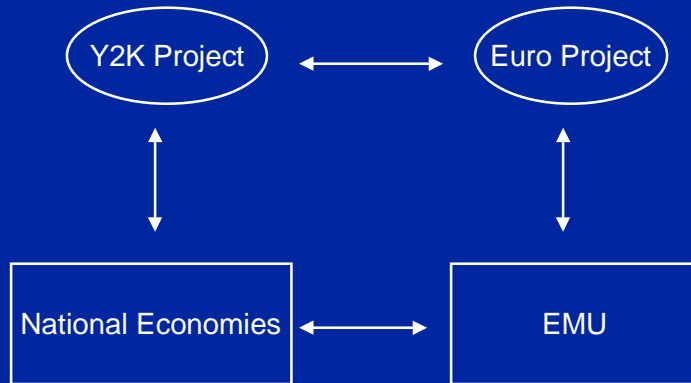
Euro characteristics (2)

- Most Euro related changes are extensions to applications
- Euro changes only affect application layer
- Euro changes are logically more complicated than Y2k changes
- You need to concurrently process new and old currencies, and store historic data

The Euro Conversion - Myth vs reality



The risk of interaction



The Euro Conversion - Myth vs reality



Conclusion

- Most Euro related changes are not mechanical conversions
- The Euro project is flexible in extent, time-scale, and strategy

The Euro Conversion - Myth vs reality



Position Paper:

The EURO Conversion – Myth versus Reality!

By Graham Titterington, *Ovum*

Euro characteristics(1)

- Euro is a business issue with IT implications
- The Euro project is about achieving business benefit
- EMU is being introduced in 3 phases
- Many new applications will be needed

Euro characteristics (2)

- Most Euro related changes are extensions to applications
- Euro changes only affect application layer
- Euro changes are logically more complicated than Y2k changes
- You need to concurrently process new and old currencies, and store historic data

The risk of interaction

Conclusion

- Most Euro related changes are not mechanical conversions
- The Euro project is flexible in extent, time-scale, and strategy

The Euro Conversion – Myth vs Reality

Discussion position paper for the 2nd International Quality Week

Graham Titterington

Ovum Ltd

e-mail gct@ovum.com

Characteristics of IT Euro projects

The introduction of the Euro is primarily a business issue. Beyond the basic requirement to enable financial transactions to occur, the purpose of a Euro project is to deliver business benefit. The role of IT is to support business needs. Therefore supporting IT projects contain a small element of mechanical conversion, and a large element of extensions to support new business processes.

IT changes for the Euro can be categorised into four groups:

- changes needed to enable businesses to trade using Euro
- changes needed to maintain historical records in old currencies
- extensions to enable systems to handle both Euro and national currencies during the transitional period
- new systems and extensions needed to support new business methods.

The second and third groups can be considered as transitional, and account for most of the subjects which regularly appear in discussions about the Euro in IT, including *triangulation*. It is important to look at the substance of the changes which have to be made, and the spirit in which they were formulated, and not to get bogged down in the negative minutiae of compliance.

Changes needed to enable businesses to trade using Euro

These comprise changes to:

- enlarge the size of fields holding monetary values
- add a decimal point in countries where there is no sub-unit of currency in common current use because the main unit of currency is of relatively low value (for example Italy, Belgium, Portugal, Spain)
- add a currency symbol to avoid ambiguity where this is not already present
- redesign forms and schemas to accommodate these changes, if necessary
- modify the ranges of values that are allowed in currency fields and variables.

Many financial processes are triggered by financial values (for example credit checks) and so clearly these trigger levels will need to be re-calculated for the Euro.

Note that there are no permanent changes affecting foreign exchange calculations. Once the transitional period is complete the Euro will be a normal international currency with a very large multi-national economy behind it.

Scope of the software problem

The number of programs affected by the EMU is much smaller than the number affected by year 2000. The Euro is concerned with money; year 2000 is concerned with time. Money is not as pervasive as time in the computing world.

Although the scope of the problem is more limited, where changes to programs are necessary, they are more logically complex than the typical date related changes. The date problem is mainly about data representation, whereas the Euro changes involve altering the logic of applications and developing some substantial new applications

Is Software Testing Scientific?

Bogdan Bereza-Jarocinski, ENEA Test

Enea Data AB

www.enea.se

Box 232, S-183 23 Täby, Sweden

Tel: +46-8-638 50 00

Fax: +46-8-638 50 50

e-mail: bogb@enea.se

Human Cognition

- **The reach of knowledge**
 - private - local - public knowledge
 - reasons for spreading knowledge
- **How new is knowledge?**
 - pure science versus applied science
- **Is knowledge true?**
 - various ways to validate knowledge

Validating Knowledge

- **Superstition**
- **Intuition**
- **Authority**
 - good and bad aspects of generalisation
- **Rational-Inductive Argument**
 - various levels of rational-inductive argument
 - correct argument but wrong conclusions
 - theorem proving in theoretical sciences
 - source of hypotheses in natural sciences

Experimentation

- **Excluding extraneous variables**
- **Measurement of results**
- **Operational definitions**
- **Experiment in engineering: build it**
- **Limitations of experimental method**
 - permanent relationship
 - statistical relationship

Descriptive Statistics

- **Statistics - cure for limited knowledge**
- **Significance**
- **Correlation**
- **Choosing right validation method**
 - experimental and observational studies
 - causal and correlational relationship
 - limitations for evaluating significance

Offences against Science 1 (3)

- **Confounds**
- **Operational definitions**
 - example: test suite effectiveness
- **Independent and dependent variable**
 - relationship between variables
- **Internal validity**
- **External validity**

Offences against Science 2 (3)

- **Experimental group**
- **Control group**
 - inadequate control groups in research around QA
- **Group equivalence**
- **Effect of longitudinal (sequential) studies**
- **Using representative samples**
- **Sampling techniques**

Offences against Science 3 (3)

- **Different populations in SW testing:**
 - user activities (SUT), population of bugs (bug seeding), population of systems
- **Observer bias and expectancy**
- **Hypothesis testing**
 - null hypothesis
- **Variance and standard deviation**
- **Multi-dimensional environment**
 - limitations in validity, possible solutions

Conclusions

- **Are we the only sinners?**
- **Is software testing science?**
 - common denominators with other testing
- **Prohibitive costs of experimental studies**
- **How to improve quality of observational**
 - comparative thinking
 - ask how representative sample is
 - estimate statistical significance

The Goal of Quality

- **Making money - market success - right quality quickly and economically - test as**
- **Correlation between quality and quality**
 - achieving quality through quality systems?
 - relationships between “qualities”

Introducing Quality Systems

- **Disadvantages of quality systems**
- **Bottom-up implementation of quality**
 - need for right dimensioning of quality system
 - disadvantages of top-down introduction
 - need for independent *quality engineers*

Psychology of Testing

- **Plenty of psychology in literature on**
 - “testing is destructive”, “test is mental discipline”, “testing is creative and difficult”, “testing requires independence”, “test results must be carefully controlled”, “test specialists must be*
- **Advantages of using mainstream psychology**
- **Theory of cognitive dissonance**

Constructive Work

- **Creative activities and intrinsic**
- **Why is testing disruptive?**
- **Re-defining test to make it more creative**
- **New role for test automation?**
- **The boredom factor**
 - optimal level of stimulation
 - how to put testing on this level

Neurosis of Testing

- **Security and anxiety**
 - emotional disorders and excessive anxiety
 - obsessive-compulsive disorders
- **Security and anxiety of testing**
 - emotional need for secure systems
 - emotionally hard to accept SUT and MTBF
 - when to stop testing - is there “obsessive

Social Psychology of Reviews

- **Potential for quality improvements**
- **Basic contradictions of reviews and**
 - unwillingness
 - cognitive impossibility
 - external impediments to reviews

Group Dynamics in Reviews

- **What is group dynamics?**
- **Disadvantages:**
 - group pressure
 - Asch conformity experiment
 - group solidarity
 - “groupthink”
 - group polarisation

Review psychology - continued

- **Communication, information gathering, creating technical solutions - when?**
- **Can algorithms be verified with**
- **Holland's six occupational interests**
- **What should be done:**
 - let psychologists re-asses existing review

experimental studies are possible in this area

Why is QA so Boring?

- **Partial explanations possible through concepts used for review analysis**
- **Hackman's model of work design:**
 - **job contents description**
 - skill variety, task identity, task significance, autonomy, feedback
 - **job evaluation**
 - meaningfulness, responsibility, knowledge of the
 - **job improvement**
 - combining tasks, natural group units, client relationship, vertical loading, better feedback

Psychology of Usability

- **For engineers: esoteric, fuzzy, soft**
- **Increasing number of unfriendly devices**
 - “human error” is really bad design
- **Bad MMI typical for software inflicts**
- **Usability testing limited to MMI, not valid for whole systems in their**

Factors of Usability

- **Intuitive usage**
- **Models**
- **Visibility**
- **Mapping**
- **Feedback**
- **Social pressures**
- **Designer terrorism**
- **Human errors and natural constraints**
- **What can be done:**
 - marketing
 - design
 - QA (requirements)
 - testing

Is Software Testing Scientific?

Bogdan Bereza-Jarocinski, ENEA Test

Enea Data AB

www.enea.se

Box 232, S-183 23 Täby, Sweden

Tel: +46-8-638 50 00

Fax: +46-8-638 50 50

E-mail: bogb@enea.se

© 1998 Enea Data AB

All rights reserved

Abstract

This paper deals with three relatively independent subjects:

- Analysis of practices used in the *study* of software testing (i.e. not in the actual process of testing itself) from the point of view of scientific methods. Applied to discussion on correlation between quality systems and product quality.
- Discussion on how certain psychological and sociological theories and concepts can be profitably applied to analysis of testing *process*.
- Psychology of usability: an attempt to depict it in broader terms, not limited to windowing standards and GUI.

These subjects, although different, have three features in common:

- They are interdisciplinary.
- They are mostly neglected or ignored at present.
- They exist at crossroads of technology and social sciences.

Key words

scientific method, descriptive statistics, psychological analysis, software testing, QA, psychology of usability, usability, experimentation, product quality, quality systems, significance, correlation, causal relationship

1 Introduction

This and the following chapters contain concepts, which may be new to software professionals. They are explained farther in the text. They are not cross-referenced to their definitions to avoid too many cross-references.

1.1 The Rules of Scientific Reasoning

The knowledge of descriptive statistics and of rules of scientific reasoning among software professionals is insufficient. Teaching courses in software testing I have many times observed that participants had problems grasping concepts of Statistical User-Based Testing and found it “fuzzy” and “not enough technical”. Concepts like *correlation* or *statistical significance* are never mentioned

1.2 Quality Assurance

Another group of my experiences, which motivated me to write this paper, refers to QA. Why are QA measures so often experienced as stiff, boring, bureaucratic – a burden rather than help for developers and project managers?

1.3 The Psychology of Testing

Studying software testing I have encountered many statements that belong to psychology. Examples: “testing is destructive”, “test is mental discipline”, “testing is creative and difficult”. However, all this is not connected to mainstream psychology; rather, authors rely on their own “psychological intuition”.

1.4 The Design of Things

The fourth area about which I have for a long time experienced intense wonderment is why so many things are so strikingly badly designed?

1.5 Are These Areas Related?

These four issues are in fact closely related, in more than one respect. They are problems that have grown at the crossroads of technology and social sciences, at no-man’s-land between everyday experience and science.

1.6 The Status of This Paper

This paper is only a tentative essay. I point out and partially describe a number of possible areas for improvement or for further study, but I have no own research results to present.

2 Human Cognition

2.1 Definition

Human beings create in their mind representation of the outside world. This representation is called *knowledge* about the world.

2.2 The Reach of Knowledge

2.2.1 Concepts

Some knowledge is *individual*, some is *local* for a limited group of people like a family or a neighbourhood, or a company department, some is *public* – it has been spread and stored so that anybody can access it.

2.2.2 Application

Applied to software testing the purpose of magazine articles, books on testing or conferences is to spread knowledge about testing, to make it more public. However, it is worth remembering that making knowledge more public is not a goal in itself. In the age of WWW we are aware that filtering and selecting knowledge is at least as important as making it accessible.

2.3 How New Is Knowledge?

2.3.1 Scientific Knowledge

Generally, gathering and producing new knowledge is the ultimate goal of pure science. For practical purposes however, it is often more useful to discover that a certain well-known fact is applicable in a given area than to create advanced theories that have no immediate application.

2.3.2 Applied Sciences

This is worth keeping in mind when dealing with applied sciences. Otherwise one may be tempted to regard one's favourite area too seriously. For a company acting in market environment, the right dimensioning and direction of its R&D effort is of prime economic importance.

2.4 Is Knowledge True?

This is the basic question for the next two chapters. Any piece of information may be true, partially true, or false. “Automating testing using our Hyper-Bang-Bug-Killer-Test-Tool will cut your lead-times by half” (and cost you a lot of money). To assess the validity of this claim may be critical for a company's survival as well.

3 Validity and Verification of Knowledge

Theories and opinions can be validated by various means. This means that assessing whether to expect that adding vinegar to water *really* does enhance the way your hair looks can be done in a number of different ways.

3.1 Superstition

You believe computer hardware is strongly influenced by full moon, therefore all testing should be done only then. Even if pure superstition of the grotesque kind like in the example above is rather rare in software testing its elements exist.

3.2 Intuition

The borderline between superstition and intuition is hard to draw. Intuition, when applied by an intelligent person with extensive experience, not strongly emotionally involved in the field that she will validate, can be very effective.

3.3 Authority

3.3.1 Example

You know that Boris Beizer is an established authority in testing, therefore you follow his recommendation that 100% statement coverage is a must.

3.3.2 Generalisation

This approach may be rational. Generalisation is a cognitive mechanism that allows humans to structure their knowledge and recognise environmental elements more effectively than would be feasible otherwise.

3.3.3 Authority as Validation Tool

To validate all information coming from various sources by checking it by oneself is a time-consuming method. An alternative way is to assess the information source; once it has been classed as trusted, you accept by default all information coming from it.

3.4 Rational-Inductive Argument

3.4.1 Examples and Limitations

Certain *rules of reasoning* are used in order to arrive at a conclusion. The rules are either well defined and precise (formal logic, mathematics), or fuzzy. They may be correct or wrong. They may be used intentionally or unconsciously. Note that even correct inductive argument may lead to wrong conclusions if premises are not true.

3.4.2 Rational-Inductive Argument in Natural Sciences

In natural and social sciences rational-inductive argument is used to create theories and formulate hypotheses, which are then proved using experimental or statistical methods (see sections below).

3.4.3 Rational-Inductive Argument in Theoretic Sciences

In mathematics and all sciences with strong mathematical component (like parts of computer science), rational-inductive argument is widely used to prove theorems. Numerous areas of software testing are derived from mathematics. Control-flow testing, path testing, loop testing, transaction-flow testing and finite-state testing have strong bonds to graph theory. Domain testing is based on linear algebra and set theory, syntax testing on metalinguistics. Matrix calculus is sometimes used in configuration testing. Formal logic and theory of probability are used extensively as well.

3.4.4 When Rational-Inductive Argument is not Enough

Of course, no experimental or statistical backing is needed for them any more than “two plus two equals four” needs statistical proving by adding twos of different objects! However, as soon as mathematical theories are used as *models* of natural phenomena, the validity of models must be verified with experimentation and statistical calculations.

3.5 Experimentation

3.5.1 Example

Compare the results of two test methods to see which is better.

3.5.2 Experiment Quality: Extraneous Variables

One difficulty in designing experiments is such a way that *extraneous variables* cannot influence results, thus hiding the influence of the *independent variable*.

3.5.3 Experiment Quality: Operational Definitions

The example above demonstrates other difficulties. For fuzzy variables like “better test method” *operational definition* of variable and its values is needed. Unless exact, unequivocal and measurable definition is used, conclusions will always be open for discussion. If operational definitions are formulated early, flaws in experimental design can sometimes be discovered in advance.

3.5.4 Experiments in Natural Sciences

Experimental method is a de-facto standard in applied sciences including engineering. If you can make something (an appliance, a piece of software) and it works as expected, you have proven that your design is correct.

In physics, chemistry and biology the result of one or few experiments may be enough, too. Unless there are strong reasons to suspect the influence of uncontrolled extraneous variables, experiments need not be repeated. If the number of suspected extraneous variables is large, then a number of required trials must be larger to minimise the probability that the result obtained in one experiment is contrary to what would occur if value of an unknown variable was different.

3.6 Statistical Methods

3.6.1 Application Area

Statistical methods are used either because our knowledge is incomplete or because “laws of nature” are statistical. When studying complex enough systems like humans or large computer systems, only statistical knowledge is available.

3.6.2 Basic Concepts: Correlation and Significance.

Correlation is “a measure of the degree of relationship between two variables”. There exists for example *positive* correlation between body height and weight but not a 100% correlation. One may expect *negative* correlation between test coverage and the number of bugs not found during testing. Actually, most studies about software testing and quality assurance are typical correlational studies.

Significance is a measure of how likely a result (a correlation a mean value, or a difference between two values) is to have occurred by chance alone. This value is of interest whenever an attempt to make conclusions about some *population* in general based on a *sample* from that population. A number of factors affect significance sample size, expected (or known) distribution, standard deviation etc.

Knowing the significance of a result is necessary to be able to predict whether it can be generalised even to other, similar situations.

3.7 Choosing the Right Verification Method

3.7.1 Experiments for Conclusions about Causality

Experiment results can be used as a basis for conclusions about the existence of *causal* relationship. Both experiments that show permanent relationship (gravity makes apples fall down) and experiments that reveal statistical relationship (boredom causes people to seek more stimulation) can be used. The important point is to design experiments in such a way that we are able to draw inferences that changes in the level of independent variable have *caused* the corresponding changes in dependent variable.

3.7.2 Observations for Conclusions about Correlation

For observational it is usually not possible to make conclusions about causal relationship. Only conclusions about the existence of *correlational relationship* are valid for them.

3.7.3 Different Methods for Different Purposes

There is no telling which method is strongest or best, it depends on what is being validated, what data is available and - last but not least - what the requirements are.

3.7.4 Pitfalls of Statistics

Statistical methods can be dangerous when used incorrectly, as they give a kind of false “dignity” to unscientific (and incorrect) conclusions.

4 Common Offences against Scientific Methods

4.1 Confounds

4.1.1 Definition

Confounds are *flaws in research design that permit alternative explanations for the results*. Not taking into account possible extraneous variables is a typical confound. Thus, confounds may – but not always do – lead to false conclusions.

4.1.2 Confounds in Studies about QA

In studying software testing and QA we are more often than not reduced to carry research on whatever data happens to be available, we cannot design own research nor carry own experiments.

There exist many kinds of confounds, they are described more in detail in the following sections.

4.2 Operational Definition

4.2.1 Definition

Operational definition is the exact procedure used to produce a phenomenon or to measure a variable. For science, operational definitions are what test instructions are for testing.

4.2.2 Example: Operational Definition of Test Effectiveness

A vital part of software testing is the study of various methods of generating test cases. They are compared to each other to find out which is most effective. However, an operational definition of how effectiveness of test methods can be measured is missing. Actually, no single definition is enough, but a number is needed. The ability to find typical bugs, the ability to find unusual bugs, the ability to find design bugs, the ability to find random bugs – all these are different scales.

4.2.3 Operational Definitions in QA - Conclusions

In the research around testing and QA it is prohibitively difficult to formulate working operational definitions. It means that it is very difficult to be exact in our evaluations of various test methods. Therefore today these evaluations are based more on experience and “gut feeling” than on substantial data.

4.3 Independent and Dependent Variables

4.3.1 Definition

The variable manipulated by researcher in an experiment is called independent variable. The variable whose changes are measured (and presumed to be caused by the changes of independent variable) is called dependent variable.

In study of testing, independent variable can be - for example - the extent to which formal reviews are used during development, and dependent variable - the ratio of bugs found before dynamic testing to those found during it.

4.3.2 Variables and Causal Relationship

The important question is whether measured changes in the value of dependent variable are really *caused* by manipulating independent variable or are they caused by other, extraneous variables?

4.3.3 Possible Extraneous Variable

For example, the introduction of formal reviews may force projects to use more written specifications or to employ more experienced project leaders, which - and not the review process - could be the real reason for better bug discovery before testing. Such situation cannot be excluded - even if it feels contrary to what you believe - unless both groups of projects can be made equivalent in all respects but the different levels of independent variable (i.e. reviews) – see chapter on group equivalence §] for more details.

4.4 Internal and External Validity

4.4.1 Definitions

Internal validity of a study and its results is a measure of how well it answers the actual question. If no metrics from the period *before the introduction* is available, no reliable answer to that question can be provided and the study has low internal validity. If those results are available, the internal validity is higher.

4.4.2 Pre-conditions of External Validity

External validity of a study is the extent to which its results can be generalised beyond the actual study. For high external validity to exist certain conditions must be fulfilled:

- some “beyond” must exist, i.e. the study is devoted to a project or product that is *not* totally unique
- there is no reason to expect any statistical distribution of the results; i.e. the relationship is permanent.
- results are statistically significant (for studies which do not fall into the category above)

Studies with high internal validity can have either high or low external validity, but not the other way round (low internal validity always entails low external validity).

4.5 Experimental and Control Groups

4.5.1 Example

In experiments, two (or more) values or levels of independent variable are used. For example, an experiment to check how reviews affect time distribution of bug discoveries may study two groups/projects: one that uses reviews (experimental group), one that does not (control group).

One widespread problem for experiments and observations in testing and QA is the lack of control group.

