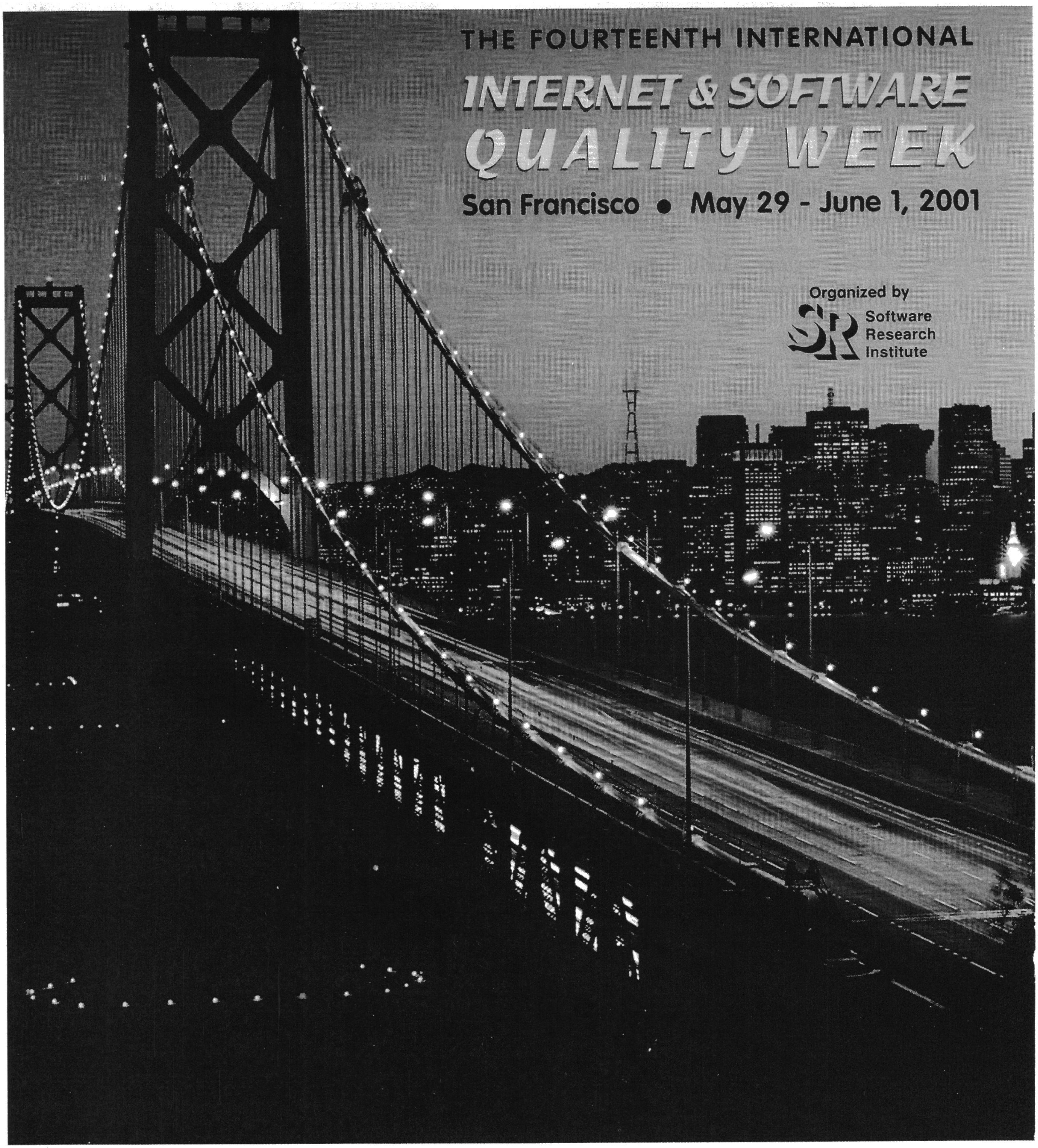


THE FOURTEENTH INTERNATIONAL

INTERNET & SOFTWARE QUALITY WEEK

San Francisco • May 29 - June 1, 2001

Organized by

SR Software Research Institute

cooperation with

rican Society for Quality ASQ tware Division

Industry Sponsors

SR SOFTWARE RESEARCH

COMPUWARE

eValid

TESTING LABS

VANTEON

VeriTest.

Media ponsors

Application Development Trends

Dr. Dobb's JOURNAL

C/C++ Users Journal

SD Times

Software BUSINESS

Windows DEVELOPER'S JOURNAL

Friday, June 1, 2001
QW 2001 Speaker Evaluation

Workshops

□W1 □W2 □W3 □W4

Speaker:____

Overall rating of presentation

(5 = excellent, 1 = poor):

5	4	3	2	1
□	□	□	□	□

Comments:____

____

____

____

____

____

____

____

____

____

____

Please return this form to the registration desk by 10:30 AM, Friday June 1, at the very latest! Thank you!

WORKSHOP NOTES

Fourteenth International
Software & Internet Quality Week 2001

29 May 2001 – 1 June 2001

Marriott Hotel
San Francisco, California USA

Software Research Institute
901 Minnesota Street
San Francisco, CA 94107 USA

Phone: (415) 550-3020 – FAX: (415) 550-3030 – E-mail: qw@soft.com

QW2001 Workshop W1

Dr. Cem Kaner
(Florida Institute of Technology)

Developing The Right Test Documentation

Key Points

- The best approach to project documentation depends on the project context, and not necessarily on published standards.
- There are useful questions for learning your project's requirements for test documentation.
- Context-free questions and lists can guide you in developing good tests and test strategies when you don't have time to develop a full set of documentation.

Presentation Abstract

The best approach to test documentation depends on the project context. For example, a paper-intensive test documentation strategy like IEEE 829 is useful for some projects but can get in the way of development of a high-volume automated testing strategy. In the course of writing the third edition of Testing Computer Software, we are looking at test planning/documentation from a different viewpoint. It seems to us that: * The set of documentation is a deliverable, that can be significantly expensive and that can have a significant impact on the project or the company. * To decide what documentation is appropriate, we should do a requirements analysis, asking

- who are the favored, disfavored, and ignored users/recipients of this documentation and why
- what they need or want, and why
- what it costs to fully or partially satisfy these requirements

We also consider the content of the test plan. We think that we have some guidance to offer in terms of evaluating the coverage of the test documentation (how well different aspects of the product are covered and how well different risks are covered). The associated paper will present some specific test documentation techniques that we use (various types of charts), that we have taught before. This presentation pulls together work from other talks and from the Los Altos Workshops on Software Testing. There is a lot of material. It can easily fill a day's tutorial and it can be scaled back to a shorter session (45-90 minutes) that is supported by a long paper.

About the Author

Cem Kaner is Professor of Computer Sciences at the Florida Institute of Technology. Prior to joining Florida Tech, Kaner worked in Silicon Valley for 17 years, doing and managing programming, user interface design, testing, and user documentation. He is the senior author (with Jack Falk and Hung Quoc Nguyen) of TESTING COMPUTER SOFTWARE (2nd Edition) and (with David Pels) of BAD SOFTWARE: WHAT TO DO WHEN SOFTWARE FAILS.

Through his consulting firm, KANER.COM, he teaches courses on black box software testing and consults to software publishers on software testing, documentation, and development management. Kaner is also the co-founder and co-host of the Los Altos Workshop on Software Testing, the Software Test Managers' RoundTable, the Workshop on Heuristic & Exploratory Techniques, and the Florida Workshops on Model-Based Testing. Kaner is also attorney whose practice is focused on the law of software quality. He is active (as an advocate for customers, authors, and small development shops) in several legislative drafting efforts involving software licensing, software quality regulation, and electronic commerce.

Kaner holds a B.A. in Arts & Sciences (Math, Philosophy), a Ph.D. in Experimental Psychology (Human Perception & Performance: Psychophysics), and a J.D. (law degree). He is Certified in Quality Engineering by the American Society for Quality.

Developing the Right Test Documentation

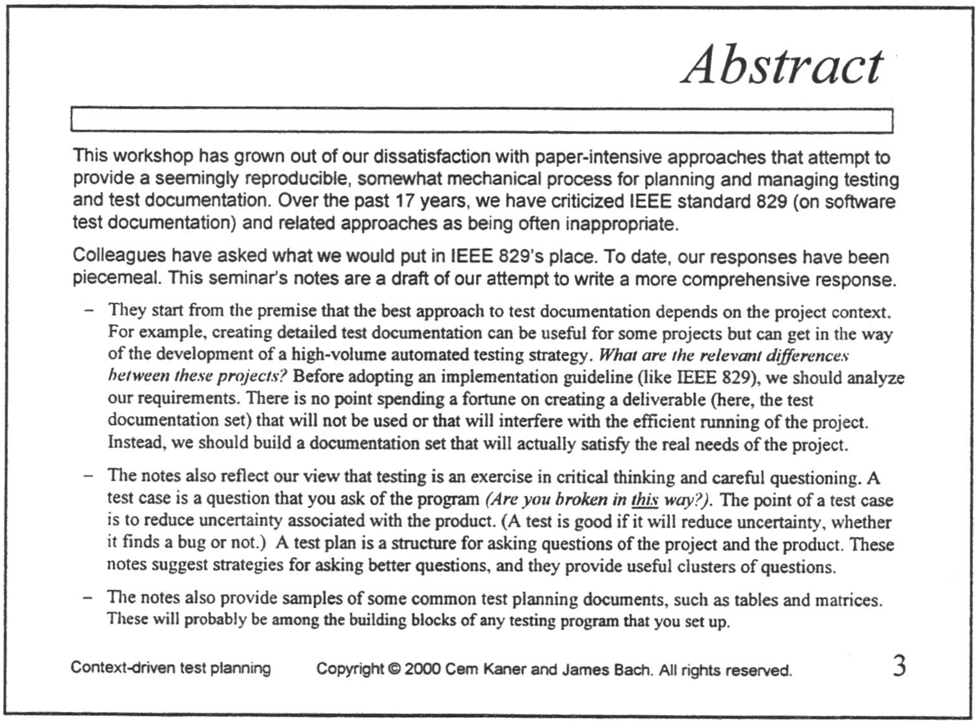
Cem Kaner, J.D., Ph.D.
Department of Computer Sciences
Florida Institute of Technology

James Bach
Satisfice, Inc.

May, 2001
Software Quality Week

Acknowledgments

- These notes outline the test planning chapter in prep for Testing Computer Software, 3rd Ed., by Cem Kaner, James Bach, Hung Quoc Nguyen, Jack Falk, Brian Lawrence & Bob Johnson. They incorporate and adapt materials by these authors.
- Many of the ideas in these notes were reviewed and refined at the Third Los Altos Workshop on Software Testing (LAWST), February 7-8, 1998, and at the Eleventh LAWST, October 28-29, 2000.
  - The participants at LAWST 3 were: Chris Agruss, James Bach, Karla Fisher, David Gelperin, Kenneth Groder, Elisabeth Hendrickson, Doug Hoffman, III (recorder), Bob Johnson, Cem Kaner (host), Brian Lawrence (facilitator), Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Johanna Rothman, Jane Stepak, Melora Svoboda, Jeremy White, and Rodney Wilson.
  - The participants at LAWST 11 were: Chris Agruss, James Bach, Hans Buwalda, Marge Farrell. Sam Guckenheimer, Elisabeth Hendrickson, Doug Hoffman, III (recorder), Bob Johnson, Karen Johnson, Cem Kaner (host), Brian Lawrence (facilitator), Alan Myrvold, Hung Quoc Nguyen, Noel Nyman, Neal Reizer, Amit Singh, and Melora Svoboda.

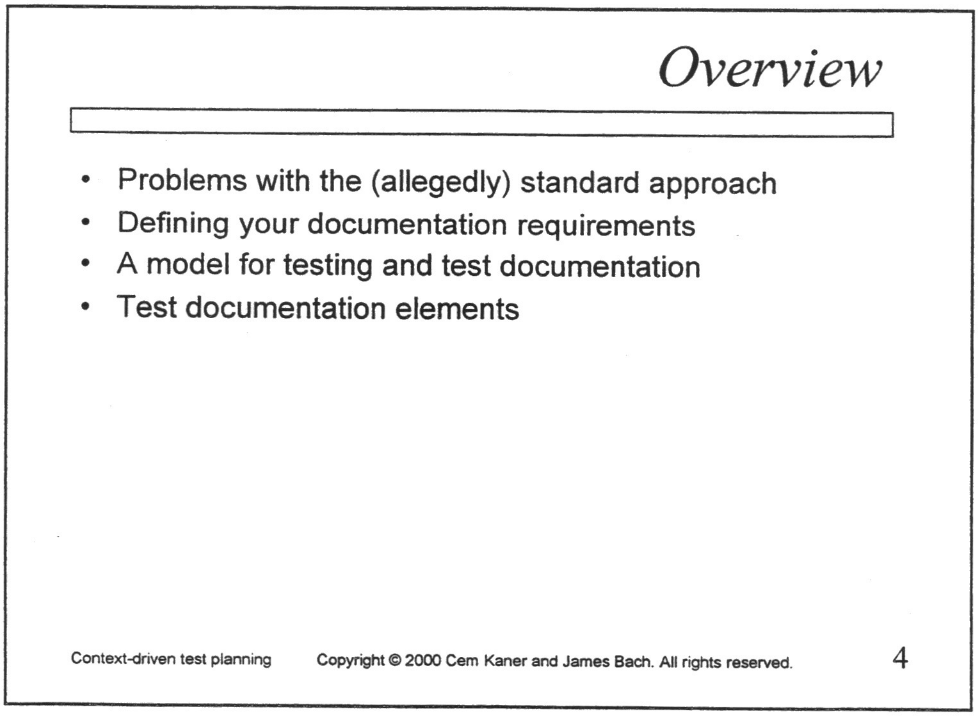


Abstract

This workshop has grown out of our dissatisfaction with paper-intensive approaches that attempt to provide a seemingly reproducible, somewhat mechanical process for planning and managing testing and test documentation. Over the past 17 years, we have criticized IEEE standard 829 (on software test documentation) and related approaches as being often inappropriate.