4.5.2 Group Equivalence

In order to minimise the influence of extraneous variables (and thus maximise the internal validity of studies), experimental and control groups should be *as equivalent* as possible with regard to those variables which can be suspected of influencing the results. If, for example, we have reasons to suspect that engineers' experience may substantially influence the results of how successful studied test method is, it is advisable to assure that experimental and control groups do not differ much in this respect.

4.5.3 Confounds in QA Due to Group Composition

Many difficulties, which decrease internal validity of research in software testing and QA, arise from the fact that the same group serves first as control group (before introducing the studied method), then as experimental group (after introducing the method).

4.5.4 Reactivity Problem

Reactivity is the change in behaviour caused by the subject's knowledge that she is being studied. In social sciences its effects are called the *Hawthorne Effect*. The mere knowledge of the subjects that productivity was studied caused increased productivity, regardless of other factors! This effect - largely unknown to engineers - certainly may lead to spectacular positive effects of introducing not only new QA measures but even of simply gathering metrics.

4.5.5 History Effects

History effects are outside conditions which - unlike extraneous variables - do not exist permanently but simply happen to occur at the same time when dependent variable is measured. Events like company acquisition, major market turbulence or radical product re-design can influence results and effectively hide the relatively smaller change caused by the independent variable.

4.5.6 Maturation Effects

Maturation effects are changes in measured values of dependent variable due to reasons like subject experience, tiredness or boredom. If one company department is used for trying many new QA methods, then after some time of such maltreatment it is bound to stop reacting to whatever new schemes are inflicted on it.

4.6 Representative Group

4.6.1 Definition

How representative the studied sample is *for the whole population* is the key for being able to evaluate external validity. Calculating statistical significance of a finding makes sense only provided the studied sample is not biased; i.e. that it fairly represents the population.

4.6.2 Consequences of Non-representative Groups

By failing to take into account that non-representative sample is used, false assumptions of external validity can be made. It is for practical reasons difficult to do much about actually choosing more representative samples for the purpose of studying test methods.

4.6.3 Various Populations in QA-studies

Evaluating the effectiveness of test methods or other QA-measures it is necessary to take into account how "representative" is the situation in which it is applied. This situation is multi-dimensional i.e. it contains elements from a number of populations.

4.6.3.1 The Population of User Activities

How well do test cases reflect what users actually do with the system? This subject is sufficiently covered by techniques like *Statistic User-Based Testing* (part of Cleanroom Engineering) and by *Software Reliability Engineering*.

4.6.3.2 The Population of Bugs

Example: for some types of bugs increasing statement coverage may yield higher bug hit ratio, whereas for certain “malicious” bugs even multiple condition coverage may not bring about any increase in the number of found bugs.

A technique called *bug seeding* makes small inroads in this direction. This technique attempts to evaluate test cases (and methods used to generate them) by applying them to programs with some known (seeded) bugs. What is missing from this technique is the evaluation of to what extent the seeded bugs fairly represent the population of all bugs. If they are not representative, then it is invalid.

4.6.3.3 The Population of Systems

This subject is fairly well covered; most test techniques are discussed from this perspective. However, these conclusions are mainly based on rational-inductive argument or intuition and cannot be considered as sufficiently proved.

4.6.3.4 The Population of Projects

Example: whether test automation is worthwhile or not does not depend so much on technical aspects of the tested system but rather on project aspects: lead-times, required quality, the expected extent of regression testing etc. This particular area is on one hand pretty well covered by numerous case-studies (“Success/failure story: XX-testing in YY-project at ZZ-company”), on the other hand this knowledge is chaotic, unstructured and its pieces are not connected to each other.

4.6.3.5 The Population of Development Techniques

Only isolated pieces of knowledge are available in this area. What test methods work best for which development paradigms?

4.7 Hypothesis Testing

4.7.1 Example and Definition

Let us assume that we have some data (from a handful of companies) about the existence of positive correlation between company’s financial results and the percentage of money it spends on test and QA in its R&D.

Even without making unwarranted conclusions about causal relationship we might wish to know how significant this finding is, how probable it is that there would be no such correlation if we took *all* software companies into account. The statistical technique to be used is called hypothesis testing. The so-called *null hypothesis* (in our example, that there is no correlation in the population in spite of the correlation present in the studied sample) is formulated. The probability of rejecting true null hypotheses and of failing to reject false null hypotheses is then calculated.

4.7.2 Significance

Hypothesis testing is central technique for the calculation of statistical significance (and thus external validity) of findings. Surprisingly, it is hardly ever mentioned in literature on testing and QA.

4.8 Many Variables Involved

4.8.1 Multiple Variables when Studying Testing

Compared to social sciences, research in the realm of testing and QA must cope with definitely less science-friendly environment for observations and experiments. The dominance of observational studies over experimental studies has been mentioned already. Another problem is that the number of variables involved is typically very large, which makes it much more difficult to discover relationship between any two of them.

4.8.2 Planning for Higher External Validity

Two possible actions can be undertaken to minimise the negative consequences of this. One, which has already been recommended a few times, is to be more prudent when drawing inferences about possible relationship between two variables.

The other possible action is - as much as possible - to plan the evaluation, introduction and follow-up of measures in testing and QA in such a way that the influence of extraneous variables diminishes or becomes more controllable.

4.9 Consequences and Possibilities

In the previous chapter I have mentioned a number of offences against what can be called good scientific style, which even technically brilliant authors and speakers commit sometimes. What are possible practical consequences of these observations?

4.9.1 Testing in Perspective

By the way, this situation is in no way unique to testing, it appears in other applied sciences too. For example, various management methods or software development methods could be very appropriate objects for parts of this paper. To take but one example, consider Object-Oriented Development. Examples can be found outside software industry as well.

4.9.2 Is Software Testing Science?

4.9.2.1 Is There Testology?

Software testing is not an established and separate branch of science, nor – in my opinion – it will ever be. The same probably applies to testing products of human activities in general: there is no “testology”, which software testing would be part of. Obviously, many similarities exist between development testing of mechanical designs (like cars or washing machines) and testing software systems. Today these two fields keep converging, because embedded systems become more widespread. Some similarities exist even between software testing and testing of still other kinds of systems: administrative routines, organisational structures, distribution systems, architecture, design of traffic flows etc. Interesting lessons for software industry can surely be learned from studying test methods used in other industries and I am looking forward to reading or listening to a comparative study of these areas. However, the existence of a common denominator for these branches does not constitute a “testology”.

4.9.2.2 Testing is a Skill

Software testing (or even system testing in general) is a practical skill or practical study, not a science. In software testing, methods borrowed from various established sciences are used, mixed with a number of practical skills. This fact defines the scope of the application of scientific rigours for studying software testing: there is no intrinsic requirement for using them.

4.9.3 Advantages of Scientific Discipline

4.9.3.1 Avoiding Pitfalls

However, as we know from everyday experience, using some scientific discipline is often beneficial even for most mundane, trivial tasks. It simply guards us to some extent against pitfalls of sloppy thinking, wishful thinking etc.

4.9.3.2 Less Chaos

A lot of chaotic discussions and spurious arguments could be avoided. Subjects like value of ad-hoc testing, benefits of test automation, strategies for regression testing or the level of required coverage are objects of heated arguments, which are in a sense futile. No number of examples, however striking, can *prove* that - for example - ad-hoc testing is either always beneficial or always harmful.

4.9.4 Recommendations

4.9.4.1 Difficult to Promote Experiments

For practical reasons, I do not believe there is much sense in promoting more *experimental studies*. The costs could be prohibitively huge, the findings probably not very significant due to a large amount of involved variables. Just imagine starting three or more parallel versions of one project in order to evaluate different testing methods!

4.9.4.2 Low Predictive Value

On the other hand, from the point of view of non-profit scientific institutions, such studies would not yield interesting enough “scientific return”. In spite of daunting organisational complexities, the findings would be rather trivial. These findings would constitute separate, isolated chunks of knowledge, which could not easily be connected together into theories. Their predictive value would be tiny.

4.9.4.3 Enhancement of Observational Studies

The quality of *observational studies* could be significantly enhanced with little additional cost. A few fundamental rules of design of scientific studies should be applied when you decide what kind of software, process, project or bug metrics to gather.

4.9.4.4 Comparative Thinking

Simply gathering lots of data that cannot be compared to any other similar data normally does not allow us to discover trends or to estimate relative effectiveness. Are there fewer bugs after shipment when the new test tools used? To get an answer, we need at least two result data sets.

4.9.4.5 Are Samples Representative?

For example, to be able to draw inferences about relative values of 60%, 80% and 100% coverage we must know that no other variable (like module complexity, programmer experience or the quality of specifications) has influenced the results.

4.9.4.6 Significance Estimation

To *calculate* statistical significance is probably hardly ever achievable since it requires knowledge of the type of distribution of the variable(s) being measured. However, an intelligent estimate would be acceptable.

4.9.4.7 Pitfalls of Unwarranted Conclusions

A typical pitfall for human mind is hasty conclusions about causal relationship “the introduction of this method has *caused* the number of bugs to lessen”. The truth is, as mentioned previously, that the *correlation* between the number of bugs and the use of a certain method has been observed. Such false conclusions are simply not true (more exactly: we do not know whether they are true or not) and very misleading for readers not prepared to critically assess the validity of conclusions.

4.9.4.8 “So What?”

It is profitable to try to formulate prudent but general and comparative conclusions even for most down-to-earth, self-contained case studies. From the point of view of distributing knowledge, just telling “how we did it” is worth much less than the same information *plus* analysis.

4.9.4.9 Open Scientific Standards

We ought to be explicit about what scientific rigours your study has observed (and keep it in mind while formulating conclusions). This information enables readers to evaluate internal and external validity of your study.

5 Process versus Product Quality

5.1 The Goal of Quality

5.1.1 Profit is the King

QA gurus and test experts often sound as if quality systems and advanced testing techniques were goals in themselves. They are not. For private companies, the ultimate goal is to make money (without breaking against ethical or legal constraints) and to survive in a competitive environment. The latter requires striking appropriate balance between short-term financial gains and long-term adaptability.

5.1.2 Profit Components

To make money, market success and competitively low costs are necessary. For market success, you must be able to deliver what customers want when they want it, and you must do it at least as well as your competitors. To deliver “right” quality products as economically as possible, without sacrificing too much of potential for future development and improvements, is the ultimate goal of QA.

5.1.3 Test and QA

Testing is an important component of QA. It is not easy to achieve the right balance between test and other components of QA. Too much testing in comparison to requirement management, configuration management, security, handling of deliveries, trouble report handling etc. means that test results are not fully utilised for higher product quality.

5.2 Correlation between Quality and Quality Systems

5.2.1 Quality Systems

A widely accepted method to assure the “economical delivery of right quality” is to adopt *quality system*: a system of co-ordinated organisational and technical measures to support high process, project and product quality in a company.

5.2.2 Do Quality Systems Bring Quality?

However, the correlation between various “qualities” (process, project and product) and quality *systems* is by no means obvious. Interesting but contradictory success and failure stories abound, and the (pretty unscientific, often superstition-ridden) quest for the Holy Grail of economic success goes on. The funny thing is that - while not even a weak *correlation* is firmly established - many QA professionals maintain that there exists a *causal* relationship.

5.3 Negative Effects of Quality Systems

5.3.1 Love and Hate Relation

Most people officially profess their belief in quality systems, at the same time hating them deeply in private. This hate is by no means irrational: the introduction of quality systems entails a large number of negative consequences.

5.3.2 Why Hate?

These negative effects are well known to engineers in software industry. You become less productive as you have to spend a large part of your time with QA-related bureaucracy. Product changes become more difficult to make, you often choose not to fix a minor bug just to avoid the hassle involved. The development process becomes more involved in its internal workings than in trying to satisfy customers. The quality system is so complex that nobody - except perhaps QA manager high up - understands it fully. Futile and time-consuming discussions about correct document classes, numbering of product releases, trouble report statuses etc. entail. The tools used for QA-support cause problems.

5.4 Top-down and Bottom-Up Quality

5.4.1 Size is Important

At the same time, QA is absolutely vital. The problem seems to be the appropriate *dimensioning* of quality system. It must address real problems experienced by developers and low-level managers in order to be accepted and really useful for them. If the system is too large and comprehensive and some parts of it inadequate to the activities, then the cost of using it (boredom, irritation, time) will be higher than its advantages. On the other hand, the cost of not having any (or too little) quality systems is prohibitive, too.

5.4.2 Quality Managers

It is my belief that the reason for this unfortunate situation is that quality systems are mostly imposed from above. This fact is reflected in situations-vacant column: companies seek test (or verification) *engineers* but quality *managers*. Thus quality systems arrive top-down, which has a number of negative consequences, some of them mentioned above.

5.4.3 Quality Engineers

A much better solution would be - in my opinion - to introduce quality systems bottom-up starting from the actual needs of developers as experienced by them. The position of *quality engineer* would be a solution. Such person would be well acquainted with all local quality problems and needs and would solve them in close co-operation with developers.

Test professionals may have in many cases right qualifications to become such quality engineers.

6 Psychology and Sociology of Testing

6.1 Generalities

In the first chapters of this paper an attempt is made to analyse practices used in the *study* of software testing (i.e. not in the actual process of testing itself) from the point of view of scientific methods. Now we take a full turn: the subject matter of this chapter is how some psychological and sociological theories and concepts can be profitably applied to testing *process*. Only minor comments will be devoted to the *study* of testing.

6.2 The Status of This Chapter

This chapter – psychology and sociology of testing - is a tentative study, which means that I just give a few examples of possible applications of psychological concepts and theories to software testing. Probably, a more comprehensive study would reveal more, and maybe better application areas.

6.3 Cognitive dissonance

6.3.1 Description

The theory of cognitive dissonance could successfully be applied to study testing and explain many of the psychological aspects of testing. This theory, together with other theories belonging to *cognitive psychology* (a general name for the branches of psychology dealing with learning and cognitive processes), could be used to describe, explain and predict human behaviour during testing. Developers build systems and expect them to work, whereas testing process reveals when they *do not* work, which means that cognitive dissonance is often encountered in test situations.

6.3.2 Psychology of Testing

At present we have some quasi-scientific rules for how to organise testing effort, which attempt to take into account psychological factors. “Developers should not test their own programs”, “testing is creative”, “testing is destructive” etc. Using more systematic approach based on established psychological theories and following the rules of scientific research, a more comprehensive set of concepts about the “psychology of testing” could be perhaps created.

6.4 Constructive Work and Motivation

6.4.1 Creative Work

Human activities have two kinds of motivation: intrinsic (doing something for “its own sake”) and extrinsic (doing something in order to achieve a desirable goal which is located outside the actual activity). One of the intrinsic rewards of work situation is the gratification (satisfaction) derived from *creative* activity. All disturbances interfering with the flow of creative activity are experienced as disruptive and avoided. This knowledge accommodates well known experiences concerning negative attitude of developers towards testing and consequently historically lower status of test as compared to new development.

* Theory of cognitive dissonance: we are motivated to make adjustments to remove cognitive dissonance – a negative emotional state set up when two simultaneously held attitudes are inconsistent or there is a conflict between attitude and behaviour.

6.4.2 Re-definition of Test

If these conclusions are true, a radical re-definition of testing is required in order to make testers and managers alike to experience it as more creative and thus find more intrinsic rewards in it. Because test automation entails plenty of creative work, it creates intrinsic incentives for testing effort. This area seems very promising for further studies.

6.4.3 Optimal Stimulation Level

Experimental data indicate that there is an optimal level of stimulation, which people seek. Too little sensory stimulation is experienced as boredom and more stimulation is sought. Too much stimulation causes stress and individual attempts to withdraw or otherwise decrease the level of stimulation. Specifically for test execution as well as test result evaluation - which are often described as repetitive and boring - ways should be sought to increase the level of stimulation in this work so that it is more enjoyable for testers. Alternatively - as stated in numerous articles on automation – one ought to prioritise boring activities on the list of candidates for automation.

6.5 Security, Anxiety

6.5.1 Basic Concepts

According to widespread opinion and a bunch of psychological theories, one of basic human needs is the need of *security*. When this need is not satisfied, people experience *anxiety*. Anxieties have various levels. *Excessive anxieties* characterise many emotional disorders. A disorder that is of interest here is called *obsessive-compulsive disorder*, which is characterised by mounting tension and anxiety, which is temporarily relieved by giving in to compulsive, repetitive, ritualistic behaviour.

6.5.2 Neurosis of Testing

All this has some bearing to testing. Apart from fully rational motives, part of the explanation why customers want tested and secure systems is emotional: we do not like insecurity and unpleasant surprises. The same applies to vendors: delivering untested products creates anxiety, which we want to avoid. This is probably why the myth of “exhaustive testing” is so strong. It appeals to our deep anxieties. It might be the reason for why the notion of statistic testing and the measure of MTBF are accepted only unwillingly: known, measured risk feels greater than unknown.

6.6 Reviews

6.6.1 Advantages

On one hand, there is plenty of data indicating that potential for quality improvements to be achieved through more extensive use of formal reviews and inspections is huge.

6.6.2 Internal Contradictions

On the other hand, although complex and detailed review techniques exist and are widely discussed, basic contradictions of review have not been solved. You are expected to appraise by means of reading and discussion (provided used method allows for the latter) complex algorithms or design that has taken their author weeks to produce.

6.6.3 External Impediments

There exist a number of external impediments to effective reviews as well. Whatever is said about team spirit, organisations recognise, promote and reward employees on the merit of their own achievements, not on how good reviewers they are.

6.6.4 Group Dynamics

6.6.4.1 Definition

Group dynamics is the study of what happens in groups, with emphasis on power, leadership, cohesiveness and decision-making.

6.6.4.2 Negative Aspects of Group Dynamics

Group pressure influence members to adjust their views to conform with those of the group, as the famous *Asch conformity experiment* amply demonstrates.

Group solidarity develops easily and may cause the development of goals not concurrent with the goals of a larger organisation.

“*Groupthink*” appears when the need to achieve consensus takes priority over the motivation to obtain true knowledge and make optimal decisions. There are many examples of disastrous decisions taken by competent groups because of this phenomenon.

Group polarisation is the tendency for the individuals in a group to take more extreme attitudes as group members than they held individually.

Generally, there are many negative consequences of acting in groups; whereas literature on reviews generally emphasises gains, which is a dangerous fallacy.

6.6.5 Problem Solving: Algorithms and Heuristics

The basic contradiction of review process is that *heuristics* is used to verify *algorithmic* designs. I have no proposal how to eliminate this serious cognitive flaw. One cannot help wondering whether one reviewer studying the object of review for a week would not do a much better job than seven people riffling through it during one day and making a spectacle of pretending to understand!

6.6.6 Personal Interests and Goals (Holland's Theory)

Holland divides occupational interests into six basic groups: *realistic* (engineering, outdoors), *investigative* (science), *artistic*, *enterprising* (sales, politics, PR), *social* (work with people in helping ways) and *conventional* (organised work like clerical or accounting). According to Holland, people seek tasks that match their interests, and avoid tasks that do not.

Serious consideration should be given to the possibility of choosing reviewers (perhaps even “professional reviewers”, whose main task would be reviewing?) according to their occupational preferences, not only their technical skills.

6.6.7 Conclusions

Let organisational and social psychologists critically assess existing review techniques. Advocate experimental studies in this area.

6.7 Why is QA Boring?

6.7.1 Some Possible Explanations

One possible explanation may be the same as in the chapter “Constructive Work and Motivation” above: that QA measures are in a sense disruptive for creative work with its intrinsic gratification and therefore create negative emotions.

Another explanation could be that QA-related activities are repetitive and routine, therefore boring.

Third explanation could be based on Holland's theory (see 6.6.6). QA-work requires perhaps “conventional interests”, while developers have realistic interest?

However, the point is not in inventing numerous more or less plausible explanations, but to provide tool for structured analysis.

6.7.2 Hackman's Model: Description

Hackman's model of work design could be useful for this purpose. This model describes job content using five dimensions. For a particular job or task, scores it gets on skill variety, task identity and task significance scales define together the *meaningfulness of work*. The level of autonomy defines experienced *responsibility for outcome* and existing feedback defines *knowledge of the results*.

6.7.3 Hackman's Model: Application

Neither detailed description of Hackman's model (one of many competing models within organisational psychology) nor the solution of the boredom dilemma is my goal here. Using Hackman's theory as an example, I want to illustrate how much can be gained by applying established psychological theories for the analysis of numerous problems in QA and testing.

6.7.4 How to Enhance "Job Quality"

The model contains rules on how to change job dimensions in order to obtain higher job quality, or higher M.P.S. They encompass combining tasks (increases skill variety and task identity), using natural work units, establishment of client relationship (influences skill variety, autonomy and feedback), "vertical loading" (closing the gap between doing and controlling – this really has strong relationship with actual QA-problems) and improving feedback channels.

7 Psychology of Usability

7.1 Current Status

7.1.1 Real Engineers Read Binary Dumps

For many engineers, "psychology of usability" or "user-friendliness" seems esoteric, fuzzy subject, with no practical application.

7.1.2 Hostile Technology

A growing number of devices become increasingly user-unfriendly, devilishly difficult to use intuitively. Accidents, routinely attributed to "human error", happen in reality because erroneous design of the user-interface has made this accident probable in the first place.

7.1.3 Bad Design is Contagious

As mechanical and software appliances increasingly grow together into embedded systems, the worst habits of software industry of producing bad MMI infect other industries. A car radio produced twenty years ago with its obvious and user-friendly knobs and buttons is much easier to operate than any modern car radio.

7.1.4 Describe a Talon, Ignore the Dragon

The study of user-friendliness and the so-called "usability testing" is today mostly confined to the study of software MMI, especially GUI, and not to the usability of the whole *systems in their social and organisational environment*. Such comprehensive studies are sometimes made (on already installed systems) by academic institutions, but seldom by vendors before delivery. A piece of software is hardly ever friendly or unfriendly itself, but this quality emerges in interaction with how and for what purpose it is used by end-users.

7.1.5 MSc in Usability?

No coherent theory on “appliance usability” exists today. However, some authors have managed to identify a series of important characteristics that influence usability. In the following section I borrow heavily from “The design of everyday things” by D. Norman.

7.2 Checklist: Some Factors of Usability

7.2.1 Intuitive Usage

The form of an appliance should somehow invite the user to perform an appropriate action. Whenever users have to turn a gadget around in order to find some tiny and hidden key to open it or switch it on, this factor is not realised properly. Example: find a switch to turn on a copier.

7.2.2 Models

The design of the interface should hide all implementation complexities and support simplified user model instead. Finite-state testing could be used for testing this. Example: buttons to operate telephones.

7.2.3 Visibility

Sensitive areas should be clearly visible, possible actions properly marked. Syntax testing could be utilised for testing or modelling the complexity (perhaps unnecessary and confusing) of the “protocols” (for example, sequences of button pressing) that are necessary for controlling appliances.

7.2.4 Mapping

The location of controls should in some intuitive, “natural” way map to the results or controlled objects. The classical example of unnatural (arbitrary) mapping is stove controls. Almost all consumer electronics appliances today have extremely user-unfriendly mapping between controls and the results of their actions.

7.2.5 Feedback

It must be easy and obvious to see in what mode a device or system currently is, or what the results of one’s actions are. If you press a key, you expect a visible result of your action and no guesswork regarding what is the current status of the device.

7.2.6 Social Pressures

Sadly, end-users have developed a tendency to blame themselves for errors caused by design deficiencies and do not exert enough pressure on vendors. As long as this attitude remains unchanged, there are no immediate financial rewards for producers who deliver better, more friendly design.

7.2.7 Designer Terrorism

A special form of social pressure is what may be called “designer terrorism”. Designers inflict on us designs, which may be aesthetically pleasing but are not functional. An object whose form does not tell us whether it is a telephone, an alarm clock or a kitchen appliance may be original, but definitely not functional and not intuitive.

7.2.8 Human Errors and Natural Constraints

Many designs are made so that they do not allow for users’ errors. The “cancel” options are missing or insufficient, and error consequences are unreasonably catastrophic.

7.3 Conclusions

Companies engage at present in futile competition in unnecessary features that are not really required by customers, even if today’s customers are conditioned to declare they wish

them. At the same time there seems to exist a huge potential of hidden demand for devices that are intuitive and not frustrating.

The connection between desirable design features on one hand, and QA and testing on the other hand, is that it is possible for QA to influence (during requirement analysis) the design process in this direction. Testing could act as an agent of improvement by developing and using methods for broader usability testing.

8 References

- Agresti, A., & Finlay, B. (1986). *Statistical methods for the social sciences*. San Francisco: Dellen.
- Beizer, B. (1984). *Software system testing and quality assurance*. New York: VanNostrand Reinhold.
- Beizer, B. (1995). *Black-box testing: techniques for functional testing of software and systems*. New York: John Wiley & Sons.
- Berne, Eric, M. D. (1978). *Games people play: the psychology of human relationships*.
- Festinger, L. (1957). *A theory of cognitive dissonance*. New York: Harper
- Gilb, T., Graham, D., Finzi, S. (1993). *Software inspection*. Perseus Pr.
- Hackman, R. (1991). *Work design*. In: R. Steers and L. Porter *Motivation and work behaviour*. McGraw-Hill.
- Hetzl, B. (1988). *The complete guide to software testing*. Wellesley: QED Information Sciences.
- Holland, J.L. (1973). *Making vocational choices: a theory of careers*. Englewood Cliffs, New Jersey: Prentice Hall.
- Janis, I. L. (1990). *Groupthink*. In: P. Chance and T.G. Harris *The best of psychology today magazine*. New Yoirk: McGraw-Hill.
- Kaner, C., Falk, J., Hung Quoc, N. (1993). *Testing Computer Software*. New York: Van Nostrand Reinhold.
- NFI Quality A/S. (1997). *TickIT auditors'training course* (course material).
- Norman, Donald A. (1988). *The design of everyday things*. New York: Currency Doubleday.
- Reber, Arthur S. (1995). *Dictionary of psychology*. Penguin Books.
- Rubin, J. (1994). *Handbook of Usability Testing*. New York: John Wiley & Sons.
- Schweigert, Wendy A. (1994). *Research methods & statistics for psychology*. Pacific Grove: Brooks/Cole Publishing Company.



The euro project for I.S.

- Conflict with Y2K
- Business Requirements
- Project management
- Quality assurance

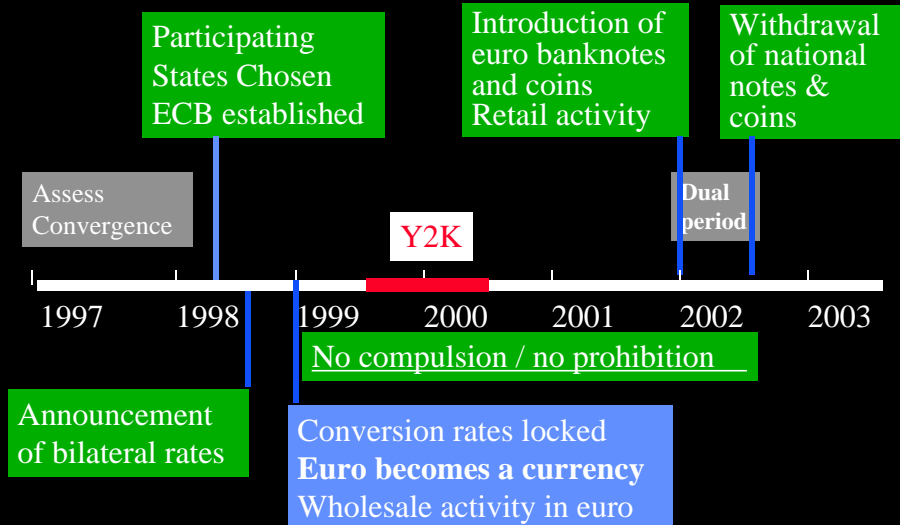
Patrick O'Beirne
pobeirne@sysmod.ie

Systems Modelling Ltd.

10/14/98 1



Timetable for the euro

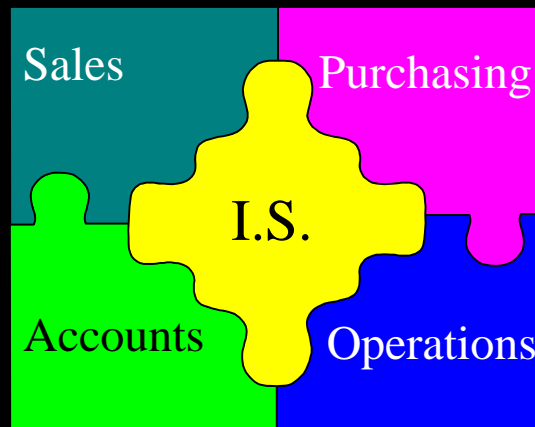


Systems Modelling Ltd.

10/14/98 2



The IS Manager's view



Project planning

- ◆ Strategic preparation
- ◆ Project Team
- ◆ Trading partners synchronisation
- ◆ Information systems changeover
- ◆ Overlap with Year 2000 project
 - Share scarce resources
 - Plan together, deliver separately
- ◆ User Training





Year 2000 & euro projects

- Year 2000
 - Maintenance project
 - Fixed deadline
 - No functionality change
 - regression tests
 - future date testing
 - No fallback
- Euro changeover
 - Strategic decisions
 - Transition period
 - New functionality user training
 - regression tests
 - New func. testing
 - Fallback

Combination may exceed management capacity



Y2K & Euro conflict

- “One of the worst public policy decisions in human history” (Capers Jones)
 - "Resource Conflicts Between the Year 2000 and Euro Currency Software Problems" (Year 2000 Journal, Jan/Feb 1998)
 - “Dangerous Dates for Software Applications”
<http://www.comlinks.com/mag/ddates.htm>
- “EMU & the Y2K Gamble” (Peter de Jager)
 - The added burden is unreasonable, and considering the consequences of failure, foolhardy
 - <http://www.year2000.com/y2kgamble.htm>



Questions

- Raise your hand if you have a high degree of confidence in your ability to deliver the Y2K project on time
- Raise your hand if over the past three years, you have delivered 100% of your applications on time
- 80% .. 60% ... 40% ?



Gartner Group

- “Billions of lines of code to be changed
- Estimated \$1.10 to fix each line of code
- Adding the euro character to font sets, printer drivers, keyboards, ATMs, cash registers
- We're talking about taking all of this information and changing it all, it's crazy”



Giga Group

- “Almost every system falls over when it has to do triangulation, the computer starts to choke
- all systems will have to be able to account for six significant decimal places
- sound like nit-picky little problems but each one of them will kill you
- if a company converts its history, the numbers become almost meaningless”



Businessweek

- GETTING READY FOR THE EURO BUG
- A scary fact: That transition period, designed to coddle consumers, poses all sorts of frightening conundrums for computer systems
- The challenges range from the mundane ... coming up with a computer key for the euro symbol,
- to inserting the most abstruse mathematics into software.
- <http://www.businessweek.com/premium/13/b3571164.htm>



Manage the risk

“Managing the risk in euro currency conversions”

- Patrick O’Beirne, Cutter IT Journal June 1998
- Most processing currency-independent
- Look for input , exchange & output functions

<http://www.iol.ie/sysmod/euroriskarticle.htm>



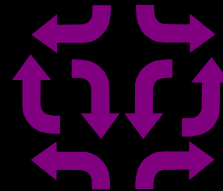
Manage the Requirements

- Opportunities for new products
- Opportunities marketing / customer service
- What Customers want (euro-friendly?)
- What Suppliers will do (power balance?)
- What Competitors could do
- What your Bank can do for you
- What legal requirements will be imposed
- Accounting standards



Changeover options

- Manual (Single currency + calculator)
- Parallel (Not integrated)
- Modify (Risk of introducing more problems)
- Replace with multi-currency
- Dual base currency?
- Timing



Systems that need conversion

- All financial systems, Treasury mgmt
- EDI & Dependence on external information providers - stock exchange feed, bureau services
- Inventory, work-in-progress, Assets
- Costing, Budgeting, DSS, Planning
- End-user developed applications
- Cash registers, specialised hardware
- Vending, note & coin handling machines



Conversion

- Big Bang or Gradual
- End of year considerations
- Timing w.r.t. partners changeover
- Time lags in account changeover of open items
orders /deliveries /invoices /payments /statements
- Print hard copies as audit trail evidence
- Historical databases & normalisation



Other issues

- Customer special price lists
- Revaluation of thresholds / bands
- Payroll, taxes, staff education
- Dependence on external service providers
of information systems - e.g. stock
exchange feed, bureau services



Web Sources

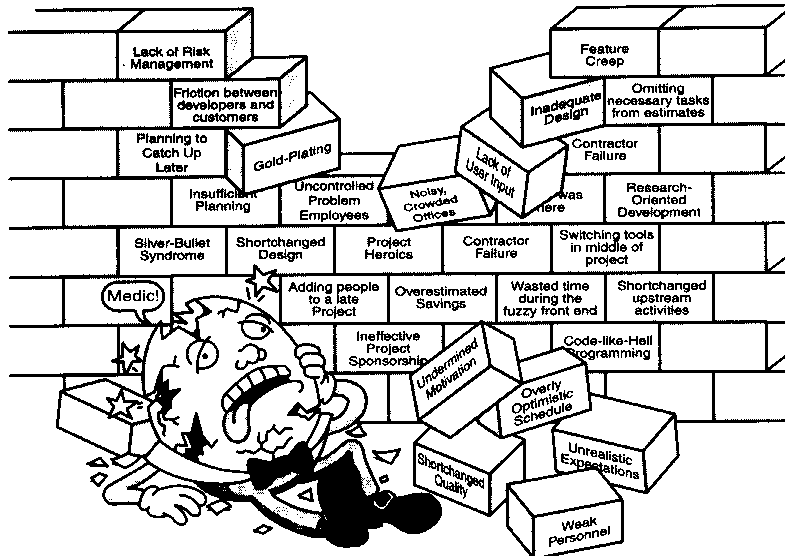
- Information Systems guidelines at
<http://www.ispo.cec.be/y2keuro>
- Federation of European Accounting Experts
<http://www.fee.be>
- Frequently Asked Questions answered
<http://www.iol.ie/sysmod/eurofaq.htm>
- Articles on EMU and I.T. :
– <http://www.iol.ie/sysmod/emu.htm>



What we know

- ◆ 80% of s/w projects are late
 - The Year 2000 project must not be late
 - The Euro changeover may be negotiable
- ◆ 10% of new code contains defects
 - 10% of the fixes will too
 - ◆ etc. etc.





Systems Modelling Ltd.

10/14/98 19



Software quality

- ◆ Preventing errors is cheaper than fixing
- ◆ Code review is the most effective
- ◆ Do you believe “Quality is free”?

Systems Modelling Ltd.

10/14/98 20

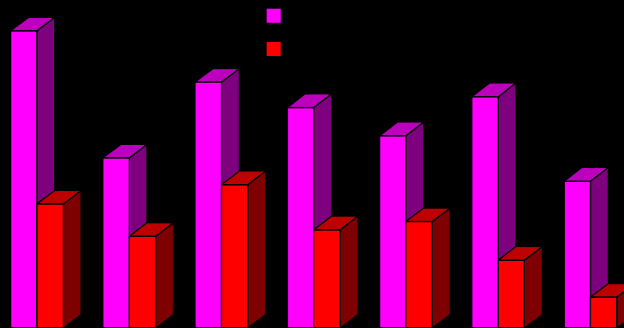


The Personal Software Process

- ◆ Results from a course given at the Centre for Software Engineering, DCU, Dublin
 - Defects reduced by 75%
 - No change in productivity
 - Therefore quality is free!
 - How is it done? - Code Review

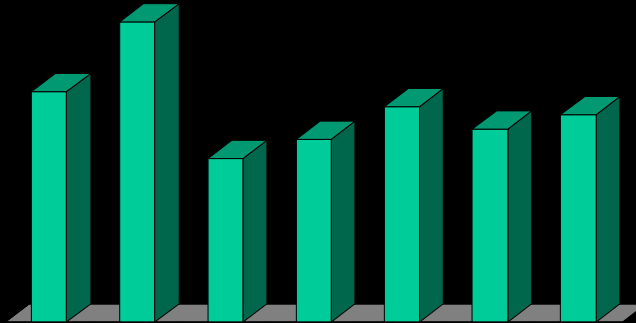


Group average defect rate





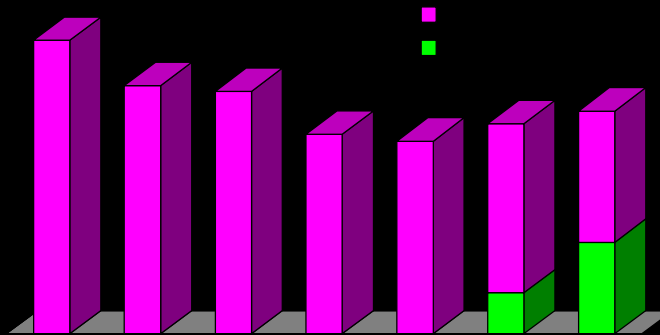
Group productivity LOC/hr



Cost of Quality measures

$$\text{Appraisal} = (\text{Des.Rev} + \text{Code Rev}) / \text{Total time}$$

$$\text{Failure} = (\text{Compile} + \text{Test}) / \text{Total time}$$





Conclusion

- The only way to meet the Year 2000 and euro challenges is to do it right, on time.
- Manage the euro requirements
- Manage Y2K : Replace/Repair/Retire
- Define the process
- Measure the performance - pilot first
- Re-plan, re-budget, re-assess.



Systems Modelling Ltd. Suite 2, Villa Alba
Tara Hill, Gorey, Co. Wexford, Ireland
Tel: +353 55 22294 Fax: +353 55 22297
<http://www.iol.ie/sysmod>

The logo for ps_testware features the text 'ps_testware' in a blue serif font. The 'w' is stylized with a red checkmark shape integrated into its right side. The background of the slide has a faint, large red checkmark watermark.

Brusselsestraat 125
B-3000 Leuven
Tel.: +32-16-310880
Fax: +32-16-310888
e-mail: ps_testware@compuserve.com

putting method into practice

Agenda

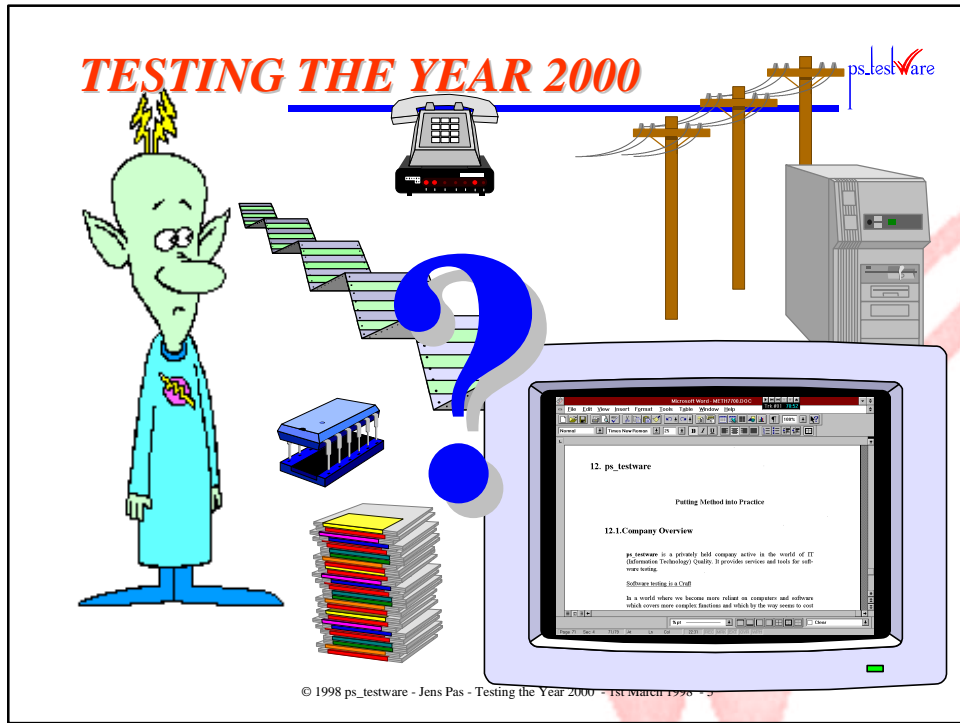
A small version of the ps_testware logo, consisting of the text 'ps_testware' in blue with a red checkmark on the 'w'.

- **ps_testware**
- **Introduction**
- **Conversion model**
- **Implementation model**
- **Metrics & Reports**
- **Tools**
- **Questions**

- 6** • **Started in 1991**
- 5** • **1993: Tools & technical services**
 - Tool Training
 - Coaching
- 6** • **1995: Services around Methods**
 - Consultancy
- 20** • **1996: Software Testing Services**
 - Test Assignments
 - Test Plan
 - Test Report
- 20** • **1997: Software Testing Services Suite**
 - Test Assessments
 - Y2K training
 - ...

20 + 20?