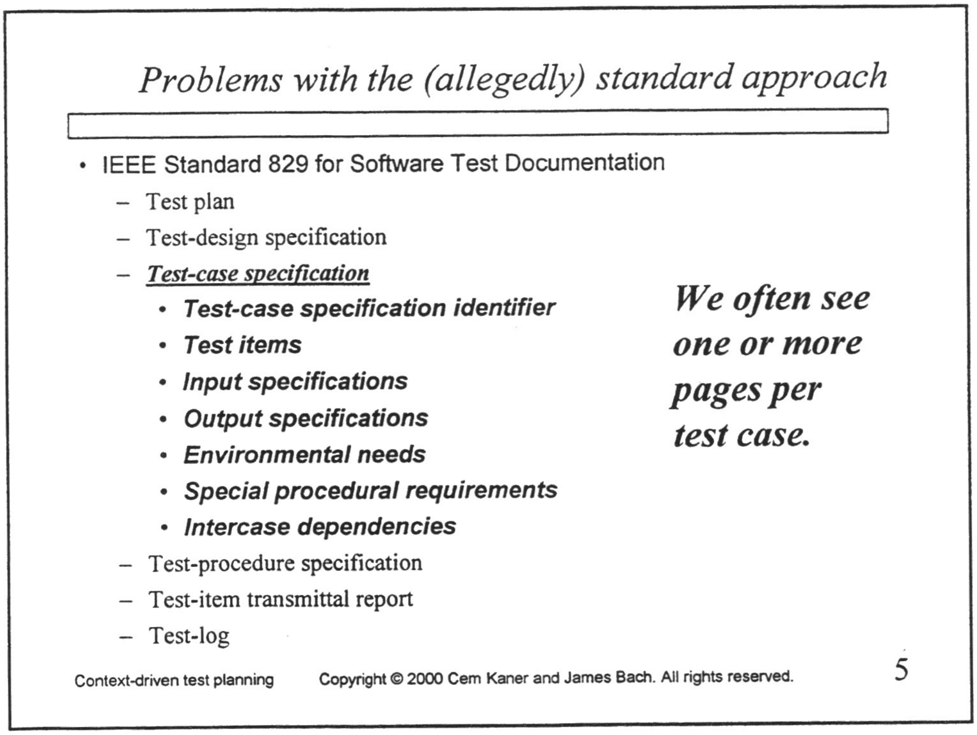
Colleagues have asked what we would put in IEEE 829's place. To date, our responses have been piecemeal. This seminar's notes are a draft of our attempt to write a more comprehensive response.

- They start from the premise that the best approach to test documentation depends on the project context. For example, creating detailed test documentation can be useful for some projects but can get in the way of the development of a high-volume automated testing strategy. *What are the relevant differences between these projects?* Before adopting an implementation guideline (like IEEE 829), we should analyze our requirements. There is no point spending a fortune on creating a deliverable (here, the test documentation set) that will not be used or that will interfere with the efficient running of the project. Instead, we should build a documentation set that will actually satisfy the real needs of the project.
- The notes also reflect our view that testing is an exercise in critical thinking and careful questioning. A test case is a question that you ask of the program *(Are you broken in this way?)*. The point of a test case is to reduce uncertainty associated with the product. (A test is good if it will reduce uncertainty, whether it finds a bug or not.) A test plan is a structure for asking questions of the project and the product. These notes suggest strategies for asking better questions, and they provide useful clusters of questions.
- The notes also provide samples of some common test planning documents, such as tables and matrices. These will probably be among the building blocks of any testing program that you set up.



Overview

- Problems with the (allegedly) standard approach
- Defining your documentation requirements
- A model for testing and test documentation
- Test documentation elements



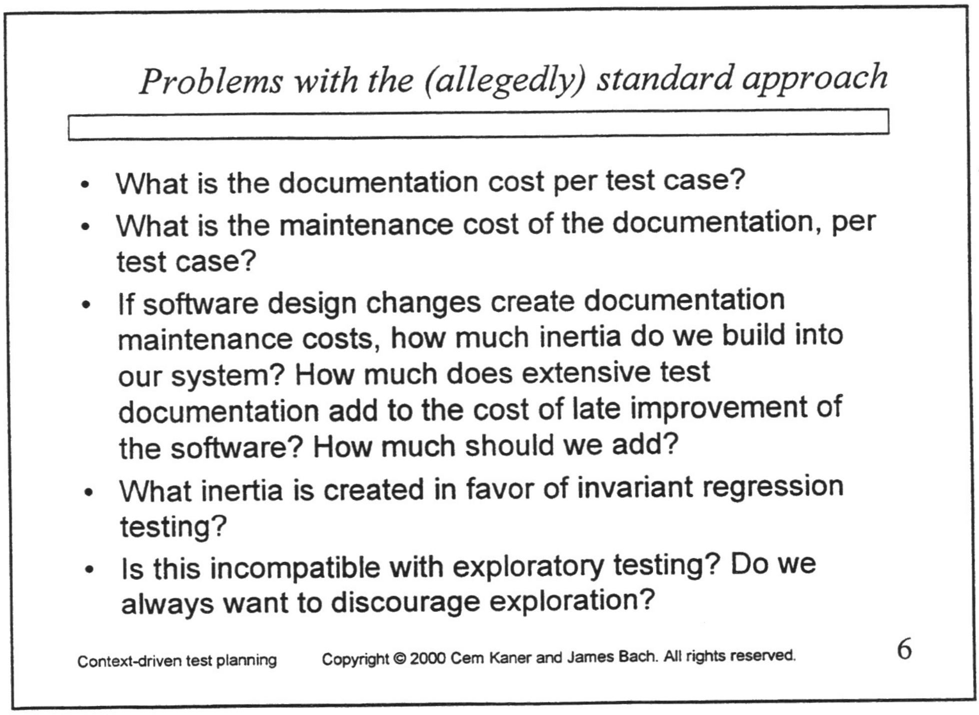
Problems with the (allegedly) standard approach

- IEEE Standard 829 for Software Test Documentation
  - Test plan
  - Test-design specification
  - ***Test-case specification***
    - ***Test-case specification identifier***
    - ***Test items***
    - ***Input specifications***
    - ***Output specifications***
    - ***Environmental needs***
    - ***Special procedural requirements***
    - ***Intercase dependencies***
  - Test-procedure specification
  - Test-item transmittal report
  - Test-log

We often see one or more pages per test case.

Context-driven test planning

5

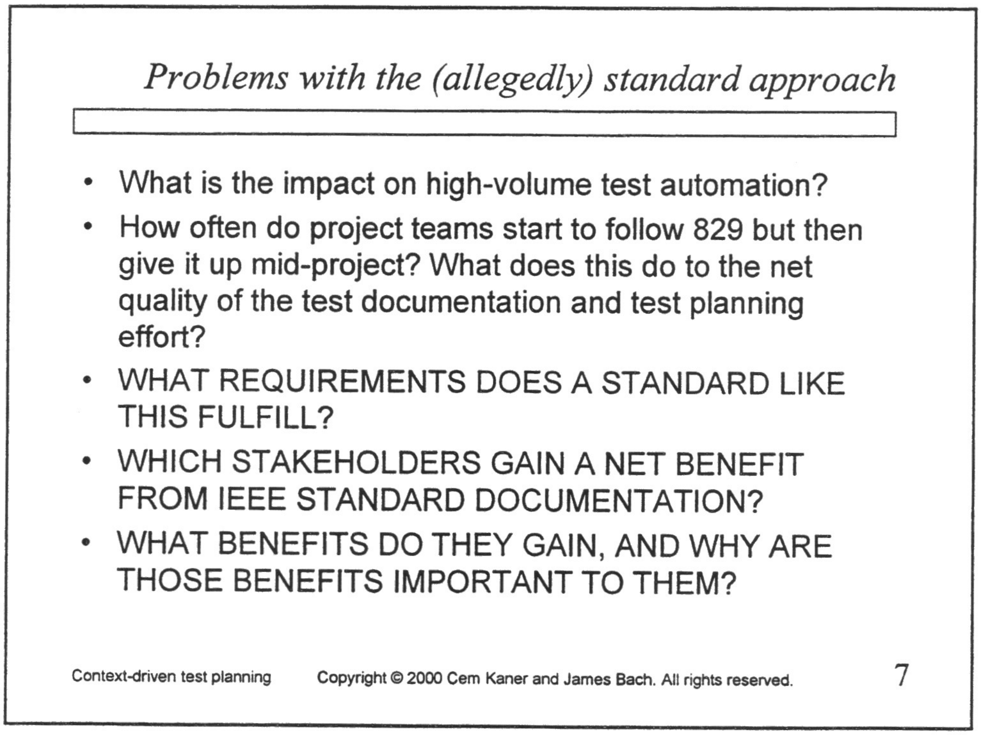


Problems with the (allegedly) standard approach

- What is the documentation cost per test case?
- What is the maintenance cost of the documentation, per test case?
- If software design changes create documentation maintenance costs, how much inertia do we build into our system? How much does extensive test documentation add to the cost of late improvement of the software? How much should we add?
- What inertia is created in favor of invariant regression testing?
- Is this incompatible with exploratory testing? Do we always want to discourage exploration?

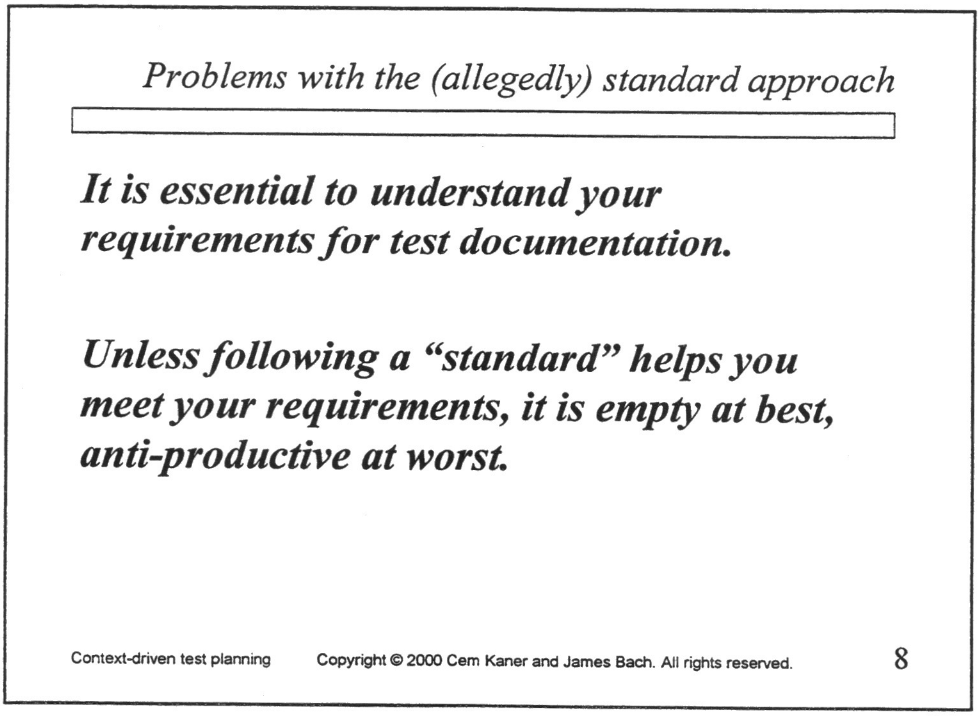
Context-driven test planning

6



Problems with the (allegedly) standard approach

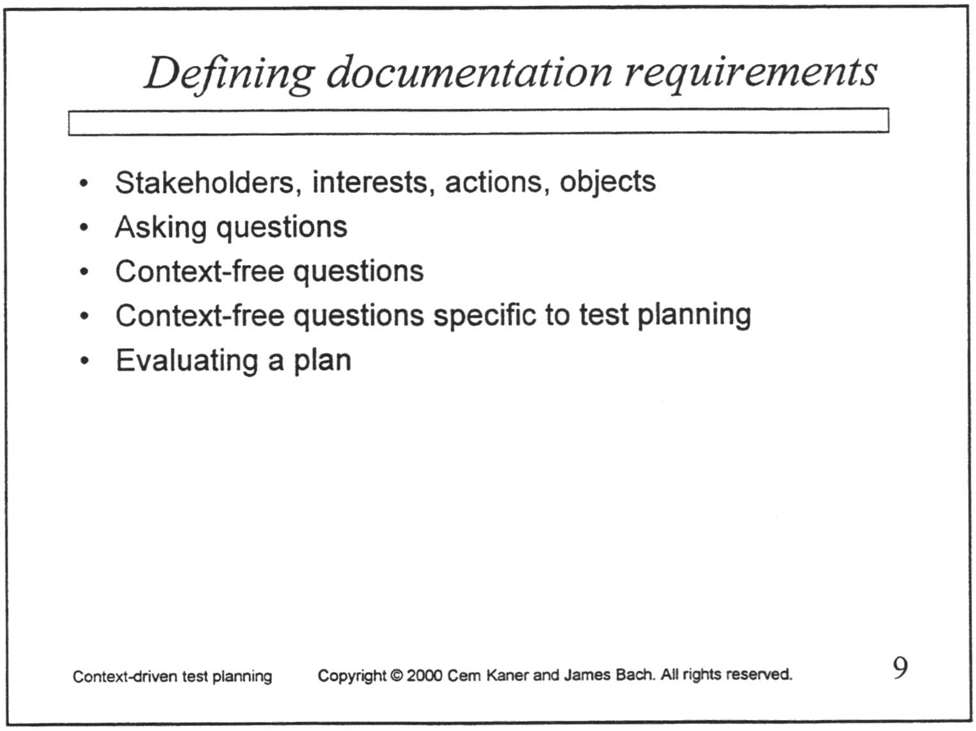
- What is the impact on high-volume test automation?
- How often do project teams start to follow 829 but then give it up mid-project? What does this do to the net quality of the test documentation and test planning effort?
- WHAT REQUIREMENTS DOES A STANDARD LIKE THIS FULFILL?
- WHICH STAKEHOLDERS GAIN A NET BENEFIT FROM IEEE STANDARD DOCUMENTATION?
- WHAT BENEFITS DO THEY GAIN, AND WHY ARE THOSE BENEFITS IMPORTANT TO THEM?



Problems with the (allegedly) standard approach

It is essential to understand your requirements for test documentation.

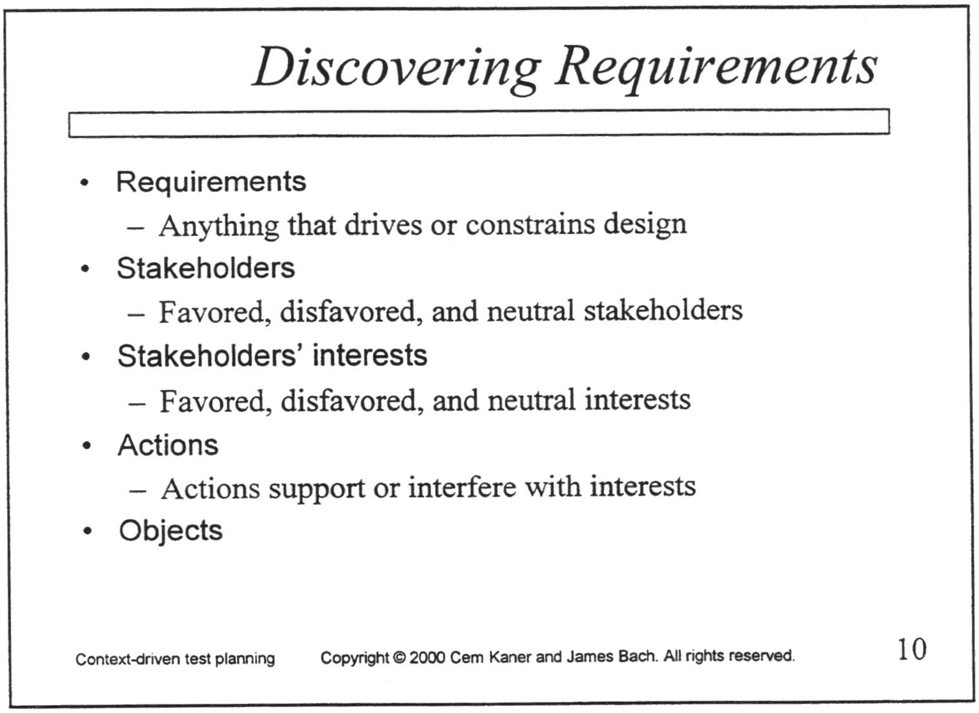
Unless following a "standard" helps you meet your requirements, it is empty at best, anti-productive at worst.



Defining documentation requirements

- Stakeholders, interests, actions, objects
- Asking questions
- Context-free questions
- Context-free questions specific to test planning
- Evaluating a plan

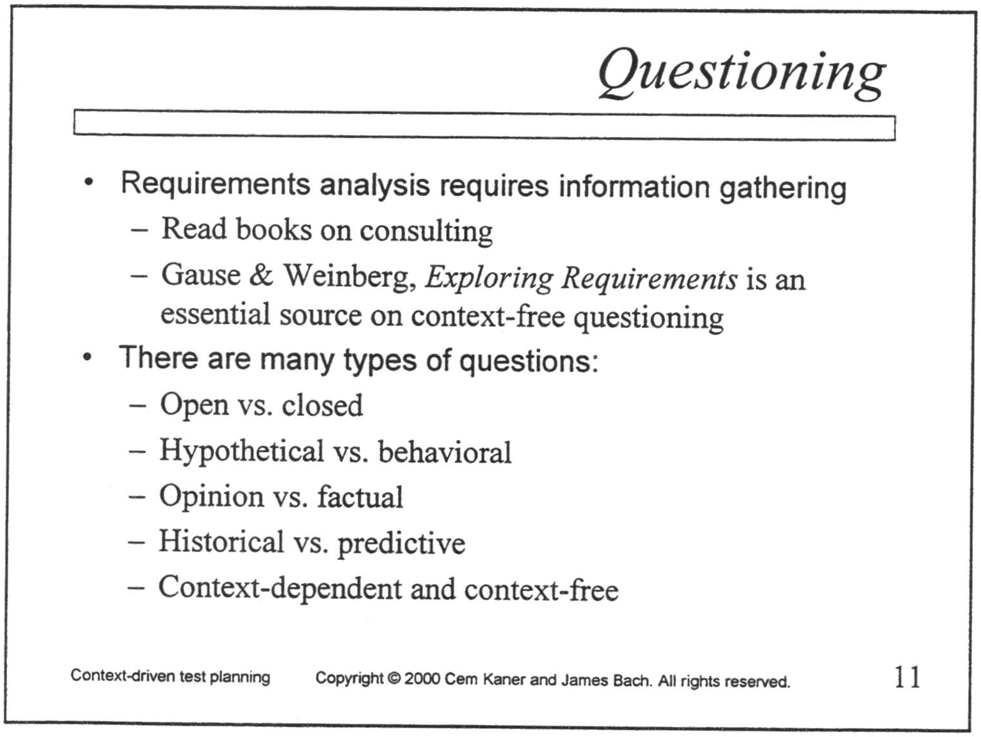
Context-driven test planning Copyright © 2000 Cem Kaner and James Bach. All rights reserved. 9



Discovering Requirements

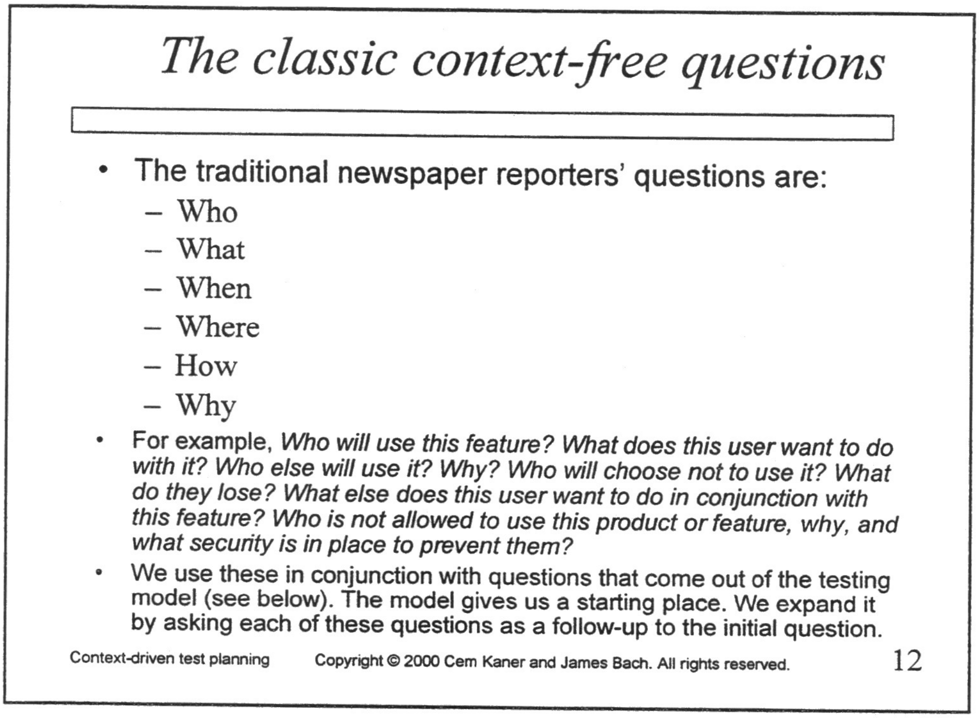
- Requirements
  - Anything that drives or constrains design
- Stakeholders
  - Favored, disfavored, and neutral stakeholders
- Stakeholders' interests
  - Favored, disfavored, and neutral interests
- Actions
  - Actions support or interfere with interests
- Objects

Context-driven test planning Copyright © 2000 Cem Kaner and James Bach. All rights reserved. 10



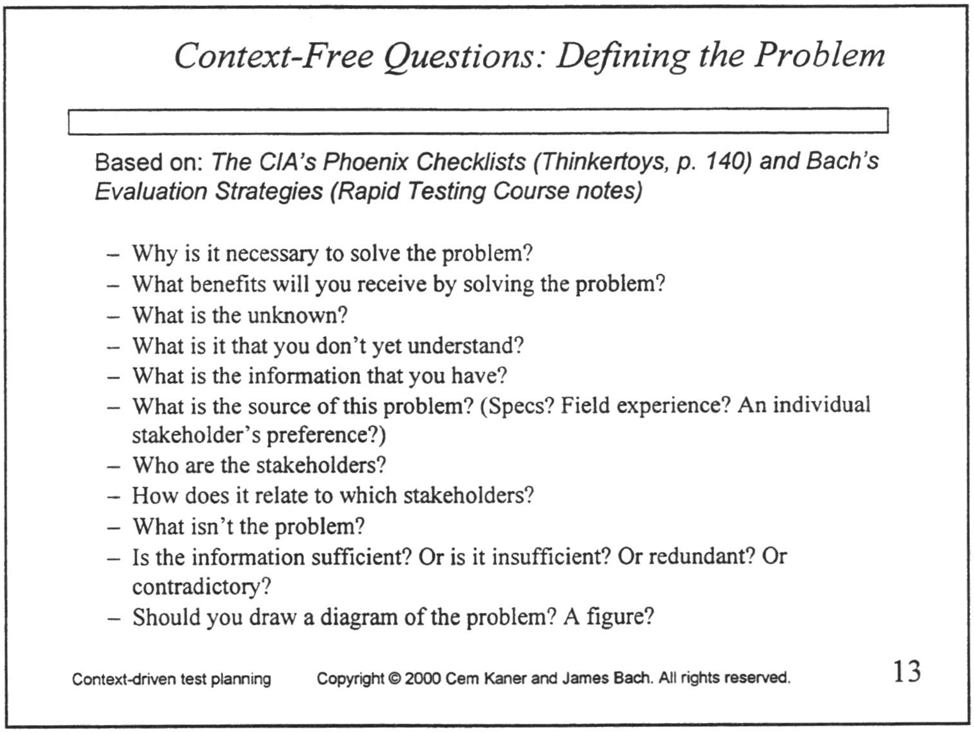
Questioning

- Requirements analysis requires information gathering
  - Read books on consulting
  - Gause & Weinberg, *Exploring Requirements* is an essential source on context-free questioning
- There are many types of questions:
  - Open vs. closed
  - Hypothetical vs. behavioral
  - Opinion vs. factual
  - Historical vs. predictive
  - Context-dependent and context-free



The classic context-free questions

- The traditional newspaper reporters' questions are:
  - Who
  - What
  - When
  - Where
  - How
  - Why
- For example, *Who will use this feature? What does this user want to do with it? Who else will use it? Why? Who will choose not to use it? What do they lose? What else does this user want to do in conjunction with this feature? Who is not allowed to use this product or feature, why, and what security is in place to prevent them?*
- We use these in conjunction with questions that come out of the testing model (see below). The model gives us a starting place. We expand it by asking each of these questions as a follow-up to the initial question.

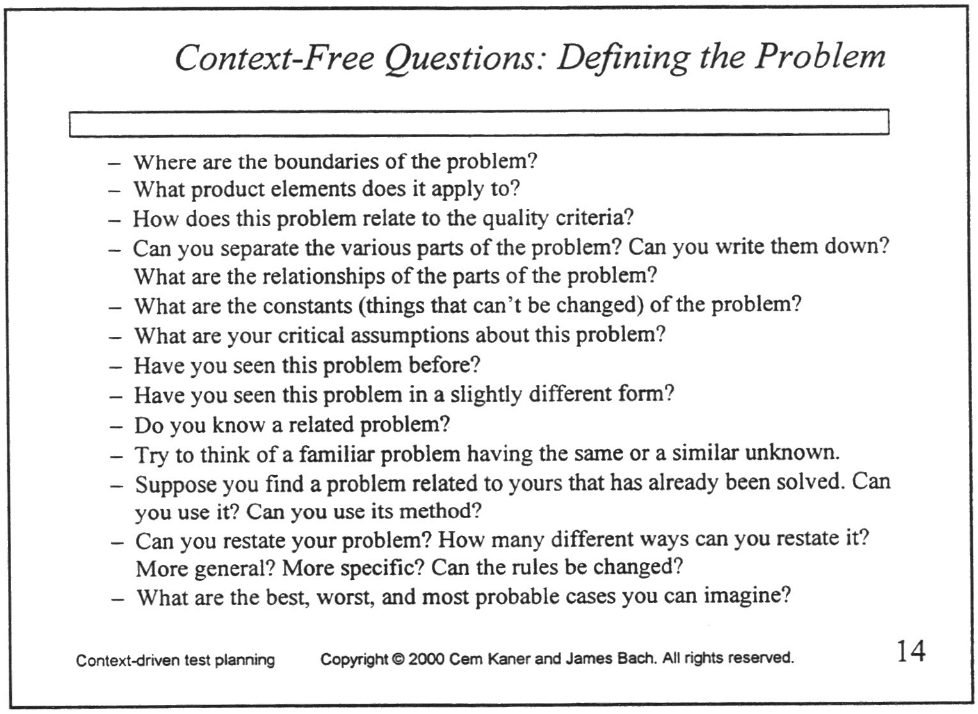


Context-Free Questions: Defining the Problem

Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140) and Bach's Evaluation Strategies (Rapid Testing Course notes)*

- Why is it necessary to solve the problem?
- What benefits will you receive by solving the problem?
- What is the unknown?
- What is it that you don't yet understand?
- What is the information that you have?
- What is the source of this problem? (Specs? Field experience? An individual stakeholder's preference?)
- Who are the stakeholders?
- How does it relate to which stakeholders?
- What isn't the problem?
- Is the information sufficient? Or is it insufficient? Or redundant? Or contradictory?
- Should you draw a diagram of the problem? A figure?

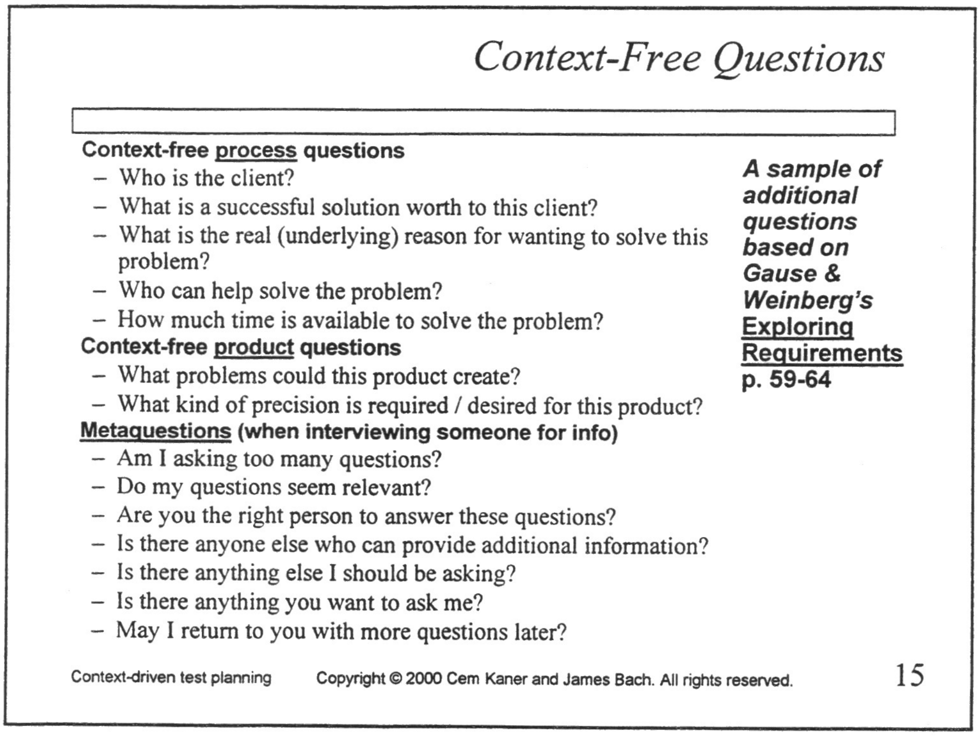
Context-driven test planning 13



Context-Free Questions: Defining the Problem

- Where are the boundaries of the problem?
- What product elements does it apply to?
- How does this problem relate to the quality criteria?
- Can you separate the various parts of the problem? Can you write them down? What are the relationships of the parts of the problem?
- What are the constants (things that can't be changed) of the problem?
- What are your critical assumptions about this problem?
- Have you seen this problem before?
- Have you seen this problem in a slightly different form?
- Do you know a related problem?
- Try to think of a familiar problem having the same or a similar unknown.
- Suppose you find a problem related to yours that has already been solved. Can you use it? Can you use its method?
- Can you restate your problem? How many different ways can you restate it? More general? More specific? Can the rules be changed?
- What are the best, worst, and most probable cases you can imagine?