- **Tools: Training and Implementation**
 - Cyrano-Test (V-Test)
 - MERCURY: (WinRunner, TestDirector)
 - Rational products (Preview, RequisitePro,SQA,...)
- **Services**
 - Training
 - Coaching
 - Consultancy
 - Outsourcing



Agenda

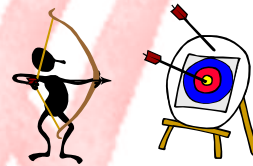
- ps_testware
- **Introduction**
- Conversion model
- Implementation model
- Metrics & Reports
- Tools
- Questions

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 6

Goal

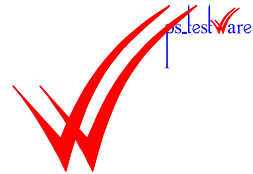


- **Real Time representation must work for past, present and future situations.**
- **Business as usual**



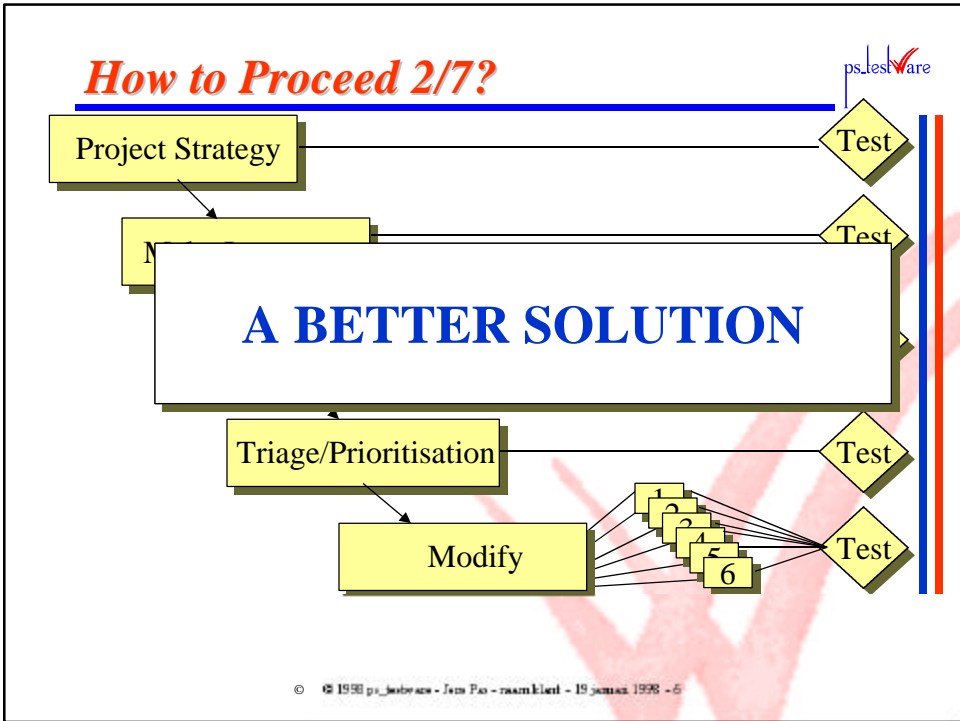
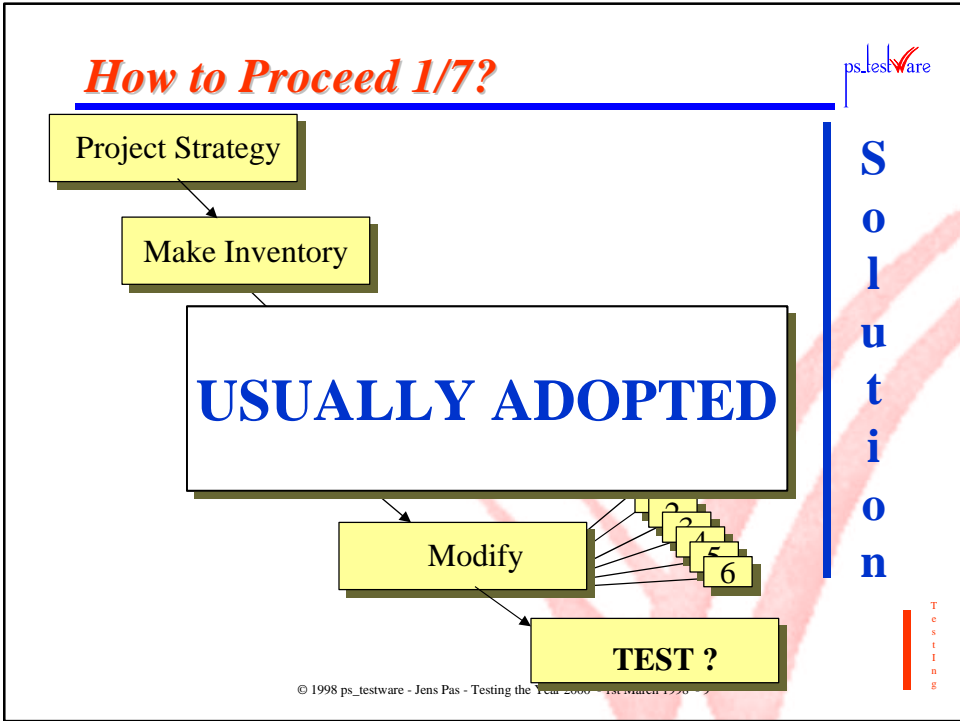
© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 7

ps.testWare

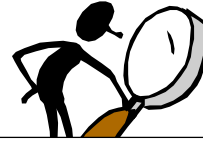


The Method

© 1998 ps_testware - Jens Pas - naam klant - 19 januari 1998 - 8



Rapid Conversion Testing (RCT) Ô

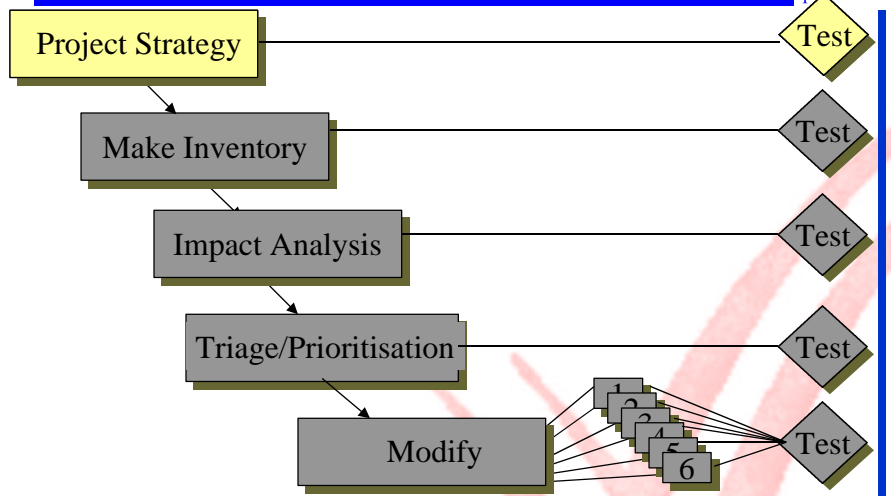


MAKE SUPER EFFICIENT TESTS :

“the right action at the right time by the right actor”

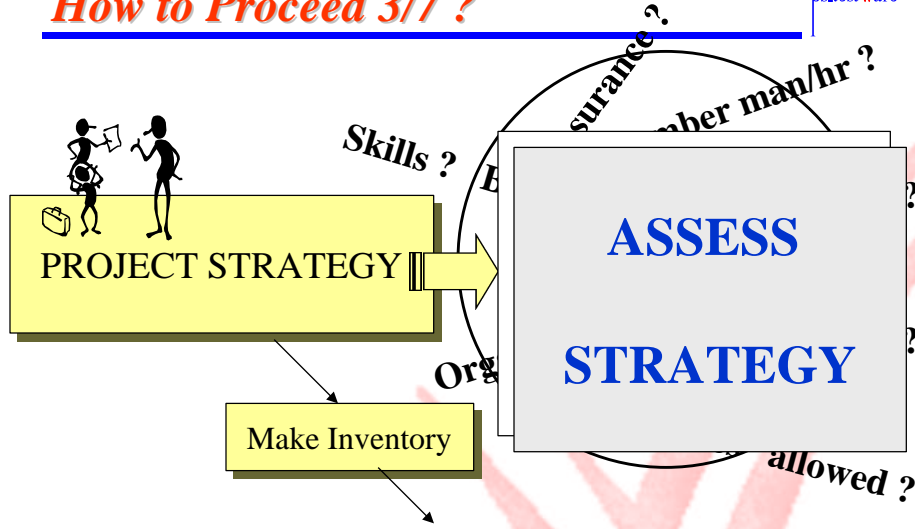


Project Strategy



How to Proceed 3/7 ?

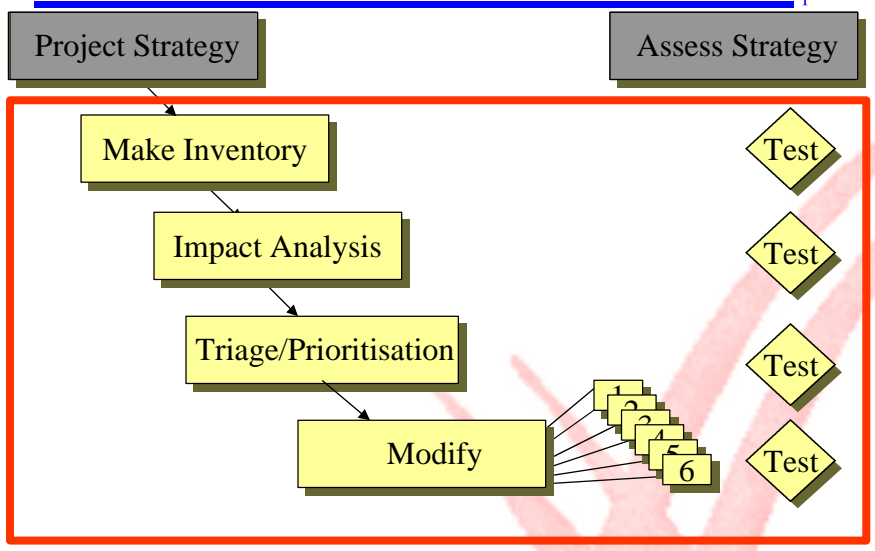
ps.testWare



© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 13

A New Model

ps.testWare



© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 14

Agenda



- ps_testware
- Introduction
- **Conversion model**
- **Implementation model**
- **Metrics & Reports**
- **Tools**
- **Questions**

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 15

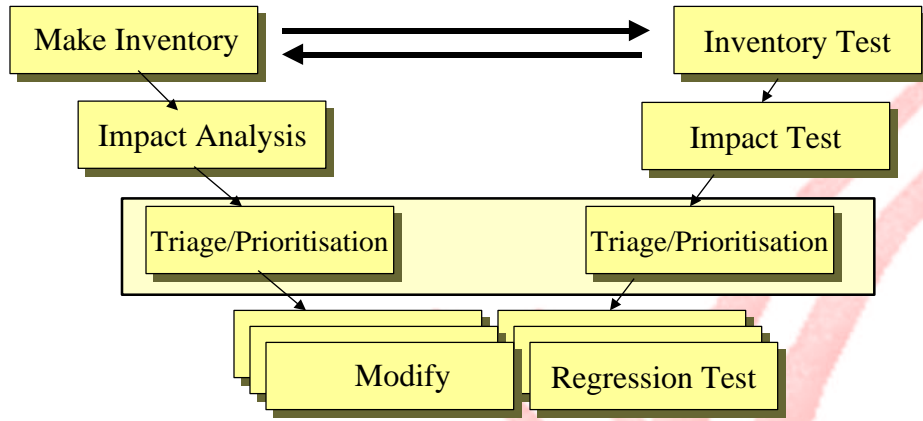


The ps_testware method for conversion testing (Y2K)

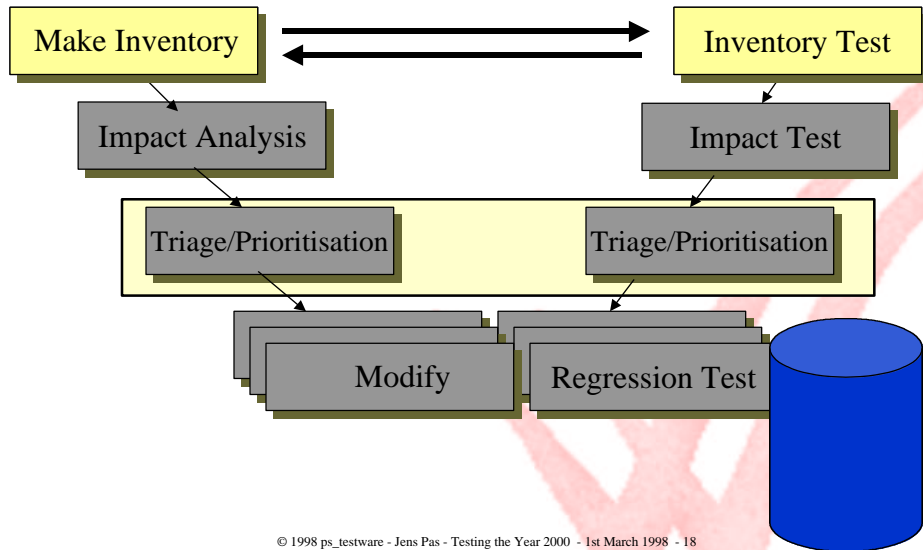
The **V** Model™

© 1998 ps_testware - Jens Pas - naam klant - 19 januari 1998 - 16

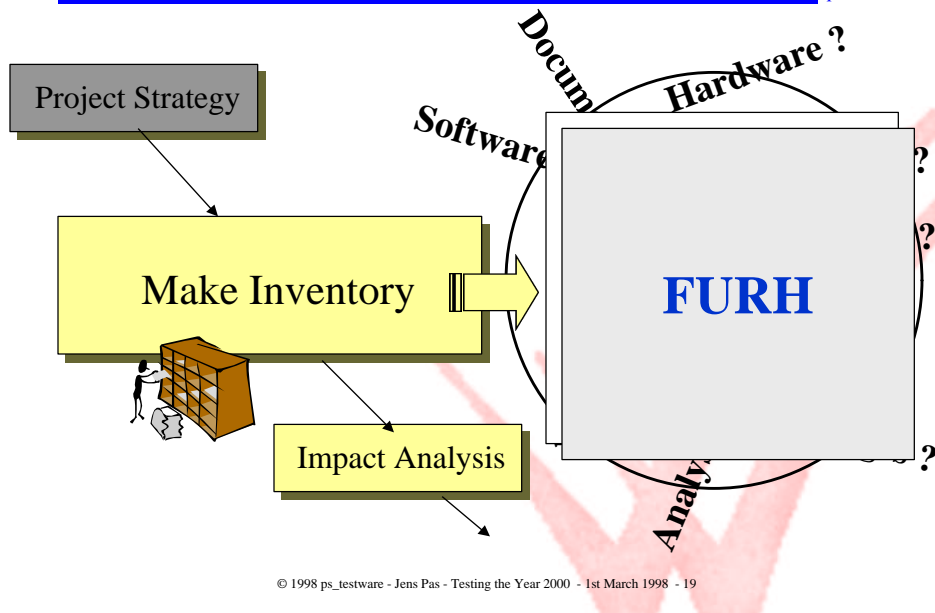
The V Model^Ô



The V Model^Ô



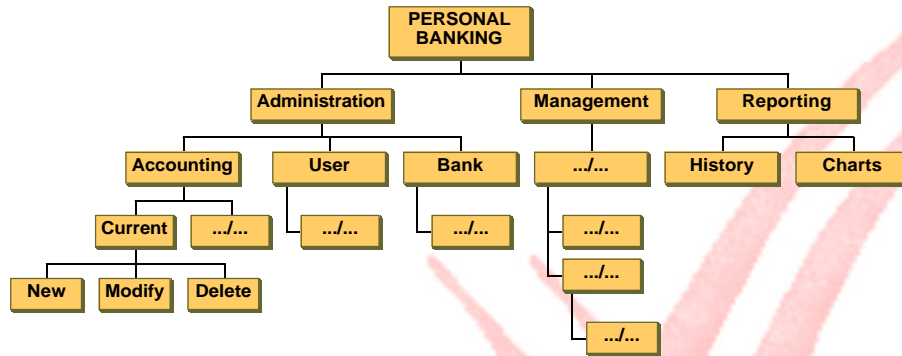
How to Proceed 4/7 ?



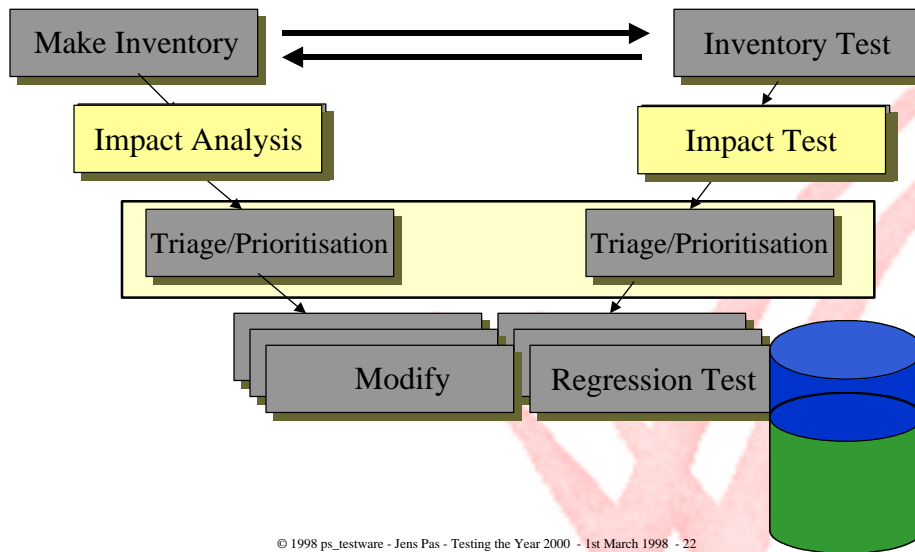
FURH ???

- **Functional Requirements**
 - application functionality the user requires to perform his daily business
- **User**
 - the person who will use the application, for whom the software is designed
- **Hierarchy**
 - a representation in a formal and structured way that takes order and sequence into account

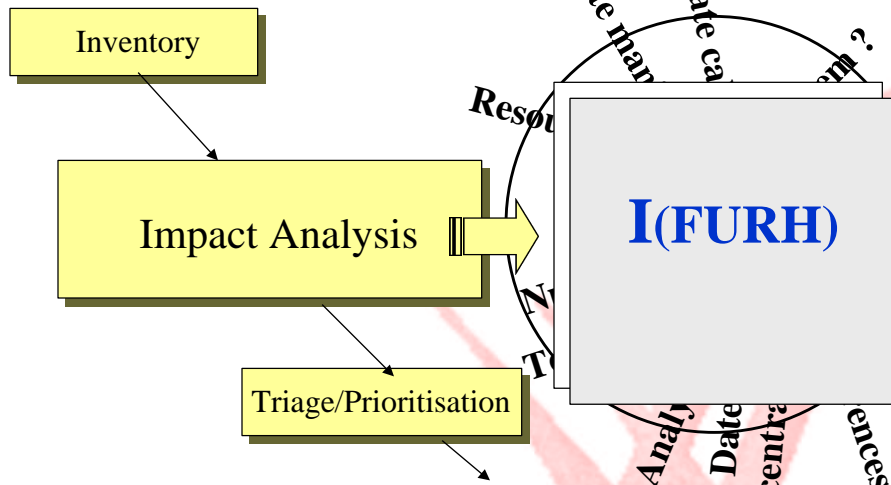
FURH (continued)



The V Model

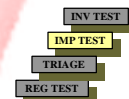


How to Proceed 5/7 ?

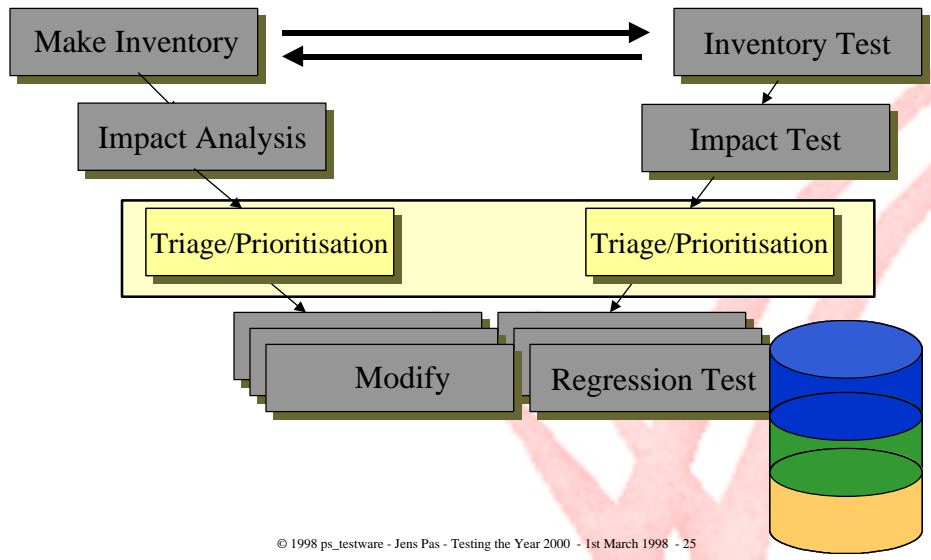


I(FURH) ???

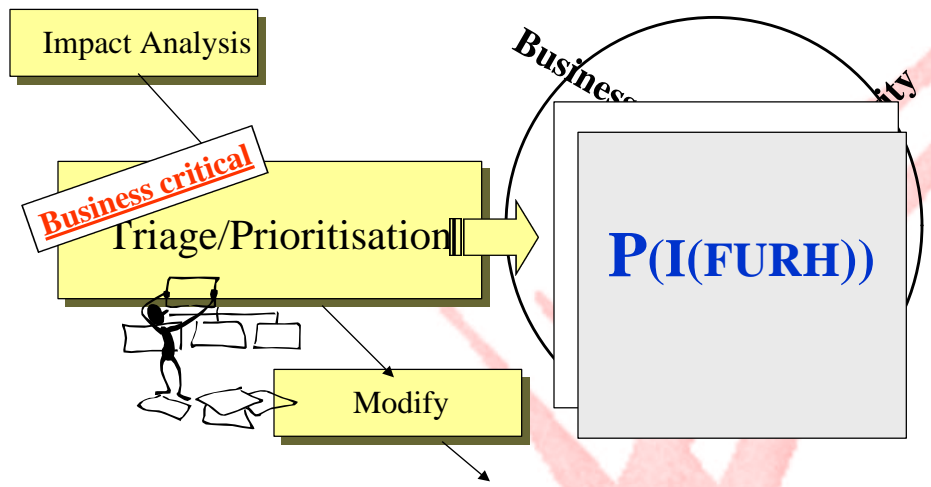
- **Business activity that uses or requires dates**
- **Tagged FURH as verification of technical impact analysis**
- **Business activities at the light of :**
 - **criticality**
 - **cost**
 - **Y2K concern**



The V Model

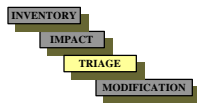


How to Proceed 6/7 ?