Context-driven test planning 14



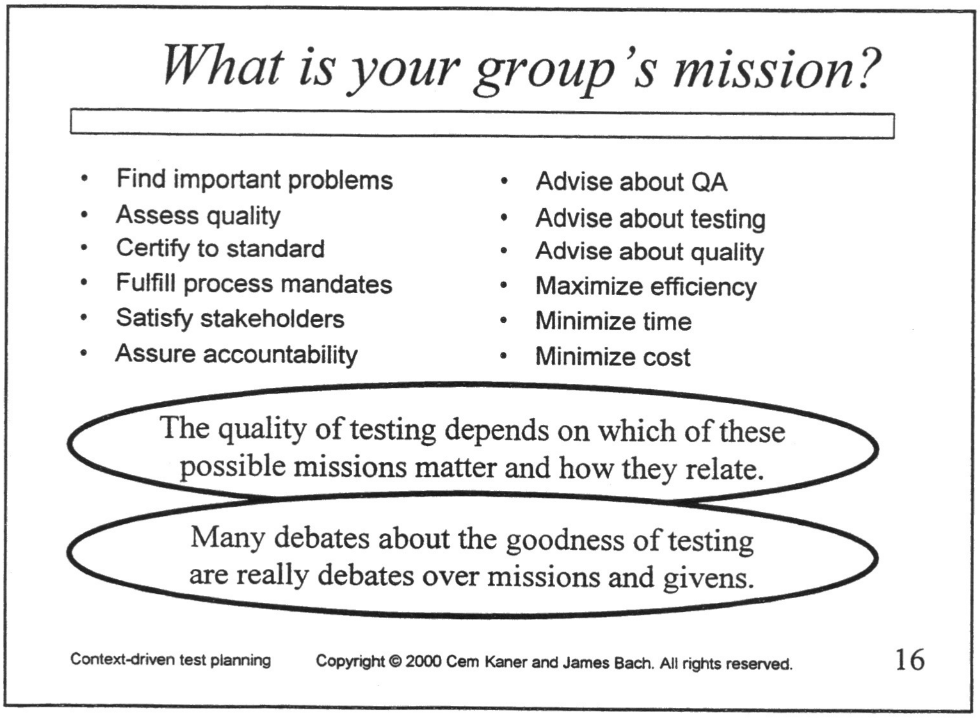
Context-Free Questions

Context-free process questions
- Who is the client?
- What is a successful solution worth to this client?
- What is the real (underlying) reason for wanting to solve this problem?
- Who can help solve the problem?
- How much time is available to solve the problem?

Context-free product questions
- What problems could this product create?
- What kind of precision is required / desired for this product?

Metaquestions (when interviewing someone for info)
- Am I asking too many questions?
- Do my questions seem relevant?
- Are you the right person to answer these questions?
- Is there anyone else who can provide additional information?
- Is there anything else I should be asking?
- Is there anything you want to ask me?
- May I return to you with more questions later?

A sample of additional questions based on Gause & Weinberg's Exploring Requirements p. 59-64

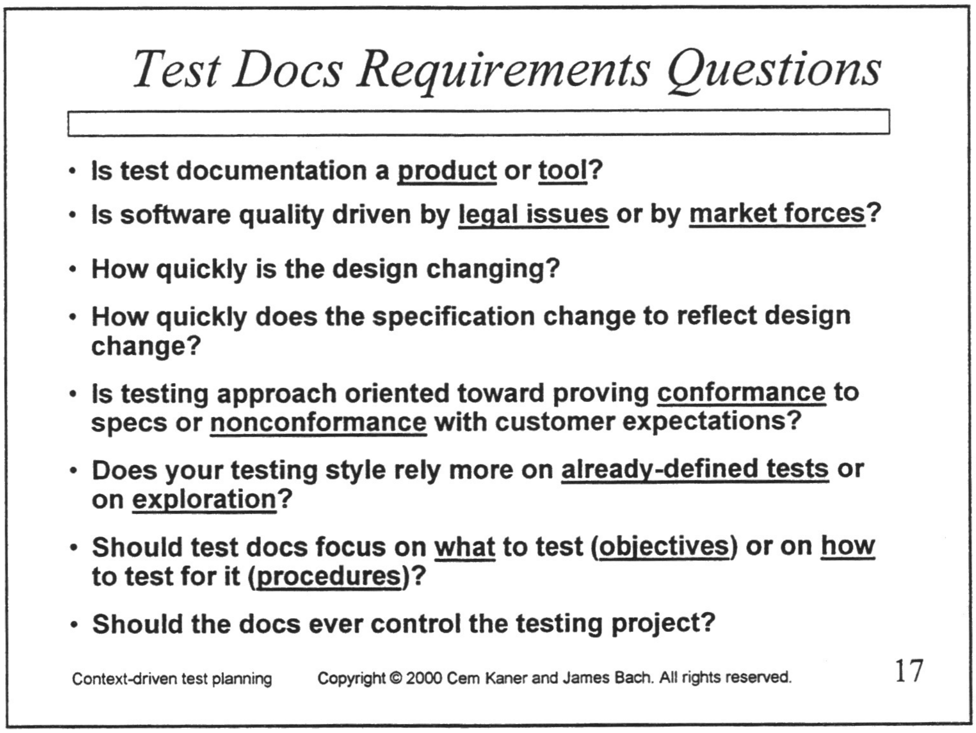


What is your group's mission?

- Find important problems
- Assess quality
- Certify to standard
- Fulfill process mandates
- Satisfy stakeholders
- Assure accountability
- Advise about QA
- Advise about testing
- Advise about quality
- Maximize efficiency
- Minimize time
- Minimize cost

The quality of testing depends on which of these possible missions matter and how they relate.

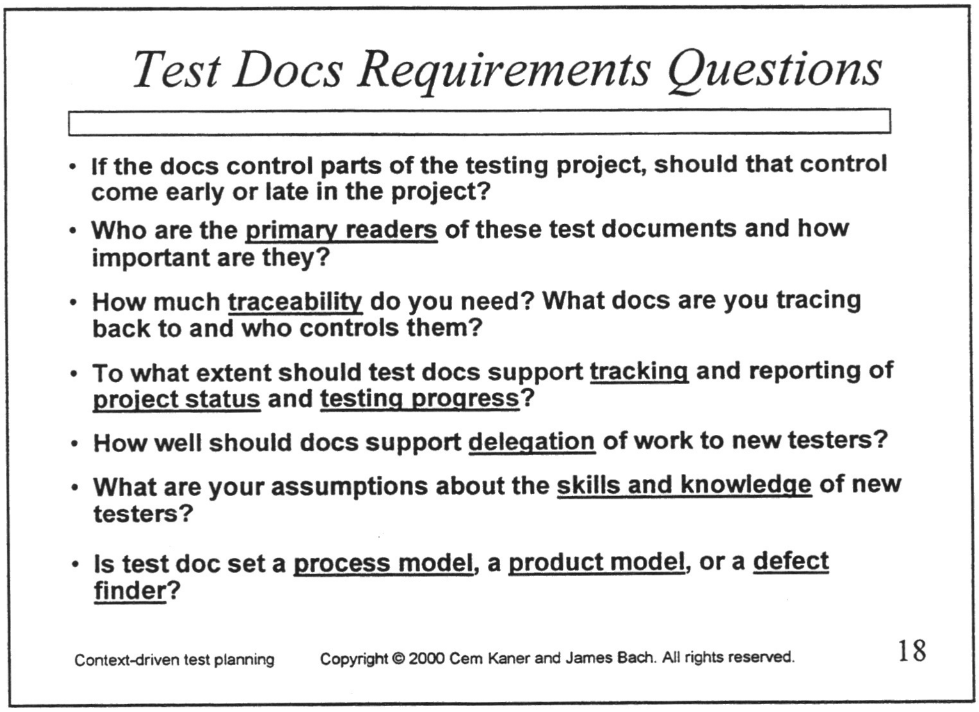
Many debates about the goodness of testing are really debates over missions and givens.



Test Docs Requirements Questions

- Is test documentation a product or tool?
- Is software quality driven by legal issues or by market forces?
- How quickly is the design changing?
- How quickly does the specification change to reflect design change?
- Is testing approach oriented toward proving conformance to specs or nonconformance with customer expectations?
- Does your testing style rely more on already-defined tests or on exploration?
- Should test docs focus on what to test (objectives) or on how to test for it (procedures)?
- Should the docs ever control the testing project?

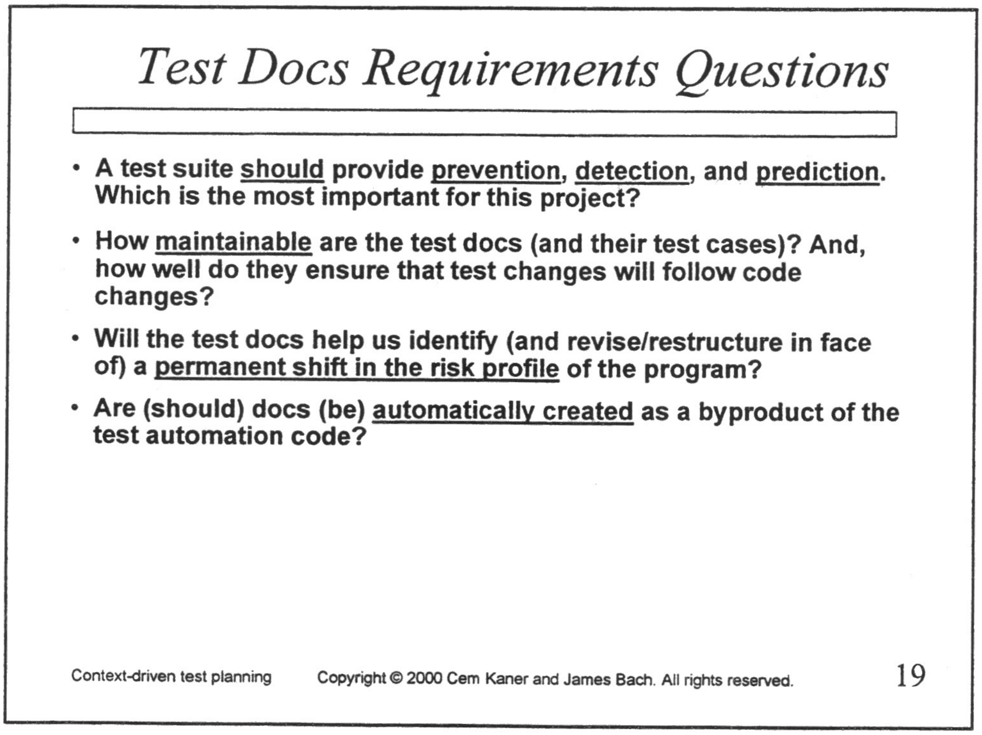
Context-driven test planning 17



Test Docs Requirements Questions

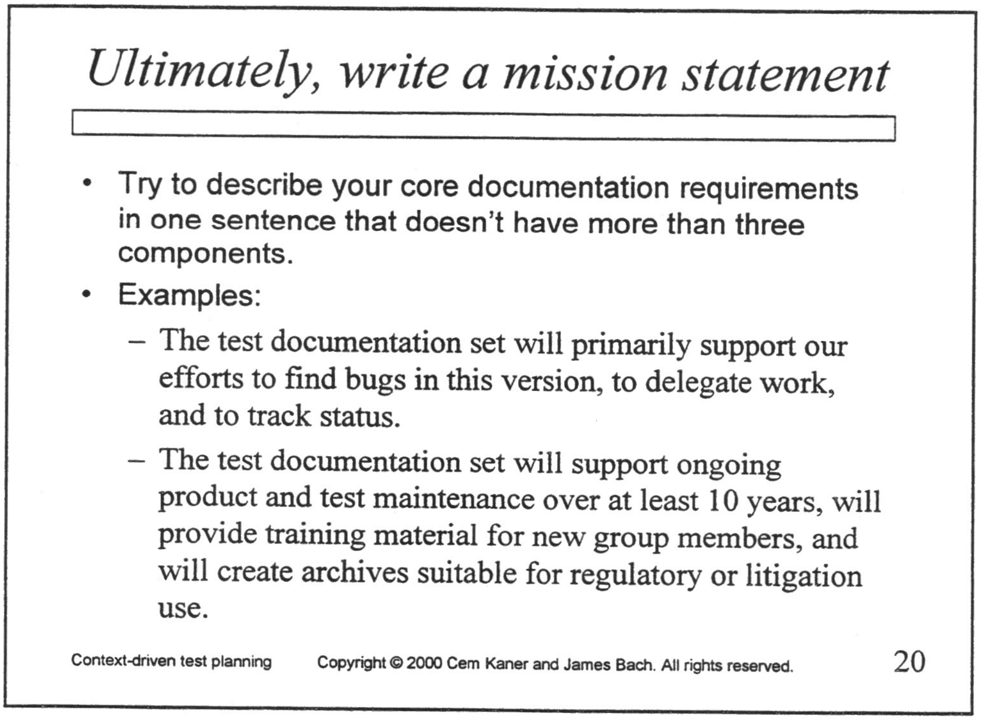
- If the docs control parts of the testing project, should that control come early or late in the project?
- Who are the primary readers of these test documents and how important are they?
- How much traceability do you need? What docs are you tracing back to and who controls them?
- To what extent should test docs support tracking and reporting of project status and testing progress?
- How well should docs support delegation of work to new testers?
- What are your assumptions about the skills and knowledge of new testers?
- Is test doc set a process model, a product model, or a defect finder?

Context-driven test planning 18



Test Docs Requirements Questions

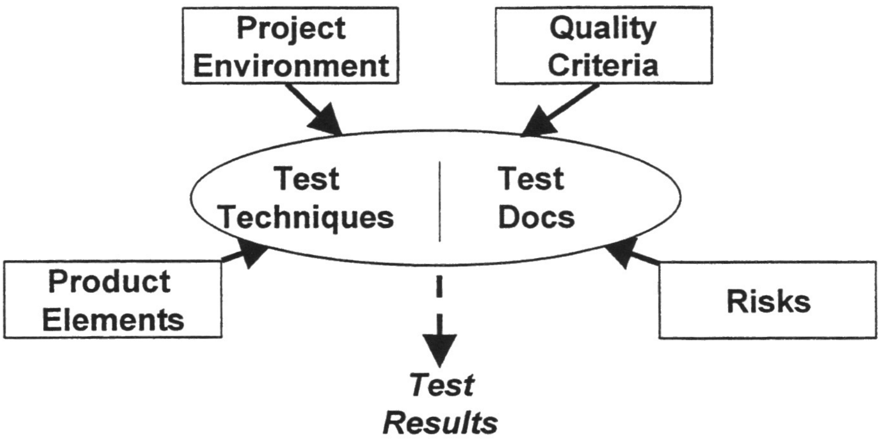
- A test suite should provide prevention, detection, and prediction. Which is the most important for this project?
- How maintainable are the test docs (and their test cases)? And, how well do they ensure that test changes will follow code changes?
- Will the test docs help us identify (and revise/restructure in face of) a permanent shift in the risk profile of the program?
- Are (should) docs (be) automatically created as a byproduct of the test automation code?