Triage/Prioritisation

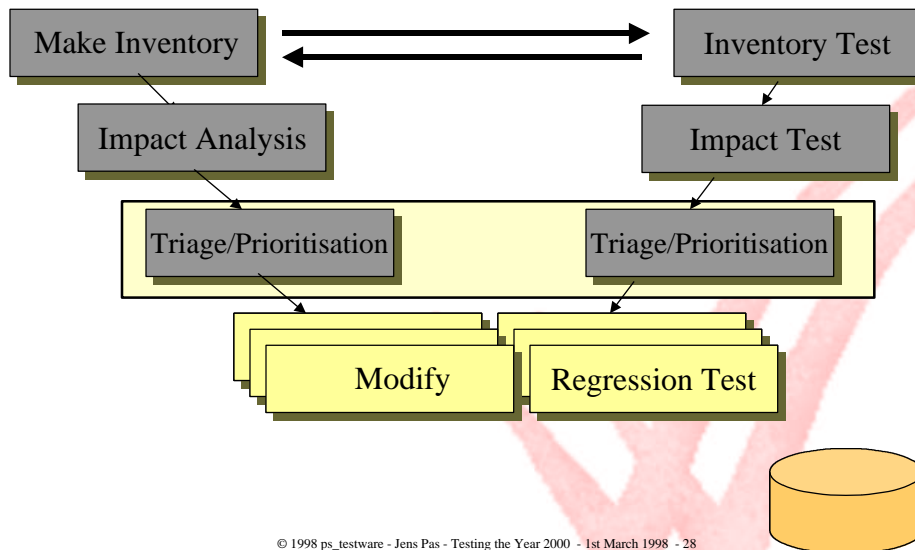
- **Business critical**
 - **Now** (no delay, do it !)
 - **Later** (not so critical, can wait some time)
 - **Never** (no more in use, drop it)



© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 27

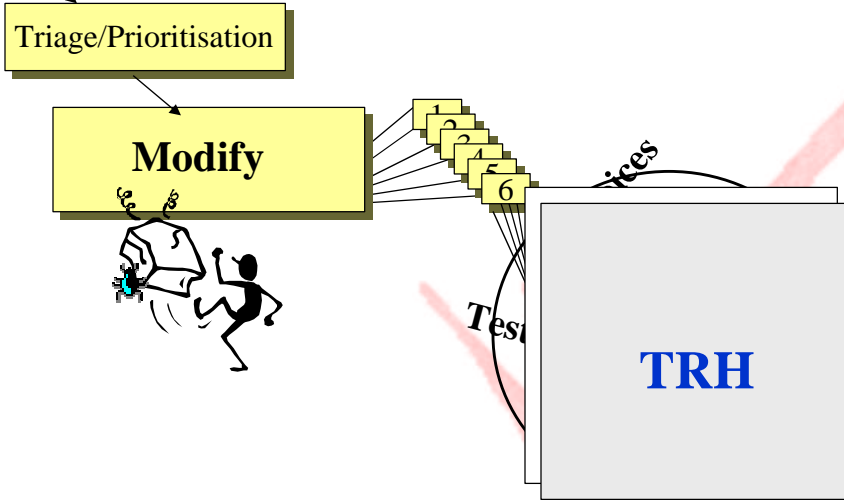


The V Model

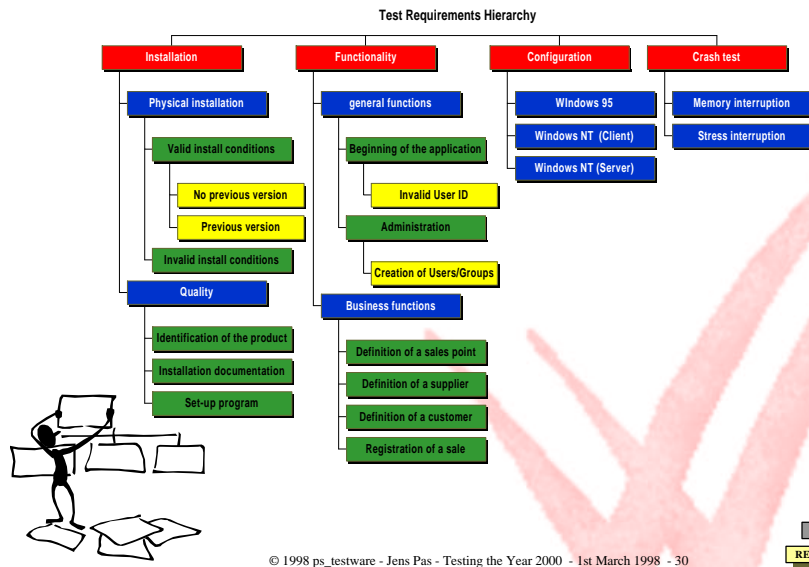


© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 28

How to Proceed 7/7 ?

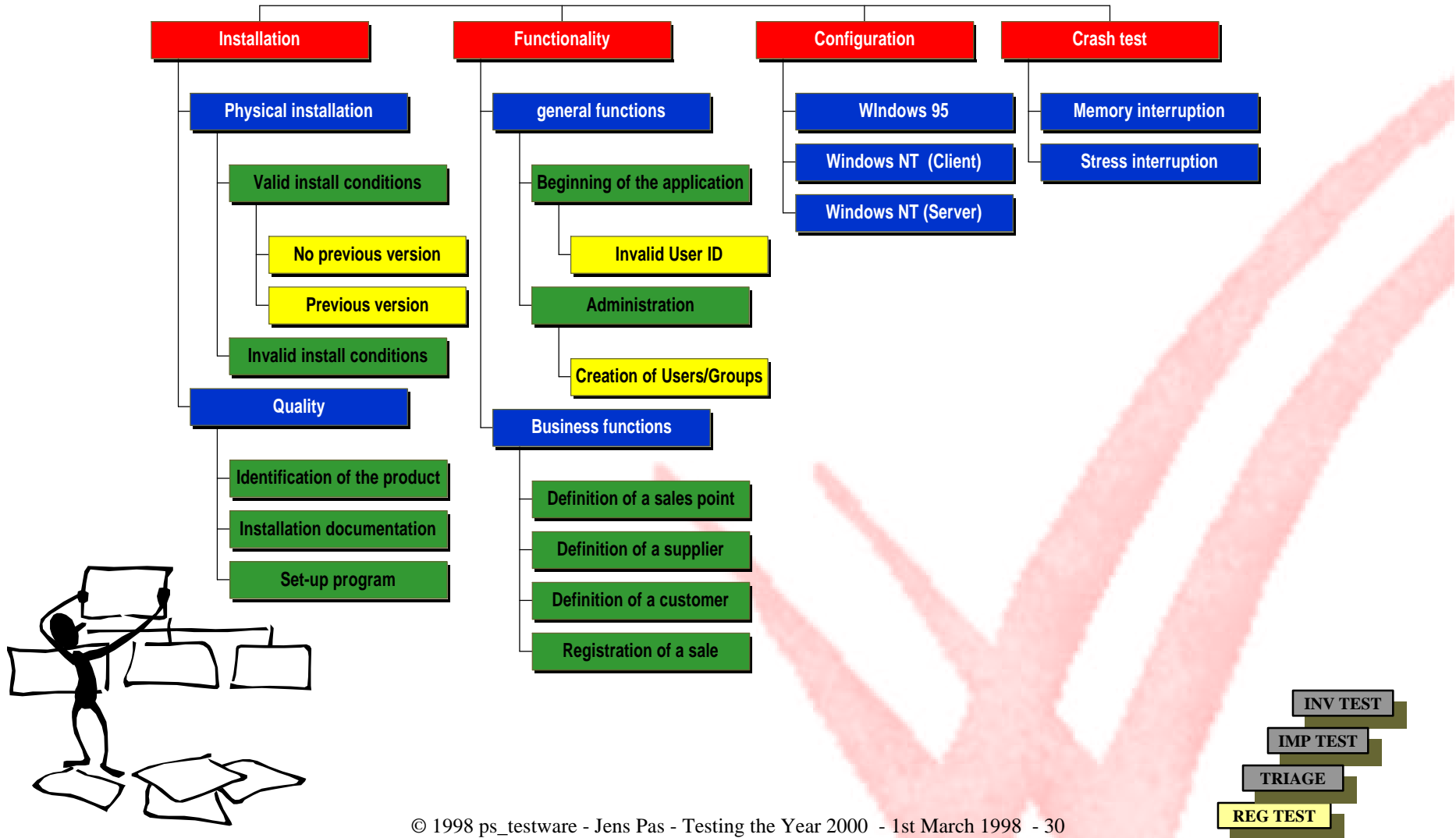


Test Requirements Hierarchy



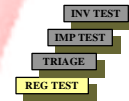
Test Requirements Hierarchy

Test Requirements Hierarchy

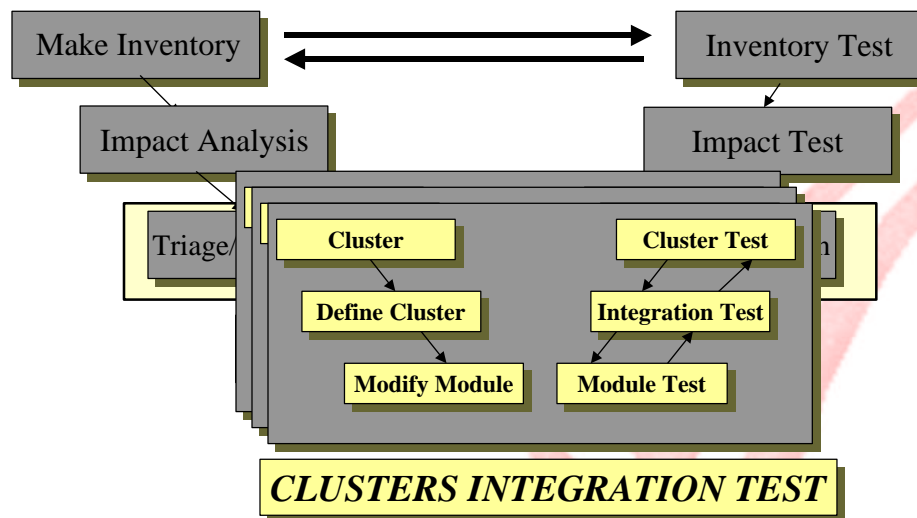


Practical Testing Aspects

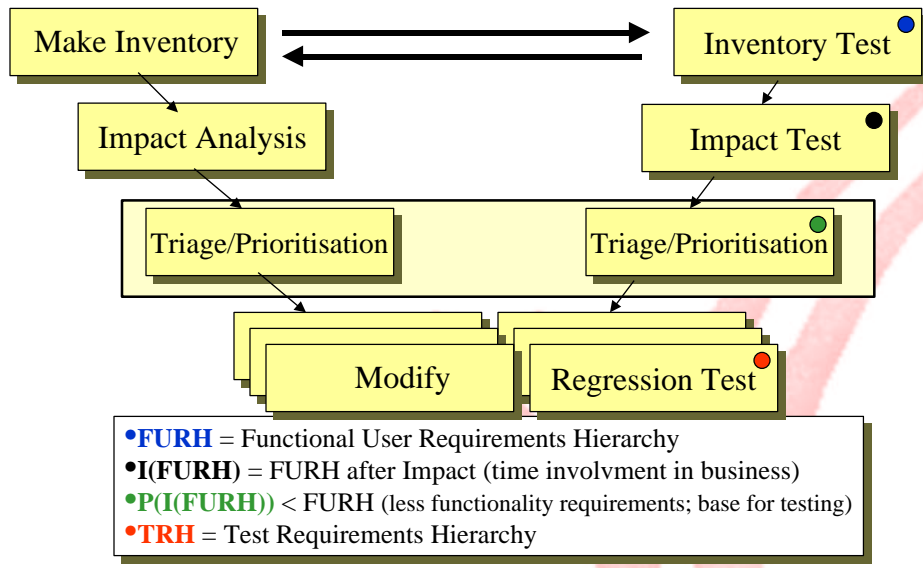
- Do not **only** use “real” data for testing Y2K
- Equivalence partitioning vs. whole database
- Boundary values and intermediate
- Do not test everything
- Code coverage complementary to FURH and TRH
- .../...



The V Model



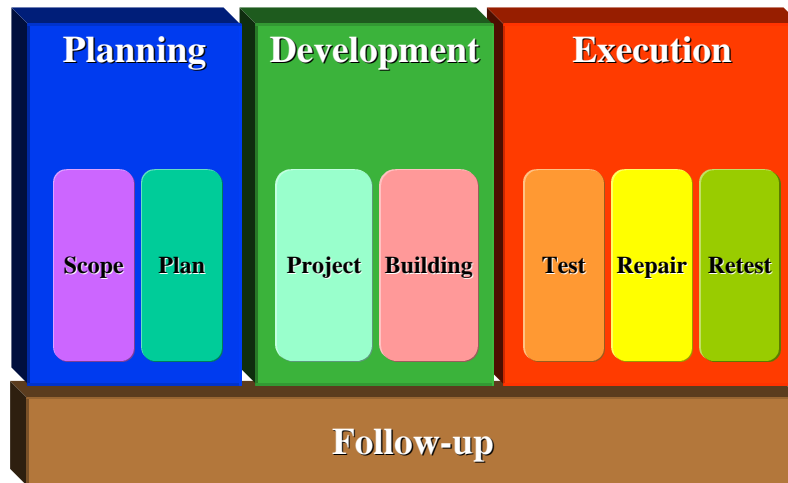
The V Model^Ô



Agenda

- ps_testware
- Introduction
- Conversion model
- **Implementation model**
- Metrics & Reports
- Tools
- Questions

Implementation model: 3 phases



Agenda

- ps_testware
- Introduction
- Conversion model
- Implementation model
- **Metrics & Reports**
- Tools
- Questions

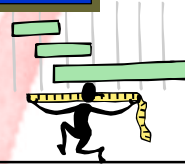
Metrics

– % Passed Product

Number of procedures designed
Number of procedures scripted
Number of procedures executed

– Throughput

The pace at which the process produces solved errors



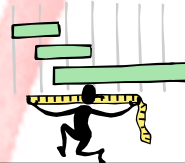
Metrics

– Test maintainability (%)

$$TO = \left(1 - \frac{\text{Man-days test maintenance}}{\text{Man-days test design}} \right) \times 100$$

– Test waste

$$TV = \frac{\text{Man-days test repair}}{\text{Man-days test creation}} \times 100$$



Weekly Progress Report

Weekly test progress report

Project ID	CCC/nnn
Project title	This is an example
Start date	30/03/1997
End date	31/12/1997

Date	24/04/1997
weeknr.	199717
Laptime spent	9%

1. Goal achievement

Test Requirement Coverage

Number of test requirements	102
Design coverage	72%
Test coverage	72%
Passed product	70%

2. Test quality

Effectiveness

Are we doing the right thing ?

	This week	Total	index	Plan (hours)	Total (T days)	Plan (T days)
Number of defects	46	8				
Hours of test preparation	0,0	5,0	13%	40,0		
Hours of test planning	6,0	15,0	13%	120,0		
Hours of test design	4,0	24,0	9%	280,0		
Hours of testing	0,0	23,0	14%	160,0		
Hours of test repairing	4,0	15,0	38%	40,0		
Hours of defect tracking	2,0	6,0	8%	80,0		
Hours of Maintenance	9,0	14,0	18%	80,0		
Hours of Overhead	1,0	15,0	19%	80,0		
Total	26,0	117,0	13%	880,0	18	110
					16%	

Efficiency

Are we doing the thing right ?

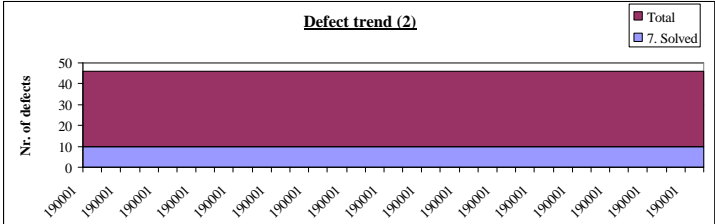
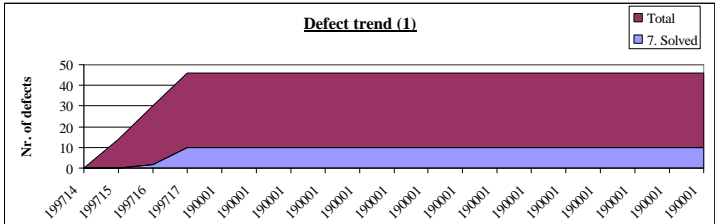
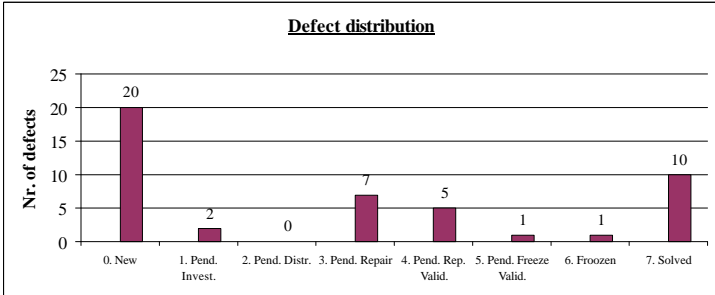
	This week	Total	index	Plan
Average hours/day	6,5	6,7	84%	8
Test maintainability	-125%	42%	58%	71%
Test spoilage	15%	13%	282%	5%
Overhead	4%	13%	141%	9%
Defect throughput (per day)	0,40	0,40		
Defect detection rate (per day)	1,84	0,32		

Weekly Progress Report



3. Constraints analysis

0. New	20
1. Pend. Invest.	2
2. Pend. Distr.	0
3. Pend. Repair	7
4. Pend. Rep. Valid.	5
5. Pend. Freeze Valid.	1
6. Froozen	1
7. Solved	10



Relative Quality Measurement SW

- **Defect Removal Effectiveness (DRE)**

The number of solved defects in a release relative to the total number of found errors.

$$\text{DRE} = \frac{\text{\# Solved defects in a release}}{\text{Total \# found errors}} \times 100$$

Agenda

- ps_testware
- Introduction
- Conversion model
- Implementation model
- Metrics & Reports
- **Tools**
- Questions

Type of Tools

- **Testing**
 - **Test & Process management**
 - **Dynamic testing tools**
 - **Static testing tools**
 - **Time manipulation**



Agenda

- **ps_testware**
- **Introduction**
- **Conversion model**
- **Implementation model**
- **Metrics & Reports**
- **Tools**
- **Questions**

Questions

ps.testWare



© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 45

ps.testWare

**Brusselsestraat 125
B-3000 Leuven
Tel.: +32-16-310880
Fax: +32-16-310888
e-mail: ps_testware@compuserve.com**

putting method into practice

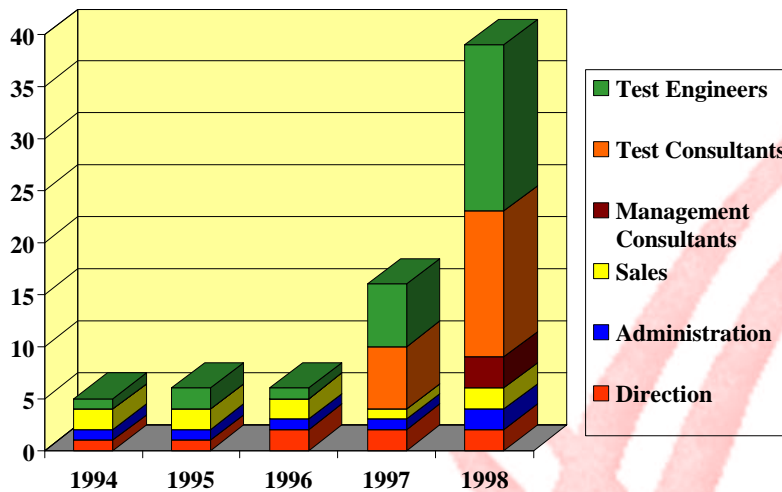
New Projects



- Kredietbank
- Barco Graphics
- Levi's
- Siemens Nixdorf
- Orda-B Antwerpen
- Tessa
- RZG
- GAK
- Cadans
- Exact Maatwerk
- Exact International
- ING Bank
- Bank Card Company
- Janssen Pharmaceutica
- LGT Soft
- Bossart

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 47

Evolution : Human Resources



© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 48

Credo



ps_testware's first responsibility goes to the customers who use our services. Our services must be of high quality and must be a reference for our customers. In line with our primary business, Structured Software Testing, we may not indulge in pressure, quantity or quick profit.

We are responsible to our members, the men and women who work with us. Every member must be respected as an individual and must be rewarded personally and fairly. We must support our members through competent management, an adequate working environment and proper working conditions. Our members must have the means to provide and receive feedback, allow them and the organisation to learn continuously. We must support our members in their family responsibilities. Our actions must be just and ethical.

Our final responsibility is to our stockholders. Our business must make a sound profit. We must innovate and continuously improve our methods and techniques. We must develop new services and implement them effectively and efficiently. We must create reserves to provide for adverse times. Our stockholders must receive a fair return on their investments.