Ultimately, write a mission statement

- Try to describe your core documentation requirements in one sentence that doesn't have more than three components.
- Examples:
  - The test documentation set will primarily support our efforts to find bugs in this version, to delegate work, and to track status.
  - The test documentation set will support ongoing product and test maintenance over at least 10 years, will provide training material for new group members, and will create archives suitable for regulatory or litigation use.

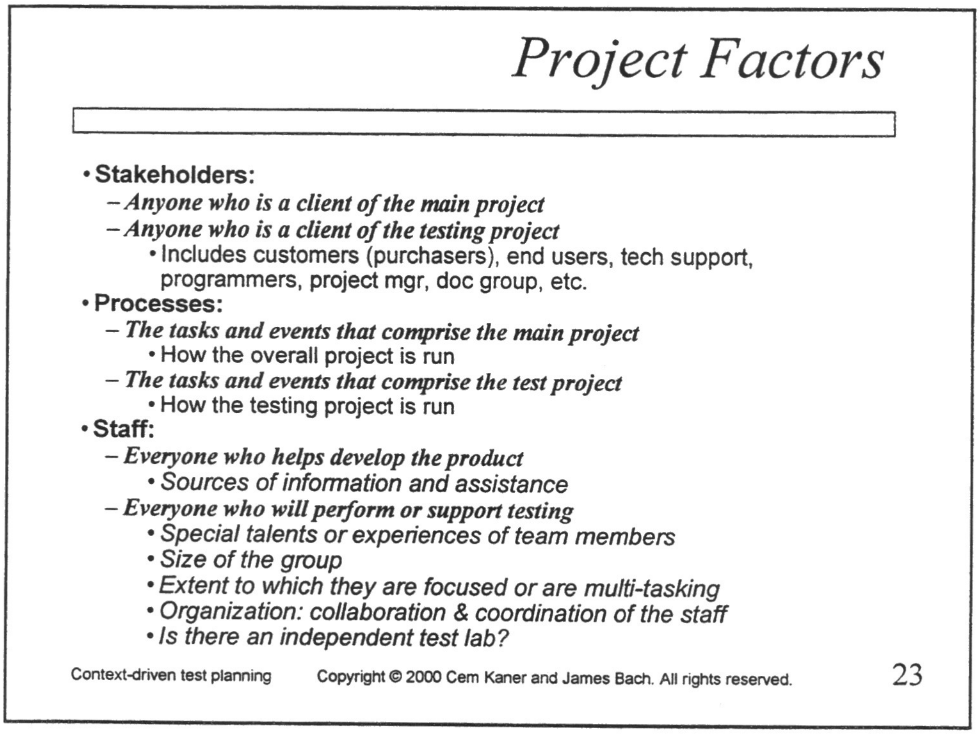
A Model of Software Testing



Project Environment Factors:

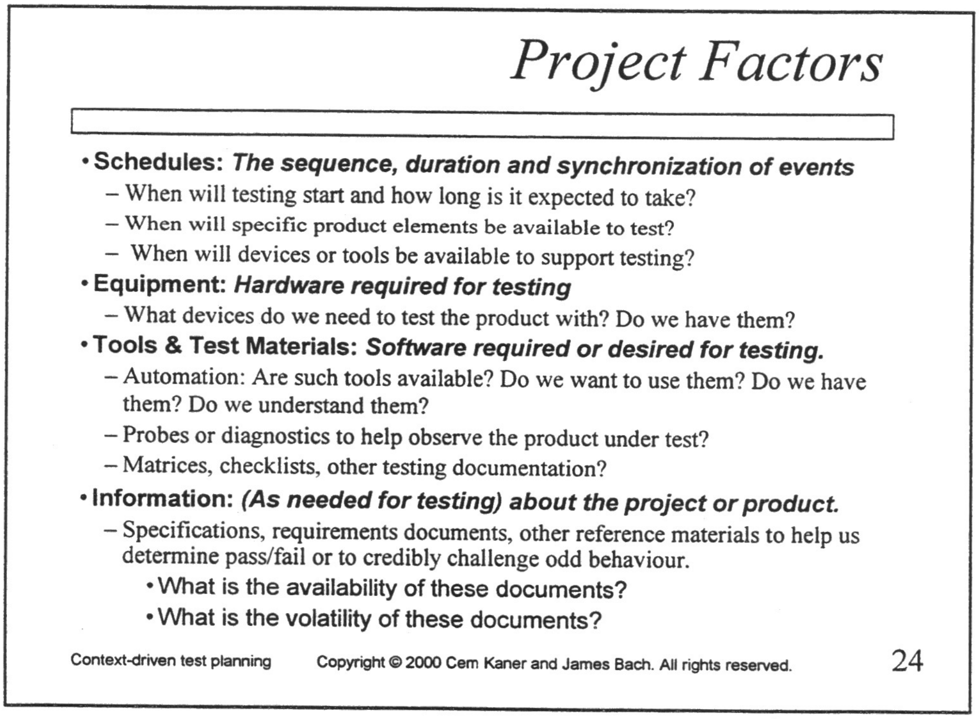
- Stakeholders
- Processes
- Staff
- Schedules
- Equipment
- Tools & Test Materials
- Information
- Items Under Test
- Logistics
- Budget
- Deliverables

These aspects of the environment constrain and enable the testing project



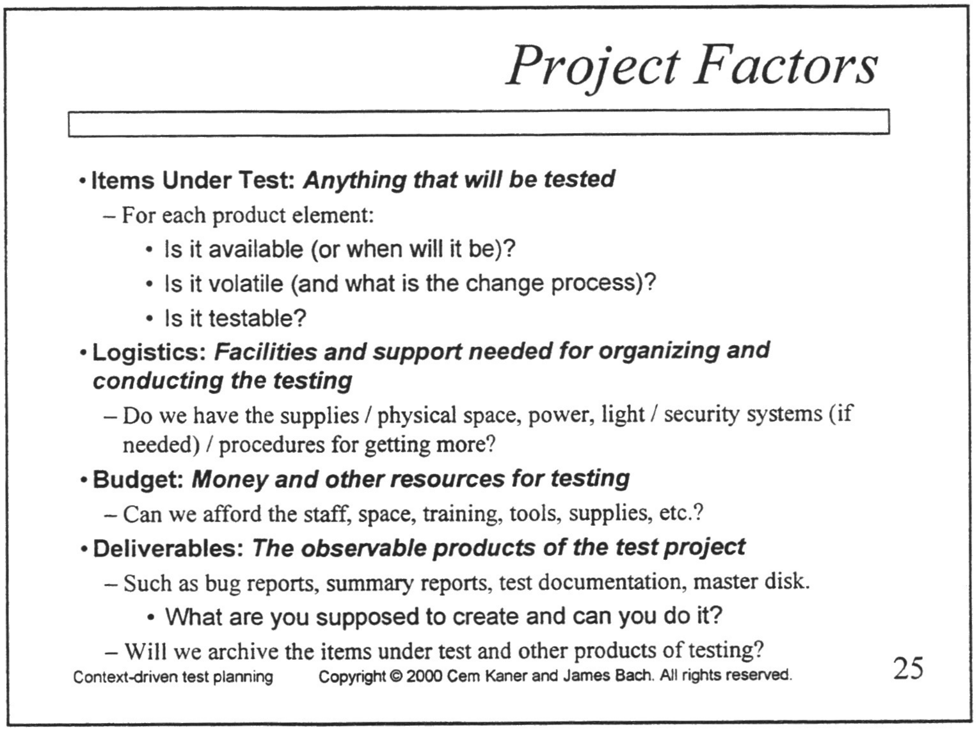
Project Factors

- **Stakeholders:**
  - ***Anyone who is a client of the main project***
  - ***Anyone who is a client of the testing project***
    - Includes customers (purchasers), end users, tech support, programmers, project mgr, doc group, etc.
- **Processes:**
  - ***The tasks and events that comprise the main project***
    - How the overall project is run
  - ***The tasks and events that comprise the test project***
    - How the testing project is run
- **Staff:**
  - ***Everyone who helps develop the product***
    - *Sources of information and assistance*
  - ***Everyone who will perform or support testing***
    - *Special talents or experiences of team members*
    - *Size of the group*
    - *Extent to which they are focused or are multi-tasking*
    - *Organization: collaboration & coordination of the staff*
    - *Is there an independent test lab?*



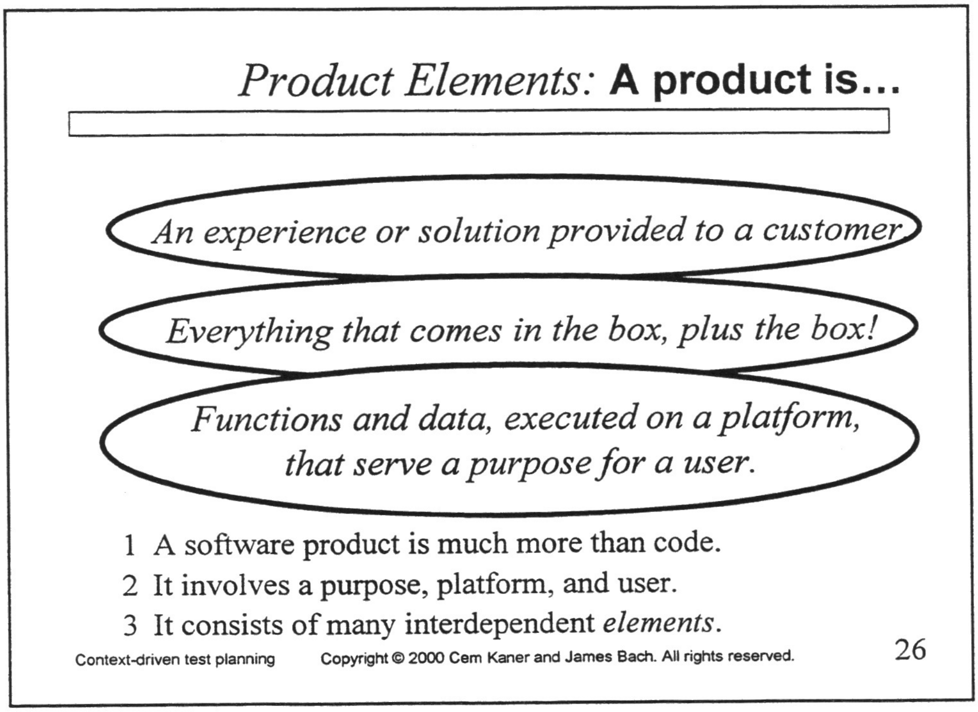
Project Factors

- **Schedules:** ***The sequence, duration and synchronization of events***
  - When will testing start and how long is it expected to take?
  - When will specific product elements be available to test?
  - When will devices or tools be available to support testing?
- **Equipment:** ***Hardware required for testing***
  - What devices do we need to test the product with? Do we have them?
- **Tools & Test Materials:** ***Software required or desired for testing.***
  - Automation: Are such tools available? Do we want to use them? Do we have them? Do we understand them?
  - Probes or diagnostics to help observe the product under test?
  - Matrices, checklists, other testing documentation?
- **Information:** ***(As needed for testing) about the project or product.***
  - Specifications, requirements documents, other reference materials to help us determine pass/fail or to credibly challenge odd behaviour.
    - What is the availability of these documents?
    - What is the volatility of these documents?



Project Factors

- **Items Under Test: *Anything that will be tested***
  - For each product element:
    - Is it available (or when will it be)?
    - Is it volatile (and what is the change process)?
    - Is it testable?
- **Logistics: *Facilities and support needed for organizing and conducting the testing***
  - Do we have the supplies / physical space, power, light / security systems (if needed) / procedures for getting more?
- **Budget: *Money and other resources for testing***
  - Can we afford the staff, space, training, tools, supplies, etc.?
- **Deliverables: *The observable products of the test project***
  - Such as bug reports, summary reports, test documentation, master disk.
    - What are you supposed to create and can you do it?
  - Will we archive the items under test and other products of testing?



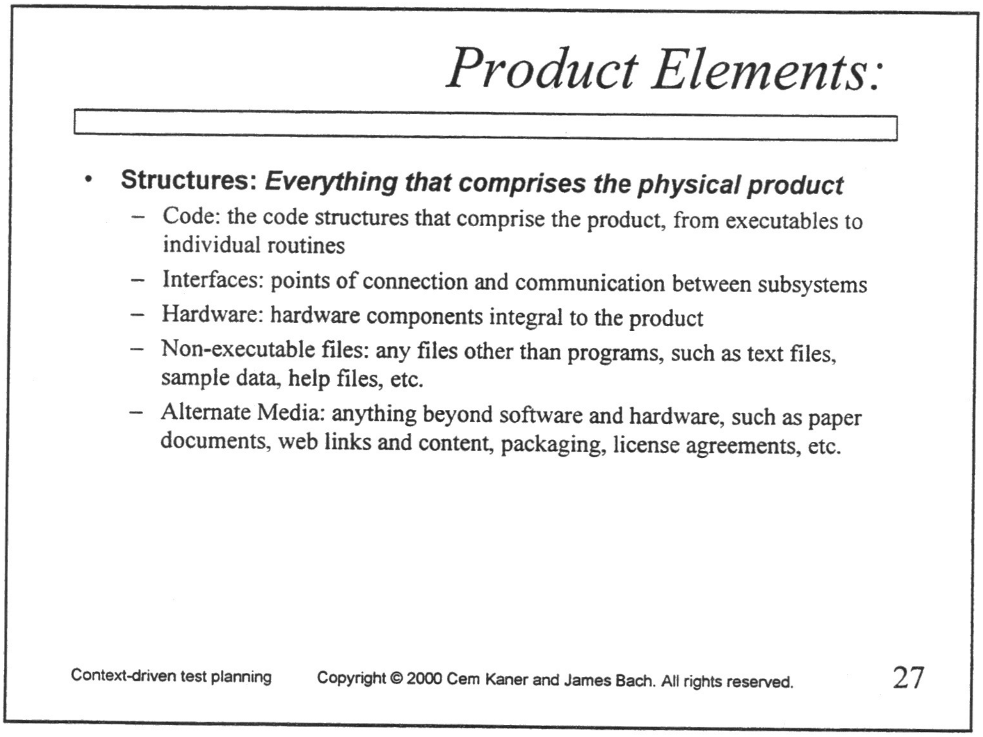
Product Elements: **A product is…**

An experience or solution provided to a customer

Everything that comes in the box, plus the box!