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 49

Services



- **Training and Support**
- **Test Process Management**
- **Coaching “on the job”**
- **Outsourcing/Consultancy**

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 50

Training & Support



- **Test Methods & Techniques**
- **Test Automation**
- **On-Site Support**
- **Telephone Support**

We teach you how to do it

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 51

Test Process Management



- **QA State of Mind**
- **Testing Strategy**
- **Productive Testing**
- **Communication**
- **Test Life Cycle**

How it should be done

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 52

Coaching



- **Test Planning**
- **Test Engineering**
- **Test Procedure Development**
- **Test Case Design**
- **Implementing Tools**

We help you do it

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 53

Outsourcing



- **Testing**
- **Test Project Management**
- **Acceptance Testing**
- **Volume/Load/Stress Testing**
- **Test Consulting**

We do it for you

© 1998 ps_testware - Jens Pas - Testing the Year 2000 - 1st March 1998 - 54

ps_testware

Brusselsestraat 125

B-3000 Leuven

Tel.: +32-16-310880

Fax: +32-16-310888

e-mail: ps_testware@compuserve.com



putting method into practice

BT Laboratories



*Richard Tinker
Ron Walters*



(c) British Telecommunications plc, 1998

System Integration and VV&T Strategies are not just Test Plans

- How we use Web to manage our System Integration and VV&T Activities
- The Processes we use
- How we develop Test Strategies
- How it is easy to produce a *Test Plan* not a VV&T Strategy



(c) British Telecommunications plc, 1998

System Integration and VV&T Strategies are not just Test Plans

- The paper shows some examples of the documents and check lists that we link to
- The most important factor is the need to make the right way of doing something easy
- The main area of interest here is the need for clear VV&T Strategies
- These are not just Test Plans



(c) British Telecommunications plc, 1998

Where we are

- All our process documents are WWW based
- These documents are accessible by all our people and most of our Customers (other BT Divisions)
- The Top Level Document is the Integration Handbook



(c) British Telecommunications plc, 1998

Typical Inputs

Support Strategy
Integration Strategy
VV&T Strategy
Designs
Deployment Design/Plan
Configuration Management Plan
Quality Plan
Requirements Traceability
System Test Plan
Design Review Report
Security Policy Documents
Cost Estimate
Project Requirement Definition
Production Plan
Information Request Note
Scope Statement
Tested System
Error List
Build List
Test Reports
Metrics
Updated ECMS
System Documentation
Integration Plan
User Documentation

Typical Actions

Maintain ECMS Model;
Build & Integrate System;
Baseline System;
Manage Configuration on ECMS;
Produce Installation Guide;
Produce Metrics Report;
Security Activities

**System Build
& Integration**

Typical Checks

Documentation is available;
Error List;
Production Plan;
Compliance With Security Requirements;
Test Reports;
Build List;
Adequacy Of Test Cover;
Under Configuration Management;
Actions From Previous Quality Gates;
Cost Estimate;
Scope Statement;
Project Requirement Definition;
Information Request Note.

Typical Outputs

Updated ECMS
Scope Statement
Error List
Integrated System
Test Report
Metrics Report
Build List
System Documentation
User Documentation
Security Policy Document
Cost Estimate
Production Plan
Project Requirement Definition
Information Request Note
Design Review Report
Installation Guide



(c) British Telecommunications plc, 1998

VV&T Strategy

The aims of the VV&T exercise

the political and budget constraints

the aims of the business case

the quality goals

how the aims of the business case will be addressed by the verification and validation process

the relationships between VV&T, requirements, design, development, project management, quality assurance and configuration management

the VV&T teams and defining roles, responsibilities, and authorities

measurable objectives for the quality and testing required

mapping of the VV&T activities onto the chosen lifecycle - this may also include systems integration stages

schedule of the VV&T activities with respect to the requirements, design, development and release schedule, and agreed with the project manager

etc....



(c) British Telecommunications plc, 1998

Important aspects of the Strategy

- The SI & VV&T Strategy is the guiding principle on how efficient use of test resource will produce a quality product that meets the objectives laid out in the business case
- The Generic Document tries to ensure that all possible problems are considered and that any defects are found as early as possible in the lifecycle.



(c) British Telecommunications plc, 1998

Important aspects of the Strategy

- The most important aspect of the strategy is to ensure that the objectives of the business are the things addressed by the VV&T work. The political and budget constraints are addressed
- It needs to ensure that the right things will be done to find problems early, it is difficult to integrate systems that contain major functional errors
- It should usually be fairly short



(c) British Telecommunications plc, 1998

What we found

- Use of Generic Documents is inconsistent
- Strategies are produced based on the Generic but with little considered thought
- Strategy contains no Strategy!
- The contents clearly show a test plan with test milestones, structure of test cases, test report requirements, etc.



(c) British Telecommunications plc, 1998

Why do Strategies become Test Plans

- Not produced early enough - decisions taken by default
- Project development-led - ignores later lifecycle stages
- Quality goals not openly discussed
- Author is usually a tester not a systems engineer



(c) British Telecommunications plc, 1998

Why do Strategies become Test Plans

- The objectives of the business case may not be clearly defined
- Time and cost pressures lead to short cuts
- The client may not fully understand the role of the VV&T process and dictate the requirements of the testing process



(c) British Telecommunications plc, 1998

Why do Strategies become Test Plans

- Produced under pressure to get a tick in a box
- So they produce a plan for their part of the lifecycle not a true strategy



(c) British Telecommunications plc, 1998

The Way Forward

- Use of web to hold latest techniques, feedback, tool tips
- Make it simple to use
- Educate about value of this approach
- Management and technical Review



(c) British Telecommunications plc, 1998

The Way Forward

- Make sure that Business Goals are identified and testable
- Make it easy to produce a good document
- Project manager education & accreditation
- Integration-led projects



(c) British Telecommunications plc, 1998

The Way Forward

- Review all strategies
- Qualified Reviewers with systems engineering skills
- Produce Strategies at the earliest possible time
- Make it clear



(c) British Telecommunications plc, 1998

The Way Forward

- Strategy is whole lifecycle document
- Make sure everyone in team knows what the strategy is
- Allow time to correct mistakes (YOU WILL MAKE THEM)
- Review result



(c) British Telecommunications plc, 1998

Millenium Testing

- Clear web based guidance and support
- Standard strategy and test cases provided
- Database of all company applications visible to all
- Formal compliance certificates used to record status



(c) British Telecommunications plc, 1998

Conclusions

- Web based process guidance is effective in spreading good practice
- Writing a good VV&T Strategy is a demanding exercise requiring systems engineering skills



(c) British Telecommunications plc, 1998

***System Integration and VV&T Strategies
are not just Test Plans***

***Richard Tinker VV&T Manager
Ron Walters Integration Manager
BT N&S Systems Engineering***

1. Introduction

This paper presents how the Systems Integration Unit in BT's System Engineering Organisation has implemented business-wide Internet based process guidelines for Systems Integration and Verification, Validation and Test (VV&T). The Systems Integration Unit carries out most of the large Integration and VV&T activities for BT's Internal Networks and Systems. It is a unit of over 1000 permanent and contract people.

The paper shows how even with example Generic Strategy Documents as guidance, it is easy to fall into the trap of creating of Test Plans, not true Strategies.

The paper highlights the use of the Internet based process guidelines and background to their creation. It shows how the generic documents were created and the lessons learned from their use. It covers the steps being taken to improve the quality of the System Integration and VV&T strategies being produced across the Systems Engineering unit.

The strategies being created in Systems Integration address BT's steps to ensure that it has taken the right actions to overcome Millennium compliance issues.

2. Process Guidelines

The top-level documents that identify how to carry out System Integration and Validation, Verification and Testing (VV&T) are contained in a number of Internet documents. These documents are available across BT and therefore most of our customers (largely other BT units) have access to them. The use of Internet has enabled us to promote the use of generic document that can be quickly updated to take account of experience of their use and improvements in technology and process.

The Internet documents can also be easily linked to the company wide advice and process documents in areas relating to security, environment, etc.

Integration Handbook

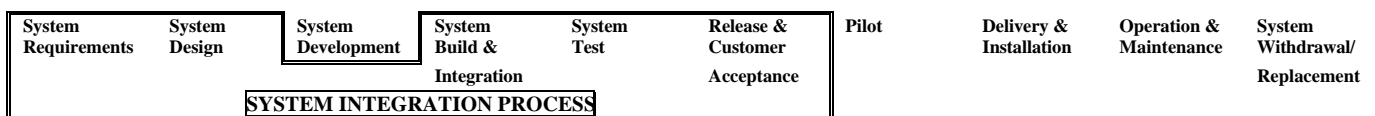
The top-level process document is the *Integration Handbook*, which is an Internet publication (a printed version is available and is titled the Integration Guide).

After the formal approval of the ISO9000/TickIT Quality Management System it became clear that given the diverse range of projects carried out in Systems Integration (SI) that a very formally structured Management System was becoming a burden and not helping to deliver good quality products. The need to reduce this bureaucracy and replace it with easily obtainable, usable guidance from the recognised experts in the organisation led to a small team producing the handbook and then getting it accepted across the Systems Engineering part of BT as the chosen way of working.

The ability to link to very good examples of the right way of doing things enabled Project Managers to choose the appropriate methods of working to suit their Project.

The Integration Handbook contains the main elements of management of an Integration project. It is designed as a guidance manual with links to the company's Quality Management System.

The guidance contained in the Handbook is in two main areas, Lifecycle activities and across-lifecycle processes. The lifecycle stages covered by the Handbook are:



A typical lifecycle stage (taken from the Integration Guide) is shown next:

System Build & Integration

This stage is part of the Development Phase of the BT Project Management Handbook [4]. The purpose of this process is to integrate the system units with each other, producing a system that will satisfy the system requirements. The main inputs to this stage are various sub-systems of known quality and the main output is a version of the final system.

The criteria for this stage are:-

Typical Actions

Maintain ECMS Model;
Build & Integrate System;
Baseline System;
Manage Configuration on ECMS;
Produce Installation Guide;
Produce Metrics Report;
Security Activities.

Typical Inputs

5.2 Support Strategy
5.2 Integration Strategy
5.2 VV&T Strategy
5.2 Designs
5.2 Deployment Design/Plan
5.2 Configuration Management Plan
5.2 Quality Plan
5.2 Requirements Traceability Matrix
5.2 System Test Plan
5.3 Design Review Report
5.3 Security Policy Documents
5.3 Cost Estimate
5.3 Project Requirement Definition
5.3 Production Plan
5.3 Information Request Note
5.3 Scope Statement
5.3 Tested System
5.3 Error List
5.3 Build List
5.3 Test Reports
5.3 Metrics
5.3 Updated ECMS
5.3 System Documentation
5.3 Integration Plan
5.3 User Documentation



Typical Outputs

Updated ECMS5.5
Scope Statement5.5
Error List.....5.5
Integrated System5.5
Test Report5.5
Metrics Report.....5.5
Build List5.5
System Documentation.....5.5
User Documentation.....5.5
Security Policy Document5.5
Cost Estimate.....5.5
Production Plan.....5.5
Project Requirement Definition5.5
Information Request Note5.5
Design Review Report.....5.5
Installation Guide5.8

Typical Checks

Documentation is available;
Error List;
Production Plan;
Compliance With Security Requirements;
Test Reports;
Build List;
Adequacy Of Test Cover;
Under Configuration Management;
Actions From Previous Quality Gates;
Cost Estimate;
Scope Statement;
Project Requirement Definition;
Information Request Note.

Note: Consideration needs to be given to the identification of Minimum Input/Output Gate Criteria.

The Internet Version has links to appropriate documents and company wide processes (i.e. Systems Security).

An examples of the type of document provide by these links is the Feasibility Checklist:

A Feasibility Report

A feasibility report identifies and evaluates which of the alternative options can best meet the clients requirements. Options identified during the Feasibility Phase are included in addition to those specified during the Inception Phase.

Alternative options may have to be supported by the study and analysis of specific technical, operational and other aspects which affect their viability. This may require advice and assistance from specialists and input from users before feasibility can be established.

A feasibility report details:

- Which options have been investigated.
- Which options could deliver the client's requirements,
- The timescales and costs which would be incurred by the adoption of each feasible option,
- The risks, assumptions, dependencies and impact to the business associated with each feasible option.
- The reasons why options were rejected,
- A recommendation,

The recommendation should be supported by:

- Financial considerations and justification and timescale for implementation
- Technical and operational viability and practicality and match to requirements.

This includes consideration of aspects such as:

- The use of the proposed technologies and their compatibility with corporate medium and long term strategic aims and policy direction.
- Compatibility and configuration control requirements in relation to dependent and driver projects.
- This may involve harmonisation and rationalisation of objectives and scope with those of other projects, which in turn may affect planned benefits to be delivered by the project.
- Future flexibility, ease of upgrading and expansion,
- Ease and speed of migration from existing to new facilities, processes and procedures,
- Competitive advantage,
- Life expectancy of the asset,
- Ease of installation.
- Priority of deliverables if phased implementation is to be involved.
- Compliance with statutory, legal and regulatory requirements and engineering standards,
- Consideration of security and audit requirements,
- Environmental considerations,
- Any local site considerations.
- Quality aspects including reliability and spares provisioning, stand-by/ back-up arrangements, operational life expectancy and ease of replacement, and functionality and performance issues.
- Operational requirements including manning levels.
- Resource requirements and availability
- A broad assessment of risks, assumptions and dependencies which could lead to cost or duration overruns together with assessment of the risk management strategy that could be employed to reduce or eliminate them.
- A glossary of terms used should be included to ensure effective communication.

The feasibility report provides input to:

- Implementation timescale and cost estimates.
 - Operational capital and revenue expenditure forecasts.
- At the end of the Feasibility Phase, other documents produced in support of the feasibility report are:
- An outline PRD,
 - An outline business case based on the option agreed by the client.
 - A Definition Phase plan detailing the resources, timescales and cost estimates required.
 - An outline plan for the whole project.

One of the areas where it became clear that good guidance was needed was in the creation of VV&T Strategies.

Generic Strategies

The generic strategies in use have been developed from those that have been proven to be successful. However they have all been the subject of considerable thought in their production. Early on it was recognised that there can be a tendency to take a generic document and change a few words without the right level of thought into its application. Also for the documents to become a larger general test plan and

not a description of the strategy proposed to undertake a complex Systems Integration or VV&T exercise.

It came as a surprise to find that all the VV&T Strategies we examined actually contained no real strategic discussion! There were several reasons for this:

- They were not produced early enough in the project, so key decisions had been taken by default;
- Projects are frequently development-led and so ignore late-lifecycle issues;
- Projects avoided discussing quality targets because of the ‘emperors new clothes’ syndrome;
- They were produced under pressure to get a ‘tick in a box’

A true strategy for large projects should address areas that relate to the political and financial aspects of the project, as well as technical issues. It should clearly identify how the work of the VV&T team is going to ensure that the benefits outlined in the Business Case that gave authority to the project are going to be validated and tested, and required quality delivered.

A dilemma here is that providing a good example to copy can reduce thinking about the project goals, whereas providing a template can produce a strategy with minimal content. We decided that it was better to provide a good example to copy.

The Generic Strategy published as part of our VV&T Handbook and linked to the Integration Handbook has the following contents. (An example is available by contacting the authors.)

VV&T Strategy

Is a strategy produced for the project or task as a whole, and it should cover the following?

the aims of the VV&T exercise
the political and budget constraints
the aims of the business case
the quality goals
how the aims of the business case will be addressed by the verification and validation process
the relationships between VV&T, requirements, design, development, project management, quality assurance and configuration management
the VV&T teams and defining roles, responsibilities, and authorities
measurable objectives for the quality and testing required
mapping of the VV&T activities onto the chosen lifecycle - this may also include systems integration stages
what will and won't be tested
schedule of the VV&T activities with respect to the requirements, design, development and release schedule, and agreed with the project manager
the approach to be used, that is, strategies, methods, techniques, tools, the pass/fail criteria, configuration management, problem reporting, co-ordination of test data and so on,
the test environment to be used
resource and training needs of VV&T team
the VV&T documentation structure, referring to the definitions of each document type
testing start, exit and handover acceptance criteria - this will mean referencing specific handover documents
interruption, suspension and resumption criteria
any standards and conventions to be used
the assumptions, risks and contingencies used in formulating the VV&T Strategy (for instance, assumptions - machine access, availability of tools, revisions of compiler; risks - use of untrained staff, complex modules, late delivery of units)
how the results of the VV&T work will be reviewed

A Strategy may contain information more properly located in the Test Plan and what happens frequently is that it effectively becomes a Test Plan because it is hard to write the strategic sections of the document. A good test plan will have the following main points:

Test Plan

A test plan prescribes the scope, approach, resources and schedule for the testing activities, and it is recommended that one should be produced for each test phase (for instance, acceptance, system). It identifies the items being tested, the features to be tested, the tasks to be performed, the personnel responsible for each task and the risks associated with the plan.

introduction - summarise the items and features to be tested. The need for each item and its history may be included.

test items - identify the test items including their version/revision level. (If version/revision numbers are likely to change during the life of the test plan, it may be better to specify these in a separate document which can be kept up-to-date to give a snapshot view of the status of the system at any time). Specify any means by which the items may be transferred (for example, programs may be transferred from tape to disc). This could be specified in the Configuration Management Plan, and referenced here. Supply references to the following item documentation, if it exists:

- requirements specification
- design specification
- user's guide
- operations guide
- installation guide.

Reference any incident reports relating to the test items. Items which are to be specifically excluded may be identified.

features to be tested - identify all features and combinations of features to be tested. Identify the test design specification associated with each feature or combination of features.

features not to be tested - identify all features and combinations of features not to be tested and the reasons.

approach - describe the overall approach to testing. Identify the approach to each test feature, or major group of features. Specify activities, techniques and tools to be used. The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to complete each one. Specify completeness criteria and degree of comprehensiveness. Identify any constraints on testing.

entry/exit criteria - specify the criteria used for the commencement and completion of the testing activity in the test phase associated with this plan - see handover criteria/quality gates.

suspension criteria and resumption requirements - specify the criteria used to suspend all or a portion of the testing activity on the test items associated with this plan. Specify the testing activities which must be repeated when testing is resumed.

test deliverables - identify the test deliverables, which may include:

- test design specifications
- test case specifications - these may be test tool scripts
- test procedure specifications
- release notes
- test logs
- test incident reports
- test summary reports
- test input/data should be identified as deliverables
- test tools (for example, drivers, stubs) should be included
- handover criteria document.

testing tasks - identify the set of tasks necessary to prepare for and perform testing, all inter-task dependencies and any special skills required. (It is highly likely that this information would be put into a project management software package, and the resulting schedule referenced here).

environmental needs - specify both the necessary and desired properties of the test environment. This specification should contain the physical characteristics of facilities including the hardware, the communications and system software, the mode of usage and any other software or supplies needed to support the test. Specify the level of security which must be provided for the test facilities, system software and proprietary components. Identify special test tools needed. Identify any other testing needs (for example, publications, office space). Identify the source for all needs which are not currently available to the testing team. Any differences between the operational environment and the test environment, and the effect this may have on the scope of testing, could be identified here.

responsibilities - identify the groups responsible for managing, designing, preparing, executing, witnessing checking and resolving. In addition, identify the groups responsible for providing the test items and environmental needs.

schedule - include test milestones identified by the project schedule as well all item delivery events. Define any additional milestones, and the time to complete each testing task and milestone. For each testing resource (for instance, tools, staff) specify its period of use. This information could be entered into a project management software package, and the resulting plan could be referenced here. risks and contingencies - identify the high risk assumptions of the test plan, and specify contingency plans for each.

approvals - specify the names and titles of all persons who must approve the plan.

The Way Forward

As part of the continuing improvement actions for N&S Systems Engineering, a VV&T Improvement team is working to improve the Strategies in use on projects. Action is in place to ensure that all VV&T strategies are properly reviewed and compared against the Generic and the identified best of type. The key areas being addressed are:

Make it simple to use

Educate about value of this approach

- Projects often developer-led – change to integration-led

- Focus on whole-lifecycle costs

Project Manager training and accreditation

Management and technical Review

- Make sure that Business Goals are identified and testable

- Identifies constraints due to organisation, market place, politics

- Identifies risks

- Use qualified reviewers – systems engineering skills are critical

- Puts requirements on other lifecycle activities eg implementation technology

- Constrains system architecture to ensure cheap, efficient testing can be done

Team input – whole lifecycle must buy into the approach

Learning and review

- Use of web to hold latest techniques, feedback, tool tips

- Research better techniques for decision making

Millenium

Finally, it is worth just looking at how WWW is used in BT's Millenium programme. All the millennium testing in BT is being carried out against a set of defined strategies and test cases. These cover both Operation Support and Network Systems. All this documentation, together with databases about the Millenium test status of all BT applications, and guidance on Millenium approaches, are available to all people within BT. This has made a significant difference to the clarity and general awareness of the Millenium issues.

Below is an example of the kind of documentation we have made available.

Conclusion

Web based process guidance with links to exemplar documents and detailed techniques has proved effective in spreading good practice over a very wide audience.

Writing a good VV&T Strategy is a demanding activity requiring excellent systems engineering skills. These skills are not as widespread as is desirable and the use of Web has helped capture best practice in a very effective, usable manner.

Generic Year 2000 Testing Checklist

This checklist defines the minimal set of tests which will be required to demonstrate that a system or component is Year 2000 compliant. It can be used to record the results of a test or to confirm that evidence is available (e.g. from test results) that the system or component is compliant.

The checklist may be used for both internally developed systems and externally supplied products and can be used as a supplement to existing acceptance/quality gate criteria.