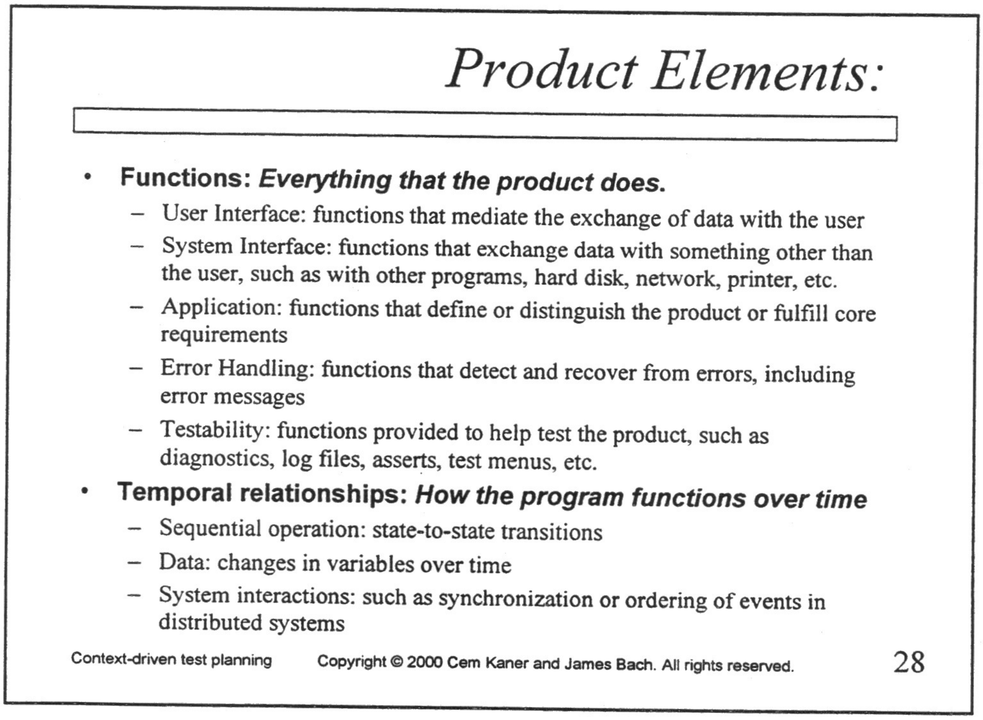
Functions and data, executed on a platform, that serve a purpose for a user.

1 A software product is much more than code.
2 It involves a purpose, platform, and user.
3 It consists of many interdependent *elements*.



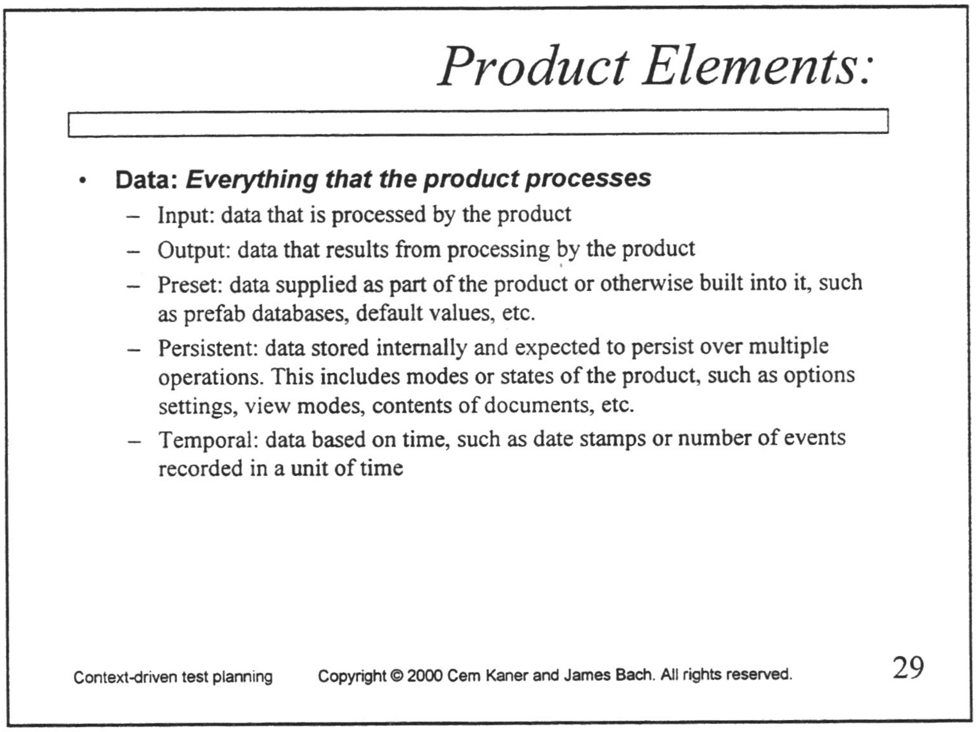
Product Elements:

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



Product Elements:

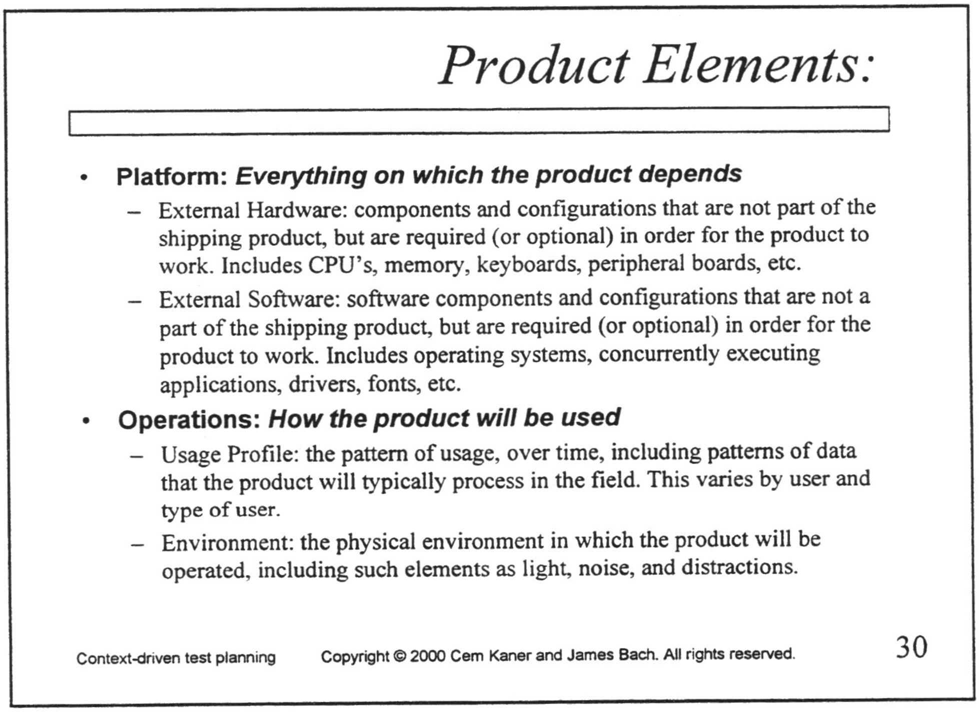
- **Functions:** ***Everything that the product does.***
  - User Interface: functions that mediate the exchange of data with the user
  - System Interface: functions that exchange data with something other than the user, such as with other programs, hard disk, network, printer, etc.
  - Application: functions that define or distinguish the product or fulfill core requirements
  - Error Handling: functions that detect and recover from errors, including error messages
  - Testability: functions provided to help test the product, such as diagnostics, log files, asserts, test menus, etc.
- **Temporal relationships:** ***How the program functions over time***
  - Sequential operation: state-to-state transitions
  - Data: changes in variables over time
  - System interactions: such as synchronization or ordering of events in distributed systems



Product Elements:

- **Data: *Everything that the product processes***
  - Input: data that is processed by the product
  - Output: data that results from processing by the product
  - Preset: data supplied as part of the product or otherwise built into it, such as prefab databases, default values, etc.
  - Persistent: data stored internally and expected to persist over multiple operations. This includes modes or states of the product, such as options settings, view modes, contents of documents, etc.
  - Temporal: data based on time, such as date stamps or number of events recorded in a unit of time

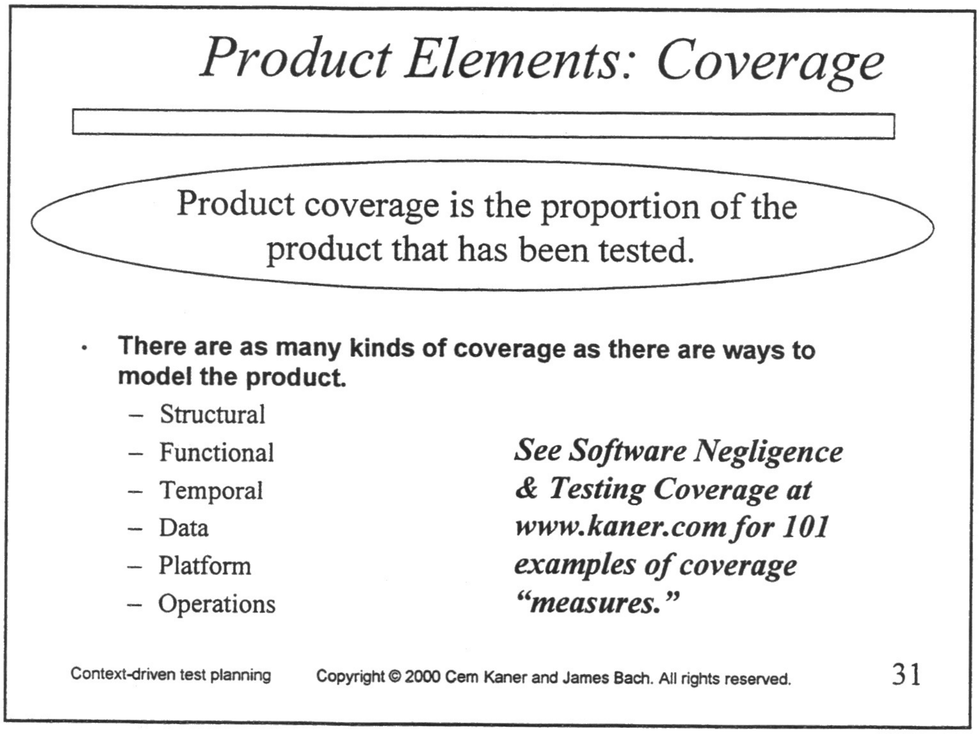
Context-driven test planning 29



Product Elements:

- **Platform: *Everything on which the product depends***
  - External Hardware: components and configurations that are not part of the shipping product, but are required (or optional) in order for the product to work. Includes CPU's, memory, keyboards, peripheral boards, etc.
  - External Software: software components and configurations that are not a part of the shipping product, but are required (or optional) in order for the product to work. Includes operating systems, concurrently executing applications, drivers, fonts, etc.
- **Operations: *How the product will be used***
  - Usage Profile: the pattern of usage, over time, including patterns of data that the product will typically process in the field. This varies by user and type of user.
  - Environment: the physical environment in which the product will be operated, including such elements as light, noise, and distractions.

Context-driven test planning 30



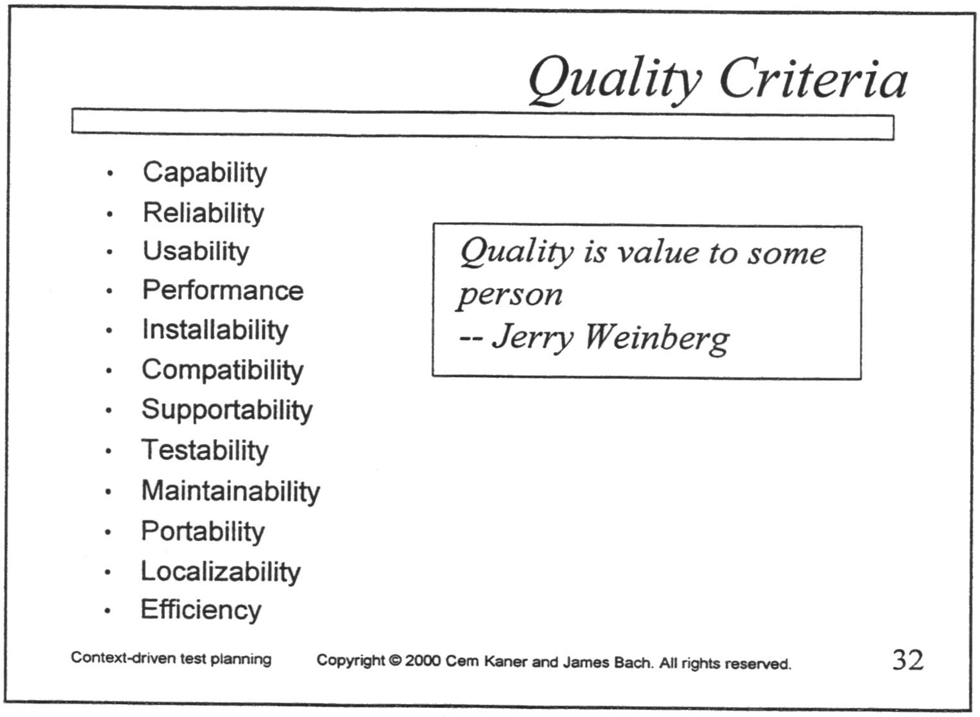
Product Elements: Coverage

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

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

See Software Negligence & Testing Coverage at www.kaner.com for 101 examples of coverage "measures."