The received system or component has been subjected to the following tests, the results of which are recorded below:

<i>Test Case</i>	<i>Test Case Reference:</i>	<i>Pass/Fail</i>
<i>Roll Over:</i>		
<ul style="list-style-type: none"> Test that the system handles the roll over from 31st December 1999 to 1st January 2000 (and similarly for 2000 to 2001) 		
<i>Day of the Week:</i>		
<ul style="list-style-type: none"> Test that 1st January 2000 has the correct day of the week, Saturday. (It is possible that a Monday is assigned incorrectly which is the day for 1st January 1900) 		
<i>Leap Year:</i>		
<ul style="list-style-type: none"> Test that the 29th February is recognised and processed as a valid date. Also test around these dates to detect secondary defects. Test specifically across the dates 27th, 28th and 29th February and 1st March 2000 (29th February is a valid date) Test the same dates for 1999 (29th February should fail) Test the same dates for 2001 (29th February should fail) Test the same dates for 2004 (29th February is a valid date). 		
<i>Financial Year:</i>		
<ul style="list-style-type: none"> Test processing of data forwards and backwards across the century date change for the financial year (e.g. dates up to and including 31st March 2000) 		
<i>Processing time horizons</i>		
<ul style="list-style-type: none"> Test the application's processing time horizon spanning the century date change, e.g. over a period extending into the next century, and from a future date in the next century covering a period extending back into this century. The processing of information under these situations must ensure that any data manipulations, calculations, sorts, etc. work correctly over the century cross over. 		
<i>Implicit Century</i>		
<ul style="list-style-type: none"> The correct century must be interpreted unambiguously and be inferred with 100% accuracy based on the value for date. Where inferencing logic is used, e.g. with 2 digit representing the year, the rules and pivot dates must be clearly defined. Test around the pivot dates to ensure that the dates are interpreted correctly. Rules for century inferencing as a whole must apply to all contexts in which the date is used, although different inferencing rules may apply to different date sets. 		
<i>High Risk/Special Meaning Dates</i>		
<ul style="list-style-type: none"> Test that the system does not use special date values as logical flags, such as "99" for the year to mean "no end date" or "00" to mean "does not apply." 31/12/1999 - should be able to distinguish between a regular end-of-year 1999 date and a special meaning date. For example, a never-expiring date indicator which is sometimes used. 9/9/1999 - should be processed correctly and is not used as a special meaning date. Test other high risk dates, e.g. 01/01/2000, 01/01/2001. 		
<i>Use of Ordinal Dates</i>		
<ul style="list-style-type: none"> For applications using ordinal dates, test that conversion to Gregorian dates are handled properly. For example, 2000/60 should convert to 29th February, 2000/61 to 1st March, etc. Check that 31st December 2000 is represented as 2000/366. 		
<i>Archives, backups and restores</i>		
<ul style="list-style-type: none"> Test correct functioning of house keeping routines, e.g. a system reload in 2000 from an archive taken in 1999 (also 2001 from a 2000 archive). 		

System Name:

Certified compliant by:

Name:

Date:

SPONSOR

Software Research, Inc.

Software Research, Inc. created the field of Automated Software Testing, and has been providing the QA community with testing tools since 1987, with CAPBAK capture/playback system, SMARTS, the corresponding test manager and the first commercial test coverage analyzer, TCAT.

Since 1995 TestWorks has been the original and only Suite of Integrated Testing Tools that includes both test regression and test coverage support for Embedded, GUI, Client/Server and Web Applications, on UNIX and Windows platforms. TestWorks has been successfully applied in enterprise wide Y2K testing applications.

More recently TestWorks has added TCAT for Java, the first coverage analyzer for Java applets and the Remote Testing Technology (RTT) with local, Email, and over-the-Web collection of snapshot user interaction and refined test coverage data from Java applets.

Software Research, Inc., the company that pioneered end-to-end testing solutions, looks forward to helping improve the Quality of your applications and achieving 100% return on your IT tools and technology investment.

TestWorks. It's what you need most.

To learn more about TestWorks from Software Research, Inc., explore the web at: <http://www.soft.com>, or send email at info@soft.com

CO-SPONSORS

Gold Sponsors

CMG Information Technology

CMG plc is a leading European IT services company, providing business information solutions through consultancy, systems and services to clients worldwide. Established in 1964, CMG now operates in more than 40 countries from its bases in the UK, The Netherlands, Germany, France and Belgium. The company is listed on the London and Amsterdam stock exchanges. CMG supplies services and products in the finance, trade and industry, transport, telecommunications, energy and public sectors. The Group also provides managed information processing services, including networks, payroll and personnel.

One of CMG's many specialisations is the automation of structured testing. Testing of new software products is a time consuming business. CMG has developed a very successful

testing method TestFrame. TestFrame is based on CMG:CAST, an approach for the structured and automated testing of new or modified software. TestFrame reduces testing times considerably, and thus speeds up the time-to-market for new products and services. With this method, CMG is currently one of the leading companies in this field. CMG is dedicated to helping its clients and their people become more successful through the quality of its services and staff. Strong employee commitment ensures the Group's long term success and hence the success of its clients. Visit CMG's website at: <http://www.cmg.nl>.

SIM Group Ltd.

SIM Group Ltd (Systems Integration Management Limited) has developed its own approach to implementing test automation technology known as Automated Testing Support (ATS). This is a proven combination of techniques, tools and training which is effective of many different types of systems and platforms. SIM's independent position on all issues related to testing is invaluable to its customers, providing an objective view of the test processes taking place and recommending the tools, utilities and technology best suited to each environment. Our working relationship with major tool vendors and experience in the implementation and use of such tools place us in an unparalleled position for independent advice, As a result of our many years of practical experience in testing, SIM offer a specialised range of services which are encompassed in the following ways:

- Software Testing**
- Testing Environments**
- Testing Methods**
- Testing Consultancy**

Visit SIM Group's website at: <http://www.simgroup.co.uk>.

Silver Sponsors

IQUIP Informatica B.V.

Since 1972 IQUIP Informatica B.V. has been supporting organisations in carrying out their core processes. With its 1200 employees IQUIP does this by constructing, maintaining testing and implementing application software systems. IQUIP increasingly concentrates on carrying out assignments with result responsibility. In testing, IQUIP achieves this through her dedicated division Components & Testing (300 employees), using the structured testing approach TMap. TMap was developed by IQUIP itself and has become a widely used international standard. TMap related methods such as TAKT (test automation), TSite (test laboratory) and TPI (Test Process Improvement) have recently completed the product range. A dedicated R&D team is continuously improving these methods and, if required, develops new products. Visit IQUIP's website at: <http://www.iquip.nl>.

Mercury Interactive

Mercury Interactive is the world's leader in enterprise application testing solutions. The company offers a comprehensive line of automated testing tools that address the full range of quality needs for testing client/server, e-business, Y2K, Euro, and packaged applications. Its testing solutions enable corporations, systems integrators and independent software vendors to identify software errors more quickly and efficiently than traditional methods allow.

Mercury Interactive offers a complete family of tools to test the enterprise. WinRunner 5.0 for Windows and XRunner 5.0 for UNIX deliver fast, accurate, repeatable and automated tests that are unrivaled in the automated client application testing tools market. They simplify test automation, providing the most powerful, productive and cost-effective test solutions. LoadRunner 5.0 is Mercury Interactive's integrated client, server and Web load testing tool. It provides the only scalable load testing solution for managing the risks of client/server systems. By using a powerful set of automated management functions and an open database repository to store and access test information, TestDirector 5.0 offers the most productive workgroup test management software. Astra SiteManager is a comprehensive visual Web site management tool, Astra Site Test is a load testing tool for Web-based systems and Astra QuickTest is an icon-based functional testing tool for e-business. Visit Mercury Interactive's website at: <http://www.merc-int.com>.

The Testing Consultancy

The Testing Consultancy is a specialist consultancy that was formed to provide independent testing services within the IT Industry.

With our highly qualified and experienced Consultants and Associates, we are strongly positioned to provide guidance, expertise and support for all your testing requirements. Our independent and objective advice will help your business to increase the quality and reliability of its systems, whilst actually reducing costs, and delivering those systems within the required timescales.

Golden Leaf Sponsor

GiTek Software n.v.

GiTek Software nv supplies a number of services offering a global solution to testing:

- Consultancy in testware**
- Structuring of the test process**
- Education and training**
- Test planning and test management**
- Test project and test execution**

PARTNERSHIPS

About ACM

ACM, the Association for Computing Machinery, is an international scientific and educational organization dedicated to advancing the arts, sciences and application of information technology. With a worldwide membership of 80,000, ACM functions as a locus for various fields of Information Technology. Membership benefits include a subscription to the Communications of the ACM, discounts on conferences and publications, 36 special interest groups and the new ACM Digital Library. The Digital Library includes unlimited access to 22 ACM publications and archives, 6 years of conference proceedings and over 100,000 pages of text, with full searching capabilities.

For more information, visit our website at: <http://www.acm.org> or contact ACM directly at: 1 (800) 342-6626 (USA & Canada) or 1 (212) 626-0500 (anywhere), by fax at: 1(212) 944-1318, email: acmhelp@acm.org, or by writing to ACM, Member Services Department, P.O. Box 11315, New York, NY 10286-1315.

European Software Institute (ESI)

The European Software Institute is one of the world's leading independent authorities on software process improvement. ESI is a non-profit making organisation driven by the demands of European industry. It is supported by the European Commission, the Basque Government and through company membership. ESI's work is centred on products and services that are tied directly to core business objectives such as reducing costs and increasing predictability of results among others. ESI's headquarters are in Bilbao, Spain.

European System and Software Institute (ESSI)

Enterprises in all developed sectors of the economy - and not just the IT sector - are increasingly dependent on quality software-based IT systems. Such systems support management, production, and service functions in diverse organisations. Furthermore, the products and services now offered by the non-IT sectors, e.g., the automotive industry or the consumer electronics industry, increasingly contain a component of sophisticated software. For example, televisions require in excess of half a Mbyte of software code to provide the wide variety of functions we have come to expect from a domestic appliance. Similarly, the planning and execution of a cutting pattern in the garment industry is accomplished under software control, as are many safety-critical functions in the control of, e.g., aeroplanes, elevators, trains, and electricity generating plants. Today, approximately 70% of all software developed in Europe is developed in the non-IT sectors of the economy. This makes software a technological topic of considerable significance. As the information age develops, software will become even more pervasive and transparent. Consequently, the ability to produce software efficiently, effectively, and with consistently high quality will become increasingly important for industries across Europe if they are to maintain and enhance their competitiveness.

The goal of the European Systems and Software Initiative (ESSI) is to promote improvements in the software development process in industry, through the take-up of well-founded and established - but insufficiently deployed - methods and technologies, so as to achieve greater efficiency, higher quality, and greater economy. In short, the adoption of Software Best Practice.

De Koninklijke Vlaamse Ingenieursvereniging (KVIV)

De Koninklijke Vlaamse Ingenieursvereniging (KVIV) was founded in 1928 and has 12,000 members: civil engineers, agricultural engineers, chemical engineers, bio-engineers, and polytechnical engineers from the Royal Military School. It also has 1800 members in its subdivision: Software Metrics.

KVIV is an open and flexible organization that gives ongoing training programs and publications for entrepreneurs, managers, docents, and government employees. Thanks to its internal structure, KVIV contacts diverse public groups as well as academic and industry circles.

Studiecentrum voor Automatische Informatieverwerking (SAI)

SAI (Studiecentrum voor Automatische Informatieverwerking) is a non-profit organization whose purpose is to promote the knowledge of Information. As far as the theoretical as practical aspects of information knowledge, the organization takes a stand on social questions that has to do with automation of information as far as it applies. The activities of the organization are focused on people who are specialists in information. SAI organizes discussions, meetings, seminars, workshops and a magazine called "Informatie".

Vendors

Blackstone & Cullen, Inc.

Blackstone & Cullen, Inc. is an information technology consulting firm. Its Data Commander product is a year 2000 testing and data management tool that also offers currency conversions.

CYRANO (UK) Ltd.

CYRANO is a worldwide provider of Testing Solutions for Client/Server, Internet and legacy application. CYRANO Millennium Test Suite: comprehensive Y2K compliance testing

McCabe & Associates

McCabe and Associates will be showing a suite of software tools for Euro Conversion, Year 2000 compliance, Quality Assurance, Test Coverage, reverse engineering and software maintenance.

McGraw-Hill Publishing Company

The McGraw-Hill Publishing Company is one of the world leading publishers in Computer Science. We publish for academics and professionals on subjects like software engineering, programming and certification.

OM Partners

OM Partners n.v. is a company specialized in consulting and software development in the area of decision support systems, and of short, medium and long term production planning and logistics. We apply computer aided software testing to ensure high quality software to our customers.

VAC Software Engineering

VAC Software Engineering provides professional software development solutions to design, build, test and maintain information technology systems including tools, consultancy, training, support and implementation services.

John Wiley & Sons

Visit the Wiley Display to view information on the latest software books and journals available. Alternatively, visit the Wiley website to keep updated - <http://www.wiley.co.uk>

Y2K Stress?



Relax.

CYRANO MillenniumTest™ is a comprehensive, easy to use solution that includes automated testing software with a unique client/server architecture, a documented Y2K methodology and training – **all specifically designed for character-based applications.** CYRANO MillenniumTest will help you validate an application's date sensitive behavior and its core functionality, as well as generate documented results that identify what was tested, when it was tested and how.

The return on investment is immediate – empower your team with **CYRANO MillenniumTest** and speed up your Y2K projects.



Relax. With **CYRANO Millennium Test.**

CYRANO specialises in the **provision of software testing solutions supplying a range of services to cater for any and/or all of your software testing needs.** This is particularly useful to many organisations that, as a result of the **Millennium Bug**, have been forced to undertake the testing of their systems, often for the first time. In such circumstances, the benefits of acquiring an automated tool are often lost due to a lack of resource and/or expertise in testing.

Y2K Compliance testing is such that traditional manual testing methods are insufficient and unlikely to identify many of the faults that will lead to system failures. Manual testing would require more time and human resource than is available, with the possibility of even the most critical of applications going untested.

CYRANO's philosophy of providing complete working solutions enables such organisations to solve their immediate testing problems.

Expenditure on Year 2000 Compliance test tools is considered the only aspect of a year 2000 project spend that has a value after the year 2000, and one that can lead to an improvement in performance. This makes the expenditure an investment rather than a cost.

MillenniumTest is the only GCAT listed testing tool for Y2K.

The Euro is coming.

Year 2000 is coming.



You've finished testing
your converted applications...haven't you?

Two major conversion projects are coming your way. We're talking about deadlines that are etched in stone. And your IT staff is already stretched to the limit by their existing workload. The only answer is to use Mercury Interactive's Automated Testing Solutions. They'll automatically test and verify all your functionality and key business processes under real working conditions, while managing the entire project. And you can use the same tools for all your testing needs across the enterprise. So, when the inevitable happens, you'll be ready. **MERCURY INTERACTIVE**

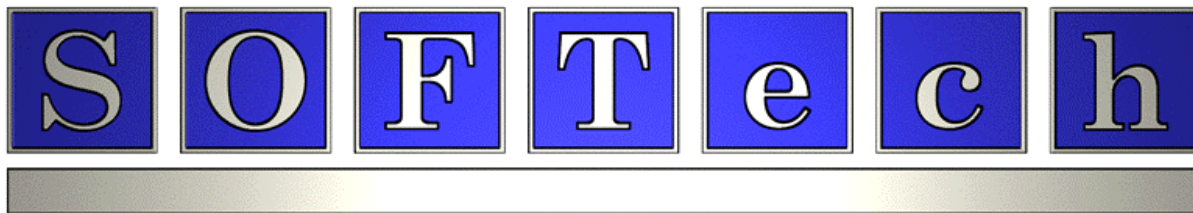




**White Rose Court
Oriental Road
Woking, Surrey
England GU22 7PJ**

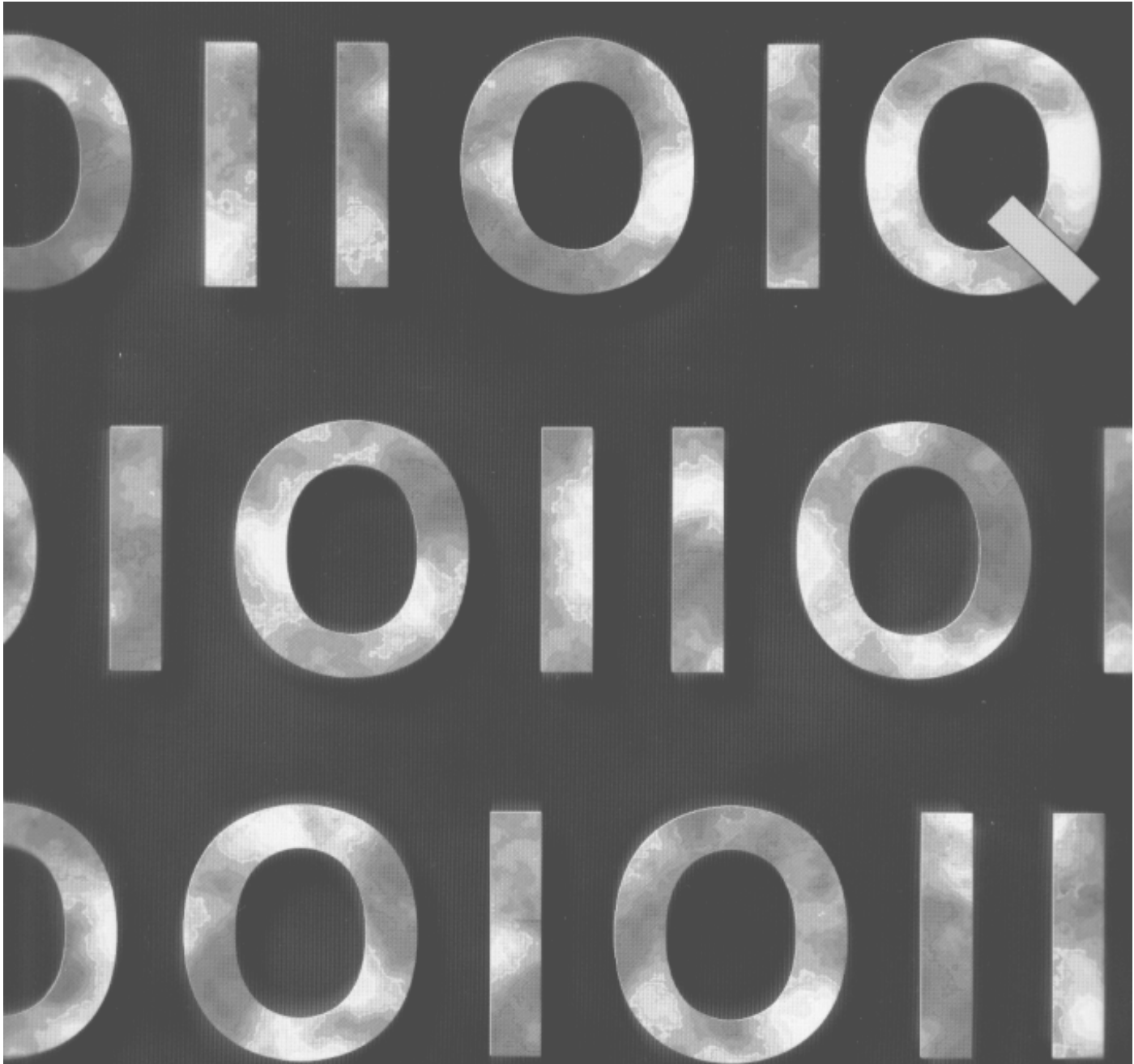
**Phone + 44(0) 1483 740289
FAX +44(0) 1483 720112
email: info@simgroup.co.uk
URL www.simgroup.co.uk**

SIM Group Ltd. are totally dedicated to software testing. SIM will supply and manage testing teams that perform testing work to the highest standards of efficiency and effectiveness. SIM performs consultancy, execution and training of testing. SIM's methods and techniques for testing include testing frame works, libraries, documented procedures and deliverable templates that can all be used to speed up the adoption and use of good testing practices. SIM's approach to testing relies heavily on the adoption of automated testing techniques as much as is practical and SIM has a proven track record of success with automated testing. SIM's subsidiary SOFTech Tools Ltd. provides selected and specialised testing tools that contribute to a fully automated testing process.



- ⇒ **Automated Testing Facility**
- ⇒ **eSuite**
- ⇒ **SunTest Suite**
- ⇒ **Quest**
- ⇒ **Test Master**

- ⇒ **Testing Projects**
- ⇒ **Testing staff**
- ⇒ **Testing Consultancy**
- ⇒ **Testing Environments**
- ⇒ **Testing Automation**



Organizations are stronger with CMG.

THE NETHERLANDS	UNITED KINGDOM	GERMANY	FRANCE	BELGIUM	NASHUA, USA	SINGAPORE
-----------------	----------------	---------	--------	---------	-------------	-----------

TOTAL QUALITY MANAGEMENT		
Test Tools	TestWorks	Your Process
Capture/Playback	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Test Management	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Test Data Generation	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Static Analysis	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Metrics	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Path & Branch Coverage	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Call-Pair Coverage	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Web Applications	<input checked="" type="checkbox"/>	<input type="checkbox"/>

All the tools you need for rock-solid code

Software Research has created a suite of fully integrated and automated testing tools to help you ensure the quality of your software products. TestWorks offers an end-to-end solution that covers all aspects of your process life cycle: test design and development, test data generation, test execution and evaluation, reporting and test management, code comprehension, coverage analysis, metrics and maintenance. Perfect Tools for the Quality Architect.

TestWorks improves the quality of your products, whether in an embedded or distributed client/server system, in GUI desktop or web applications. Its open architecture empowers your work across the major UNIX and all windows platforms and OSs, in C, C++, Ada, Fortran and Java.

You need to deliver high-quality software on time, under budget?

We have TestWorks! Contact the company that created the field.

SR Software Research

1 (800) 942-SOFT
email:info@soft.com

<http://www.soft.com>



“We will see...”

When you implement a software system you expect it to function properly. But you can only be sure when you test your system thoroughly before use. Not only by using the classical test tools but also by involving well-trained critical professionals who test the system with you, in your organization. For more than 10 years, Gitek Software has been a major player in the arena of software development for large accounts. Our broad experience has taught us that software testing is not only useful but also absolutely necessary if you want to avoid large costs at the end of the development cycle. Gitek uses the Test Management Approach (TMap®) for software testing. TMap® is based on four cornerstones: the test life cycle (what and when?), techniques (how?), infrastructure (where and using what?) and organization (who?). TMap® can be used during development and during maintenance. Testing with Gitek offers you the same critical approach we apply for software development. Do you want quality in your IT? Test IT! Call +32 (0)3 231 12 90, fax +32 (0)3 226 10 83 or E-mail: gitek@gitek.be for more information.

GiTek Software n.v.

We'd better test

St. Pietersvliet 3 • B-2000 Antwerp

Are you prepared for Year 2000 and the euro?

Are you sure?

**Let Data Commander™ help ensure that your
answer to these questions is YES.**

**Data Commander:
Quality Year 2000 Testing and Data Management Tool
Featuring eurodollar conversions,
smart output testing and data migration**

**Data
Commander™**

Your Y2K & Euro Solution Release 2.1

Developed by:



BAC Blackstone & Cullen, Inc.

Call Blackstone & Cullen, Inc. today:

**2000 RiverEdge Parkway, Suite 750
Atlanta, Georgia 30328 USA
+ 1 770-612-1550/ FAX: +1 770-612-1471
<http://www.bac-atl.com>
<http://www.datacommander.com>**

COMPANY

OM Partners is a solution provider with 15 years of experience in developing and implementing Supply Chain Planning Systems. Beyond the delivery of software, we provide professional assistance from analysis, over implementation to support.

SECTORS

- **Flow Shop** like: Corrugated Board, Metals, Paper, Plastics, Solid Board, Textiles, ...
- **Semi Process** like: Animal feed, Chemicals, Dairy, Fertilizers, Food & Beverages, Pharmaceuticals, Starch, ...
- **Hortica** for horticultural production and optimization of crossings.

PRODUCTS

The OM Partners product range for Supply Chain planning integrates all levels of the planning hierarchy with both interactive and intelligent solutions. These are standard software packages, which can be parametrised to your needs.

OMP Optimization

OMP Optimization (OMP) is our solution for the highest level of Supply Chain planning. It deals with strategic issues like optimal product mix of co- and by-products, sourcing and make-or-buy decisions, warehouse localization and (re)allocation problems.

OMP Master Production Scheduler

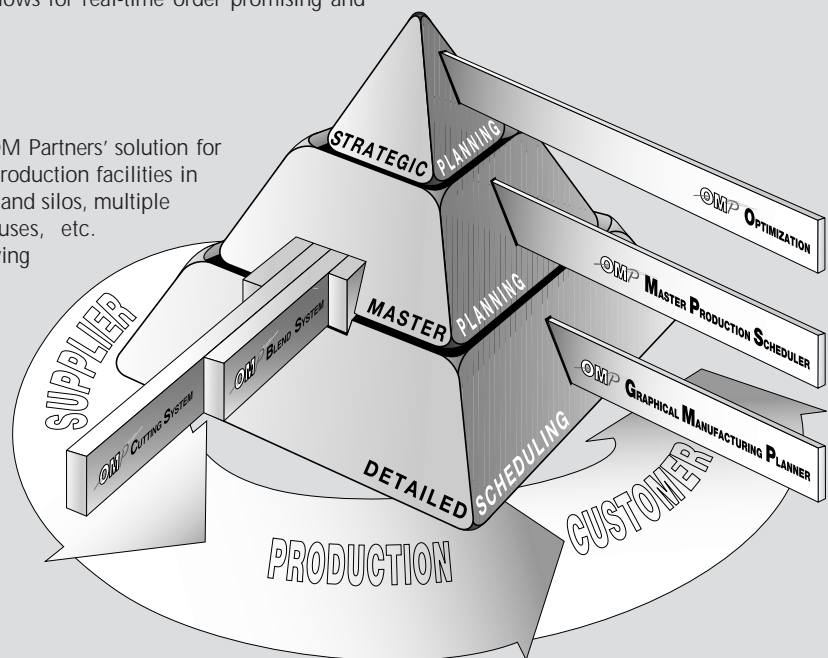
OMP Master Production Scheduler (MPS) is an interactive graphical package for Master Planning. Based on sales forecasts and orders from different customer groups in various countries, MPS can take into account machine-, resource- and storage capacities in different plants, material requirements, transportation (truck, train, barge, etc.) and customer specific constraints (batch sizes, buffering policies, etc.). MPS will automatically generate a multi-level finite capacity plan with minimal transportation, manufacturing and storage costs, minimal deviation from inventory and production targets, etc.. MPS also allows for real-time order promising and service level optimization.

OMP Graphical Manufacturing Planner

OMP Graphical Manufacturing Planner (GMP) is OM Partners' solution for detailed scheduling problems. GMP models your production facilities in great detail: set-ups, precedence relationships, tanks and silos, multiple BOMs and resources, batching rules, warehouses, etc. GMP allows to simulate in a mouse-driven windowing environment: drag & drop facilities, order split & merge, graphs and reports built through interactive contents selection, etc. You can monitor inventory and WIP evolution, see the impact of production events and order changes on set-up costs, resource usage, due date performance.

Sophisticated solvers

Intelligent modules exist for optimization problems like cutting, blending, sequencing, cycle planning, ...





The Testing Consultancy

SPECIALISTS IN TESTING AND TEST AUTOMATION

The Testing Consultancy, *Dynamic Business Practitioners*, providing independent testing services to the IT industry. Tool independent, our recommendations are based solely on your requirements, helping you to increase the quality of your systems, while actually reducing costs and delivering on time

Our mission : “to provide a practical testing solutions”

Our services include

- Testing Consultancy
Test Strategy Definition, Test Planning & Test Management
- Test Process Improvement
Testing Review, Test Measurement & Metrics, QA Procedures
- Test Outsourcing
On site or Off site
- Test Automation
Tool Evaluation, Selection & Implementation
- Testing Resources
Test Managers, Test Designers, Test Analysts & Technical Specialists
- Test Training
Bespoke training courses for all phases of testing

Contact Details

- Lynn Thopmson c/o The Testing Consultancy Ltd
- 3 Linford Forum, Rockingham Drive Linford Wood, Milton Keynes. MK14 6LY
- Tel : +44 1908 395050 Fax : +44 1908 395051 info@testing-

The Testing Consultancy
SPECIALISTS IN TESTING AND TEST AUTOMATION





Software Quality Management & Automated Testing

The demand for reliable high quality software is increasing continuously to support the business needs. One of the aspects to improve software quality is testing. A clear and concise vision towards software testing is one of the success factors of this aspect as well as an adapted infrastructure of testware.

As a distributor of software tools in the Benelux, VAC focuses on:

- support of the development and test process by a suitable choice of available tools, and
- adaption of the user interface of test tools to the end user.

We believe a table driven test suite combined with a configuration management tool, a problem tracking system, and a suite of testware to support automated software testing leverages to a higher level of development process maturity.

VAC Software

Engineering provides professional software development solutions to design, build, test and maintain information technology systems including tools, consultancy, training, support and implementation services.

Come and see us

To experience our table driven test suite we invite you to Dr. Boudewijn Schokker's presentation at the Vendor Technical Track on **Thursday 12. November at 11.00 am**

VAC

Hullenbergweg 371
Postbus 12470
1100 AL Amsterdam
The Netherlands
Phone: + 31 (0)20 6520200
Fax : + 31 (0)20 6918522
E-mail: solutions@vac.nl
Internet:
<http://www.vac.nl>

VAC is authorised distributor in the Netherlands ie the Benelux of BitbyBit, Help Desk Systems, Information Advantage, Micro Focus, MKS, Netron, Progress Software and Softbridge. For IT training has VAC an exclusive partnership with Gartner Learning. VAC is a subsidiary of Management Share N.V., listed at the Amsterdam Stock Exchange.



International Software Quality Week Conferences

Request for Conference Proceedings Request for Tutorial Notes

Yes, I am interested in copies of the Proceedings of the International Quality Week Conferences (QW and QWE). This form indicates the publications I am ordering.

Note: Email all questions or comments related to Quality Week to qw@soft.com.

Conference Proceedings

Specify Number	Description	Price
	CD-ROM Quality Week Europe 1998 (QWE'98)	\$50
	Quality Week Europe Proceedings 1998 (QWE'98)	\$50
	Quality Week Proceedings 1998 (QW'98)	\$50
	Quality Week Europe Proceedings 1997 (QWE'97)	\$25
	Quality Week Proceedings 1997 (QW'97)	\$25
	Quality Week Proceedings 1996 (QW'96)	Sold Out
	Quality Week Proceedings 1995 (QW'95)	\$25
	Quality Week Proceedings 1994 (QW'94)	Sold Out
	Quality Week Proceedings 1993 (QW'93)	\$25

Tutorial Notes

Specify Number	Description	Price
	QWE'98 Tutorial Notes	\$25
	QW'98 Tutorial Notes	\$25
	QWE'97 Tutorial Notes	\$25
	QW'97 Tutorial Notes	\$25
	QW'96 Tutorial Notes	Sold Out
	QW'95 Tutorial Notes	\$25

Shipping and Handling Per Proceeding Set (USA Destination)**USA Destination:**

UPS Ground	\$10
US Postal Service Priority	\$25
US Postal Service Express	\$30

Canada & Mexico:

UPS Standard	\$30
UPS Expedited	\$65
UPS Express	\$70

Shipping and Handling Per Proceeding Set (International)**Europe:**

US Postal Service - Surface	\$35 (2-3 weeks)
US Postal Service - Air	\$50
UPS Eastern Europe	\$170 (4-7 days)
UPS Western Europe	\$110 (4-7 days)

Pacific Rim:

US Postal Service - Surface	\$35 (2-3 weeks)
US Postal Service - Air	\$70
UPS Expedited Air	\$90 (Tokyo, Hong Kong)
UPS Expedited Air	\$100 (Other Asia)

Quantity of Proceedings Ordered:

Total Proceedings Shipping Cost:

Shipping and Handling Per Tutorial Set (USA Destination)**USA Destination:**

UPS Ground	\$10
US Postal Service Priority	\$20
US Postal Service Express	\$35

Canada & Mexico:

UPS Standard	\$25
UPS Expedited	\$50
UPS Express	\$55

Shipping and Handling Per Tutorial Set (International)**Europe:**

US Postal Service - Surface	\$15 (2-3 weeks)
US Postal Service - Air	\$30
UPS Eastern Europe	\$110 (4-7 days)
UPS Western Europe	\$75 (4-7 days)

Pacific Rim:

US Postal Service - Surface	\$15 (2-3 weeks)
US Postal Service - Air	\$35
UPS Expedited Air	\$65 (Tokyo, Hong Kong)
UPS Expedited Air	\$70 (Other Asia)

Quantity of Tutorials Ordered:

Total Tutorials Shipping Cost:

TOTAL SHIPPING PRICE:

TOTAL PRICE:

I'm returning this form with:

Check
Credit Card
WireTransfer

Visa Number Exp. Date

MasterCard Number Exp. Date

AMEX Number Exp. Date

Advance Payment by international wire transfer to:

Name: Software Research Institute

Account Number: 0052-078029

Address: Wells Fargo Bank
 Brannan Office
 601 Third Street
 San Francisco CA 94107 USA

Please send the Quality Week Proceedings or Tutorials without delay to my attention:

First Name: Last Name:

Title:
Company:
Address1:
Address2:
City: State: Postal/ZIP Code:
Country:
Phone:
FAX:
E-Mail:

SUBMIT INFORMATION CLEAR FORM

Home	Index	News	Products	Technology	Solutions	App Notes	Screen Shots	Support
Quality Week	Institute	Company	Distributors	Partners	Careers	Users	Information	



Software Research, Inc.
625 Third Street
San Francisco, CA 94107-1997 USA

Phone: +1 (415) 957-1441
TollFree: +1 (800) 942-SOFT [USA Only]
FAX: +1 (415) 957-0730
Email: info@soft.com



2nd INTERNATIONAL
SOFTWARE QUALITY WEEK EUROPE
(QWE'98)

9-13 November 1998, Brussels, Belgium

SPONSORING ORGANIZATIONS

[QW Series | QWE'98 Home | Download (Call, Ad) | Download Brochure | Send Brochure]
[Kudos | Tour | PROGRAM | Abstracts | Bios | Y2K Clock]
[Advisory Board | Sponsors | Exhibits | REGISTER | Hotels | Brussels]

Quality Week/Europe '98 (QWE'98) is run by Software Research Institute (SR/Institute), a not-for-profit subsidiary of Software Research, Inc.. Organizations which have agreed to be sponsors of the QWE'98 event are given below.

The QWE'98 event is organized and presented in cooperation with certain other organizations, whose support is also acknowledged below. Note: Additional sponsors and cooperating organizations are expected to be added. The information given below was last updated on 26 August 1998.

IN COOPERATION WITH...

ABOUT THE ACM...



The ACM (Association for Computing Machinery) has approved Quality Week '98 as an event presented in cooperation with the ACM. ACM members receive a 10% discount when they register using their ACM Membership Number.

ACM, the Association for Computing Machinery, is an international scientific and educational organization dedicated to advancing the arts, sciences, and application of information technology. With a worldwide membership of 80,000, ACM functions as a locus for various fields of Information Technology.

Membership benefits include a subscription to Communications of the ACM, discounts on conferences and publications, 36 special interest groups and the new ACM Digital Library. The Digital Library includes unlimited access to 22 ACM publications and archives, 6 years of conference proceedings and over 100,000 pages of text, with full searching capabilities.

For more information visit our website at: <http://www.acm.org> or contact ACM directly at: 1 (800) 342-6626 (U.S.A. & Canada) or 1 (212) 626-0500 (anywhere), by fax at: 1 (212)

944-1318, email: acmhelp@acm.org, or by writing to ACM, Member Services Department, P.O. Box 11315, New York, NY 10286-1315.

ABOUT ESI...



The European Software Institute is one of the world's leading independent authorities on software process improvement. ESI is a non-profit-making organisation driven by the demands of European industry. It is supported by the European Commission, the Basque Government and through company membership. ESI's work is centred on products and services that are tied directly to core business objectives such as reducing costs and increasing predictability of results among others. ESI's headquarters are in Bilbao, Spain.

ABOUT ESSI...



Enterprises in all developed sectors of the economy - and not just the IT sector - are increasingly dependent on quality software-based IT systems. Such systems support management, production, and service functions in diverse organisations. Furthermore, the products and services now offered by the non-IT sectors, *e.g.*, the automotive industry or the consumer electronics industry, increasingly contain a component of sophisticated software. For example, televisions require in excess of half a Mbyte of software code to provide the wide variety of functions we have come to expect from a domestic appliance. Similarly, the planning and execution of a cutting pattern in the garment industry is accomplished under software control, as are many safety-critical functions in the control of, *e.g.*, aeroplanes, elevators, trains, and electricity generating plants. Today, approximately 70% of all software developed in Europe is developed in the non-IT sectors of the economy. This makes software a technological topic of considerable significance. As the information age develops, software will become even more pervasive and transparent. Consequently, the ability to produce software efficiently, effectively, and with consistently high quality will become increasingly important for all industries across Europe if they are to maintain and enhance their competitiveness.

The goal of the European Systems and Software Initiative (ESSI) is to promote improvements in the software development process in industry, through the take-up of well-founded and established - but insufficiently deployed - methods and technologies, so as to achieve greater efficiency, higher quality, and greater economy. In short, the adoption of Software Best Practice.

ABOUT KVIV...



De Koninklijke Vlaamse Ingenieursvereniging (KVIV) was founded in 1928 and has 12,000 members: civil engineers, agricultural engineers, chemical engineers, bio-engineers, and polytechnical engineers from the Royal Military School.

KVIV is an open and flexible organization that gives ongoing training programs and

publications for entrepreneurs, managers, docents, and government employees. Thanks to its internal structure, KVIV contacts diverse public groups as well as academic and industry circles.

ABOUT SAI...



SAI (Studiecentrum voor Automatische Informatieverwerking) is a non-profit organization whose purpose is to promote the knowledge of Information. As far as the theoretical as practical aspects of information knowledge, the organization takes a stand on social questions that has to do with automation of information as far as it applies. The activities of the organization are focused on people who are specialists in information. SAI organizes discussions, meetings, seminars, workshops and a magazine called "Informatie".

SPONSORED BY...

ABOUT SR/TestWorks...



Software Research, Inc. (SR) offers the TestWorks suite of integrated software test tools that include regression and coverage testing support for UNIX and Windows platforms, in use by thousands of sites worldwide. SR's active program of new-product development includes current applications in Web Testing, UML-Based Test Planning and Development and Remote Testing Technology. SR publishes the widely subscribed TTN-Online, an Emailed monthly newsletter (click to subscribe).

CO-SPONSORS...

GOLD SPONSOR

ABOUT CMG Information Technology...



CMG plc is a leading European IT services company, providing business information solutions through consultancy, systems and services to clients worldwide. Established in 1964, CMG now operates in more than 40 countries from its bases in the UK, The Netherlands, Germany, France and Belgium. The company is listed on the London and Amsterdam stock exchanges. CMG supplies services and products in the finance, trade and industry, transport, telecommunications, energy and public sectors. The Group also provides managed information processing services, including networks, payroll and personnel.

One of CMG's many specialisations is the automation of structured testing. Testing of new software products is a time consuming business. CMG has developed a very successful testing method TestFrame. TestFrame is based on CMG:CAST, an approach for the

structured and automated testing of new or modified software. TestFrame reduces testing times considerably, and thus speeds up the time-to-market for new products and services. With this new method, CMG is currently one of the leading companies in this field.

CMG is dedicated to helping its clients and their people become more successful through the quality of its services and staff. Strong employee commitment ensures the Group's long term success and hence the success of its clients.

GOLD SPONSOR

ABOUT SIM Group Ltd...



SIM Group Ltd. are totally dedicated to software testing. SIM will supply and manage testing teams that perform testing work to the highest standards of efficiency and effectiveness. SIM performs consultancy, execution and training of testing. SIM's methods and techniques for testing include testing frame works, libraries, documented procedures and deliverable templates that can all be used to speed up the adoption and use of good testing practices. SIM's approach to testing relies heavily on the adoption of automated testing techniques as much as is practical. SIM has a proven track record of success with automated testing. SIM's subsidiary SOFTech Tools Ltd. selected and specialised testing tools that contribute to the full automated testing process.

SILVER SPONSOR

ABOUT IQIP Informatica B.V...



Since 1972 IQIP Informatica B.V. has been supporting organisations in carrying out their core processes. With its 1200 employees IQIP does this by constructing, maintaining, testing and implementing application software systems. IQIP increasingly concentrates on carrying out assignments with result responsibility. In testing, IQIP achieves this through her dedicated division Components & Testing (300 employees), using the structured testing approach TMap®. TMap® was developed by IQIP itself and has become a widely used international standard. TMap® related methods such as TAKT (test automation), TSite (test laboratory) and TPI® (Test Process Improvement®) have recently completed the product range. A dedicated R&D team is continuously improving these methods and, if required, develops new products.

SILVER SPONSOR

ABOUT Mercury Interactive...



Mercury Interactive is the world's leader in enterprise application testing solutions. The company offers a comprehensive line of automated testing tools that address the full range of quality needs for testing client/server, e-business, Y2K, Euro, and packaged applications. Its testing solutions enable corporations, system integrators and independent software vendors to identify software errors more quickly and efficiently than with traditional methods.

SILVER SPONSOR

ABOUT The Testing Consultancy...



The Testing Consultancy

The Testing Consultancy is a specialist consultancy that was formed to provide independent testing services within the IT Industry.

With our highly qualified and experienced Consultants and Associates, we are strongly positioned to provide guidance, expertise and support for all your testing requirements.

Our independent and objective advice will help your business to increase the quality and reliability of its systems, whilst actually reducing costs, and delivering those systems within the required timescales.

Gold Leaf Sponsor

ABOUT GiTek Software n.v...



GiTek Software nv supplies a number of services offering a global solution to testing:

- Consultancy in testware
- Structuring of the test process
- Education and training

- Test planning and test management
- Test project and test execution

ORGANIZERS...

ABOUT SR/INSTITUTE...



Software Research Institute (SR/Institute), a not-for-profit subsidiary of Software Research, Inc., was founded to promote the issues of Software Quality throughout the software development community.

In addition to the Quality Week Conference series SR/Institute sponsors continuing education seminars in the general area of software quality and software engineering, and Software Quality Forums at which software quality industry leaders provide state of the art technology transfer to industry executives about how to best apply current technology to immediate software quality needs.

Home	Index	News	Products	Technology	Solutions	App Notes	Screen Shots	Support
Quality Week	Institute	Company	Distributors	Partners	Careers	Users	Information	



Software Research, Inc.
625 Third Street
San Francisco, CA 94107-1997 USA

Phone: +1 (415) 957-1441
TollFree: +1 (800) 942-SOFT [USA Only]
FAX: +1 (415) 957-0730
Email: info@soft.com