Context-driven test planning 31

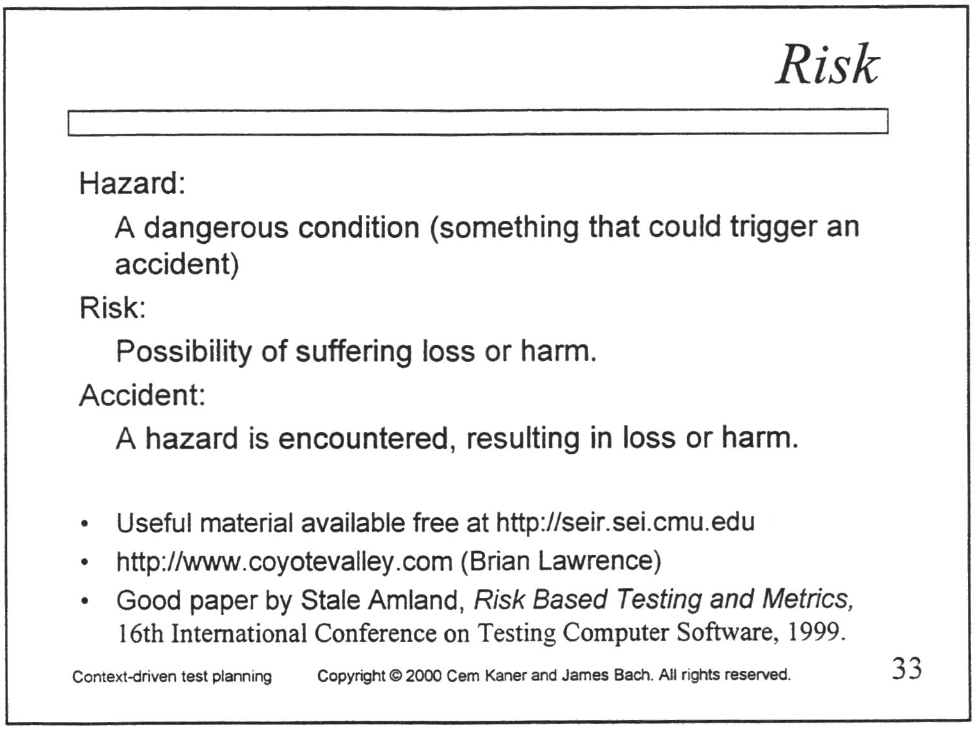


Quality Criteria

- Capability
- Reliability
- Usability
- Performance
- Installability
- Compatibility
- Supportability
- Testability
- Maintainability
- Portability
- Localizability
- Efficiency

Quality is value to some person
-- Jerry Weinberg

Context-driven test planning 32



Risk

Hazard:

A dangerous condition (something that could trigger an accident)

Risk:

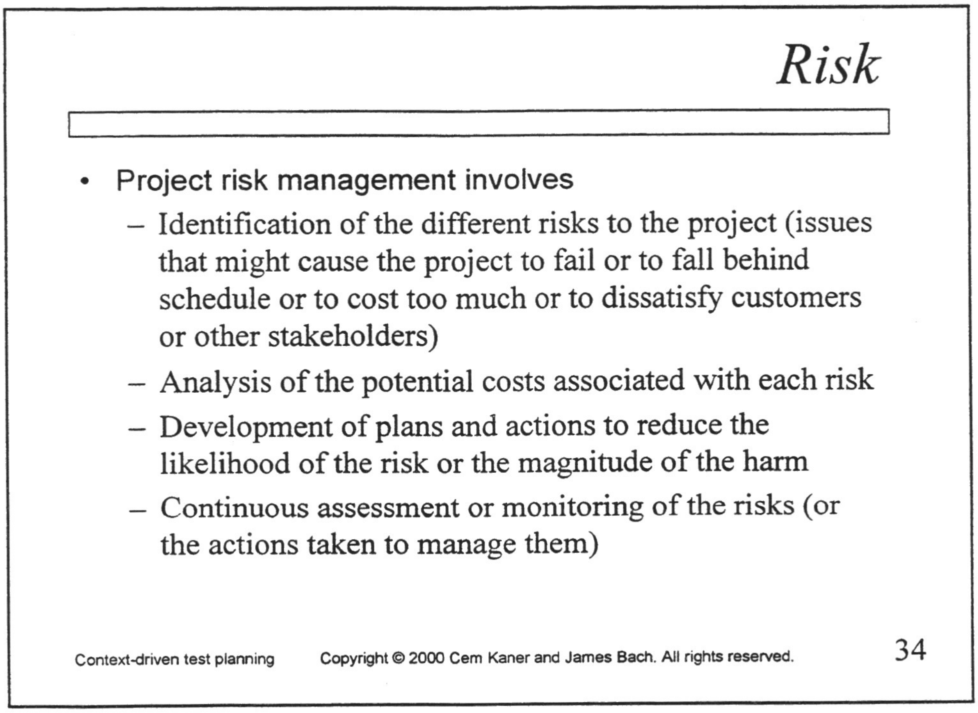
Possibility of suffering loss or harm.

Accident:

A hazard is encountered, resulting in loss or harm.

- Useful material available free at http://seir.sei.cmu.edu
- http://www.coyotevalley.com (Brian Lawrence)
- Good paper by Stale Amland, *Risk Based Testing and Metrics,* 16th International Conference on Testing Computer Software, 1999.

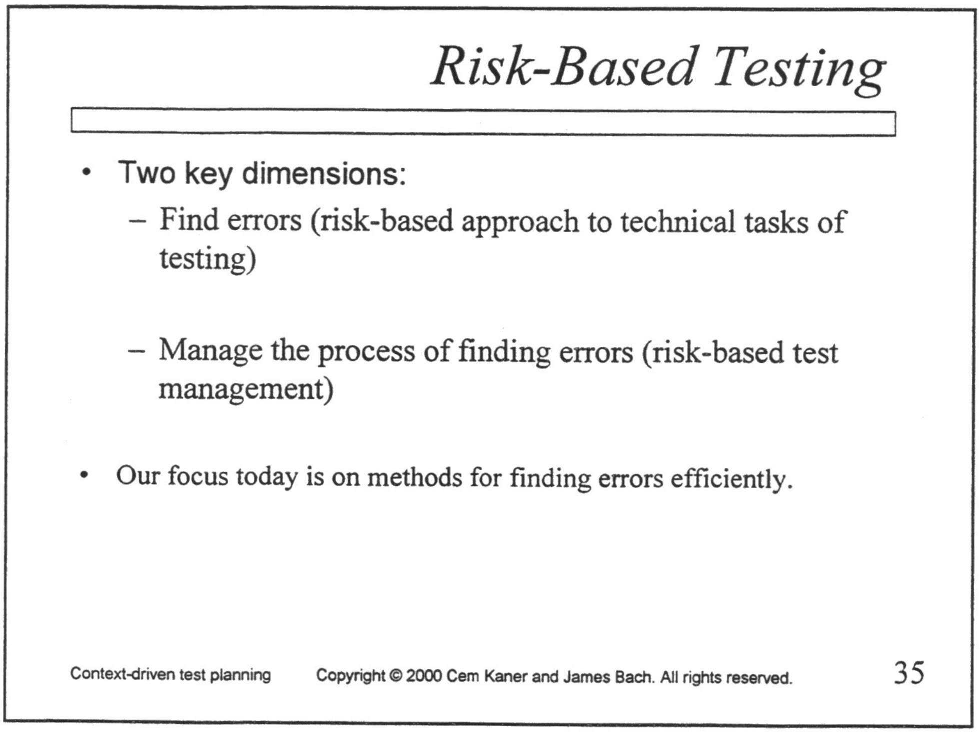
Context-driven test planning 33



Risk

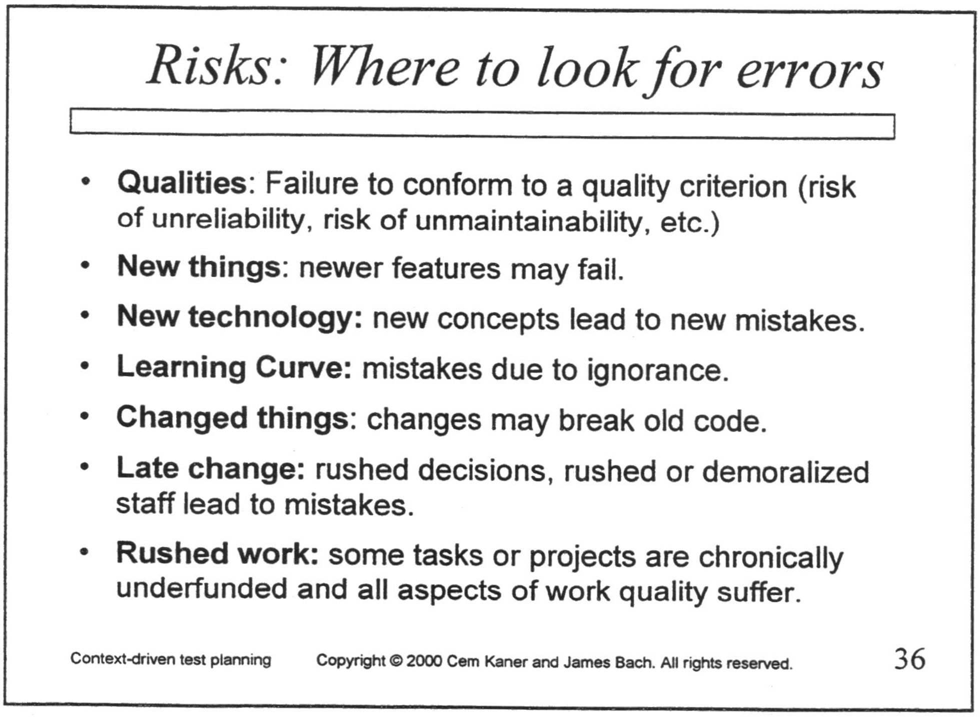
- Project risk management involves
  - Identification of the different risks to the project (issues that might cause the project to fail or to fall behind schedule or to cost too much or to dissatisfy customers or other stakeholders)
  - Analysis of the potential costs associated with each risk
  - Development of plans and actions to reduce the likelihood of the risk or the magnitude of the harm
  - Continuous assessment or monitoring of the risks (or the actions taken to manage them)

Context-driven test planning 34



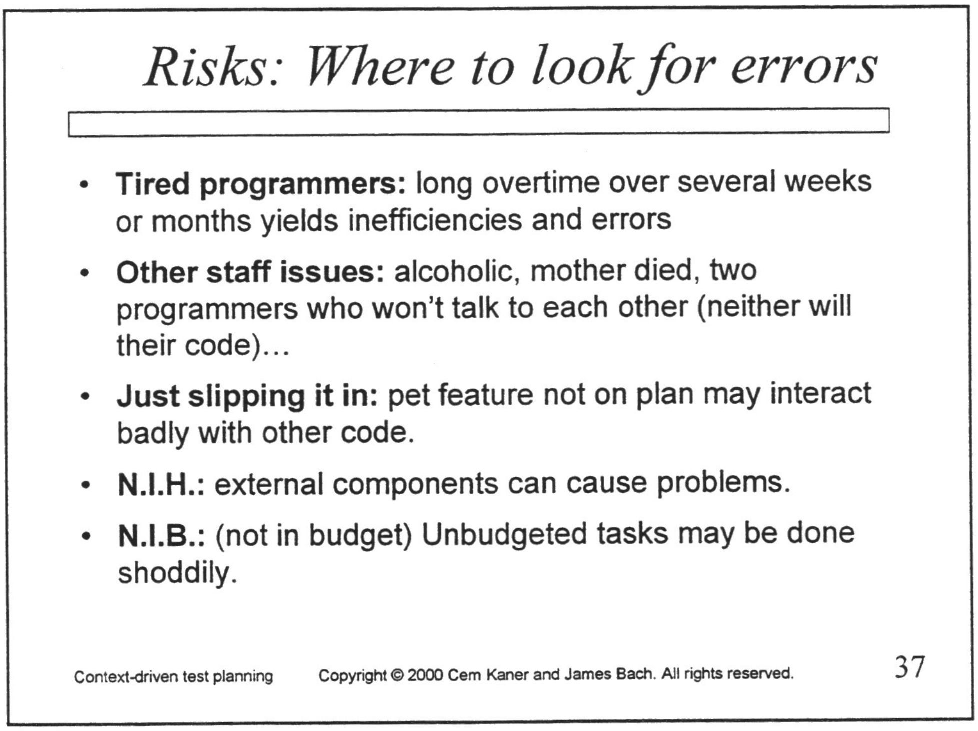
Risk-Based Testing

- Two key dimensions:
  - Find errors (risk-based approach to technical tasks of testing)
  - Manage the process of finding errors (risk-based test management)
- Our focus today is on methods for finding errors efficiently.



Risks: Where to look for errors

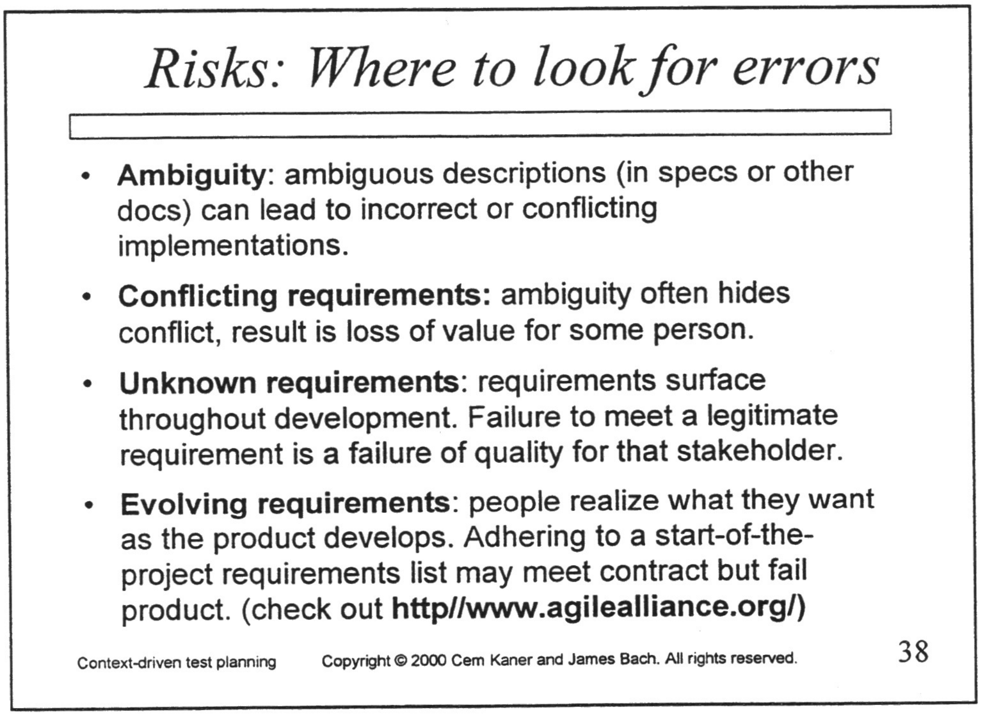
- **Qualities**: Failure to conform to a quality criterion (risk of unreliability, risk of unmaintainability, etc.)
- **New things**: newer features may fail.
- **New technology:** new concepts lead to new mistakes.
- **Learning Curve:** mistakes due to ignorance.
- **Changed things**: changes may break old code.
- **Late change:** rushed decisions, rushed or demoralized staff lead to mistakes.
- **Rushed work:** some tasks or projects are chronically underfunded and all aspects of work quality suffer.



Risks: Where to look for errors

- **Tired programmers:** long overtime over several weeks or months yields inefficiencies and errors
- **Other staff issues:** alcoholic, mother died, two programmers who won't talk to each other (neither will their code)…
- **Just slipping it in:** pet feature not on plan may interact badly with other code.
- **N.I.H.:** external components can cause problems.
- **N.I.B.:** (not in budget) Unbudgeted tasks may be done shoddily.

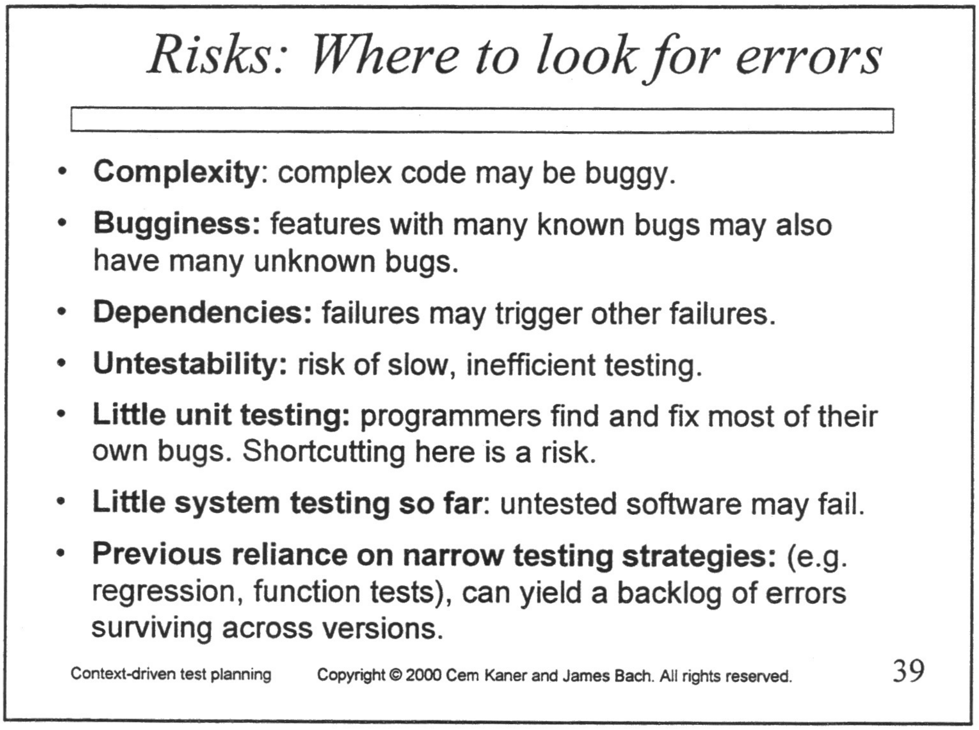
Context-driven test planning 37



Risks: Where to look for errors

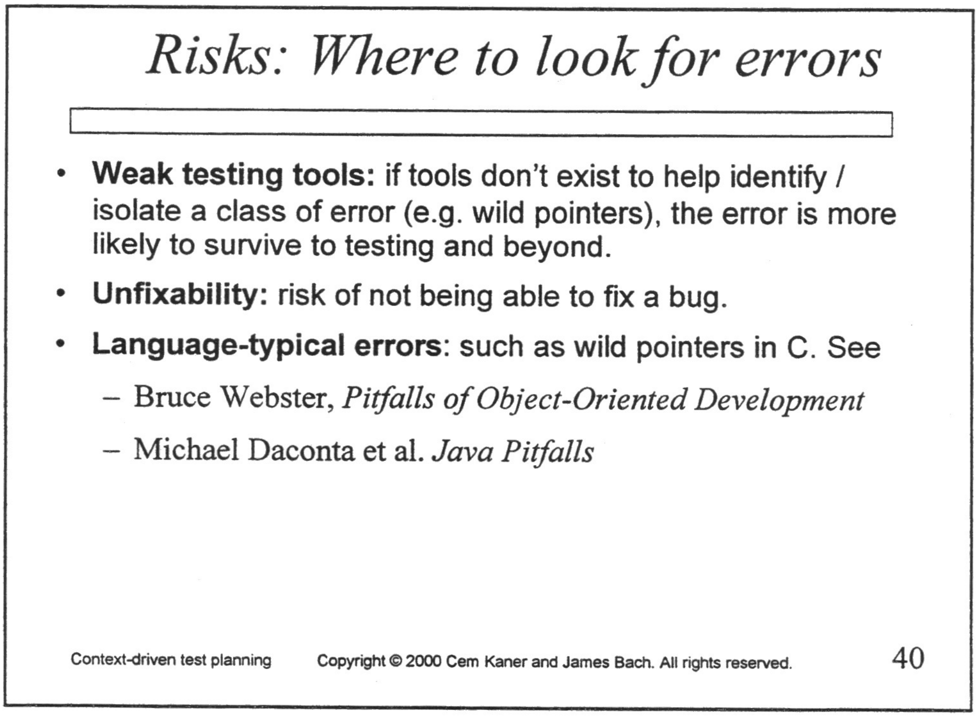
- **Ambiguity**: ambiguous descriptions (in specs or other docs) can lead to incorrect or conflicting implementations.
- **Conflicting requirements:** ambiguity often hides conflict, result is loss of value for some person.
- **Unknown requirements**: requirements surface throughout development. Failure to meet a legitimate requirement is a failure of quality for that stakeholder.
- **Evolving requirements**: people realize what they want as the product develops. Adhering to a start-of-the-project requirements list may meet contract but fail product. (check out **http//www.agilealliance.org/)**

Context-driven test planning 38



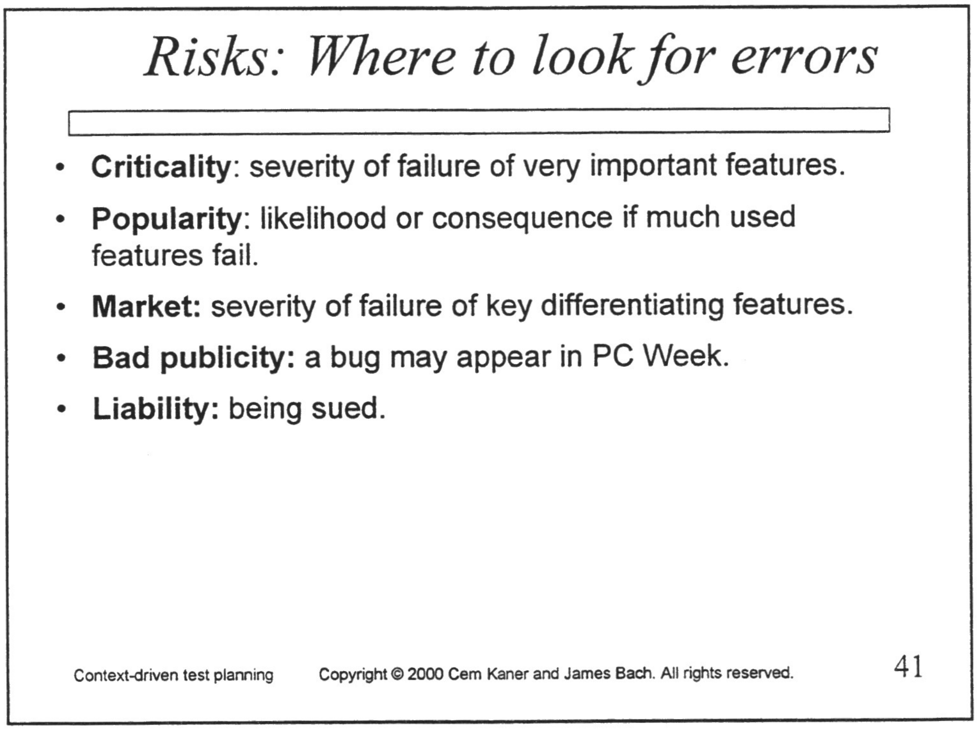
Risks: Where to look for errors

- **Complexity**: complex code may be buggy.
- **Bugginess:** features with many known bugs may also have many unknown bugs.
- **Dependencies:** failures may trigger other failures.
- **Untestability:** risk of slow, inefficient testing.
- **Little unit testing:** programmers find and fix most of their own bugs. Shortcutting here is a risk.
- **Little system testing so far**: untested software may fail.
- **Previous reliance on narrow testing strategies:** (e.g. regression, function tests), can yield a backlog of errors surviving across versions.



Risks: Where to look for errors

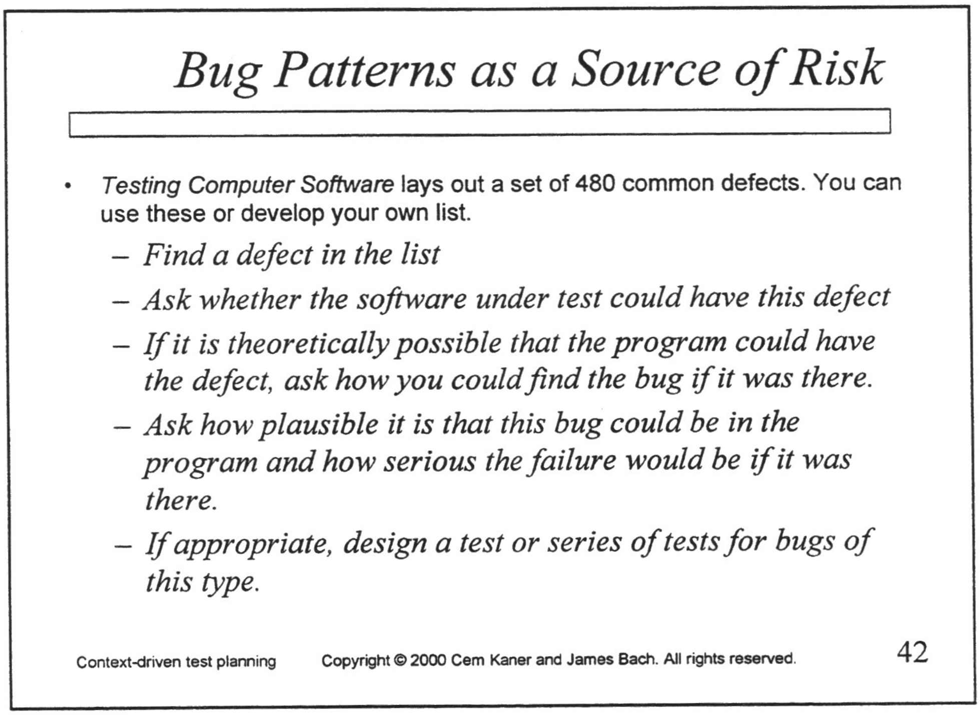
- **Weak testing tools:** if tools don't exist to help identify / isolate a class of error (e.g. wild pointers), the error is more likely to survive to testing and beyond.
- **Unfixability:** risk of not being able to fix a bug.
- **Language-typical errors**: such as wild pointers in C. See
  - Bruce Webster, *Pitfalls of Object-Oriented Development*
  - Michael Daconta et al. *Java Pitfalls*



Risks: Where to look for errors

- **Criticality**: severity of failure of very important features.
- **Popularity**: likelihood or consequence if much used features fail.
- **Market:** severity of failure of key differentiating features.
- **Bad publicity:** a bug may appear in PC Week.
- **Liability:** being sued.

Context-driven test planning 41



Bug Patterns as a Source of Risk

- *Testing Computer Software* lays out a set of 480 common defects. You can use these or develop your own list.
  - *Find a defect in the list*
  - *Ask whether the software under test could have this defect*
  - *If it is theoretically possible that the program could have the defect, ask how you could find the bug if it was there.*
  - *Ask how plausible it is that this bug could be in the program and how serious the failure would be if it was there.*
  - *If appropriate, design a test or series of tests for bugs of this type.*

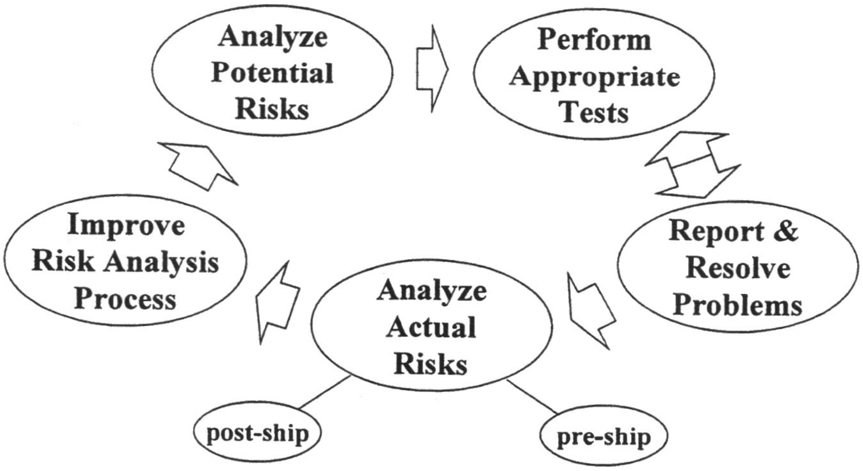
Context-driven test planning 42

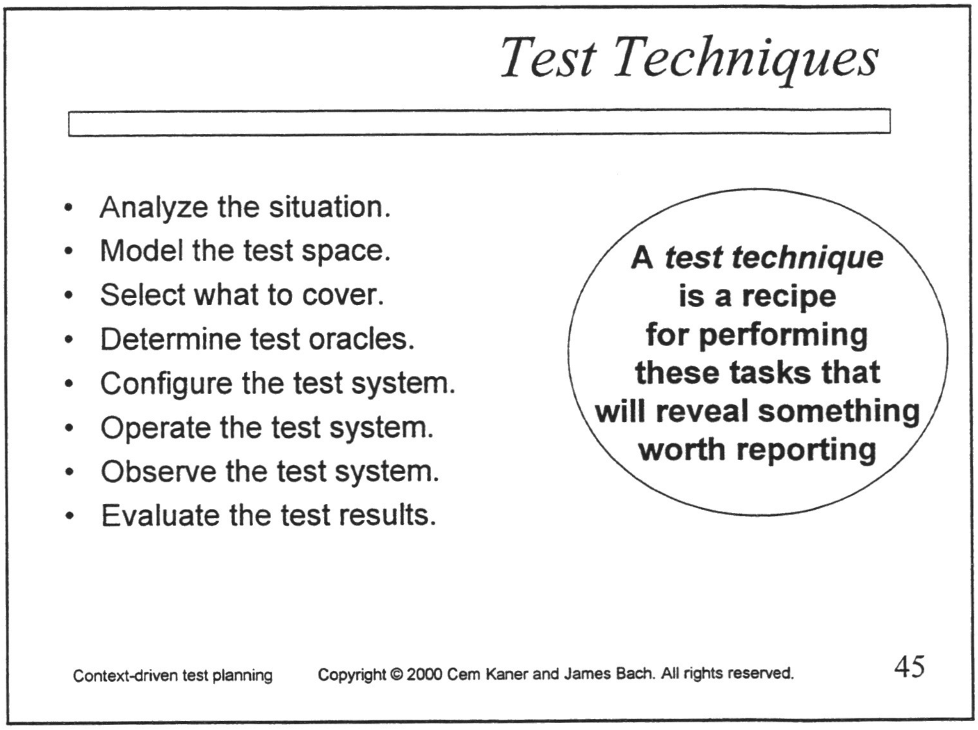
Build Your Own Model of Bug Patterns

Too many people start and end with the TCS bug list. It is outdated. It was outdated the day it was published. And it doesn't cover the issues in *your* system. Building a bug list is an ongoing process that constantly pays for itself. Here's an example from Hung Nguyen:

- This problem came up in a client/server system. The system sends the client a list of names, to allow verification that a name the client enters is not new.
- Client 1 and 2 both want to enter a name and client 1 and 2 both use the same new name. Both instances of the name are new relative to their local compare list and therefore, they are accepted, and we now have two instances of the same name.
- As we see these, we develop a library of issues. The discovery method is exploratory, requires sophistication with the underlying technology.
- Capture winning themes for testing in charts or in scripts-on-their-way to being automated.

Risk-Driven Testing Cycle

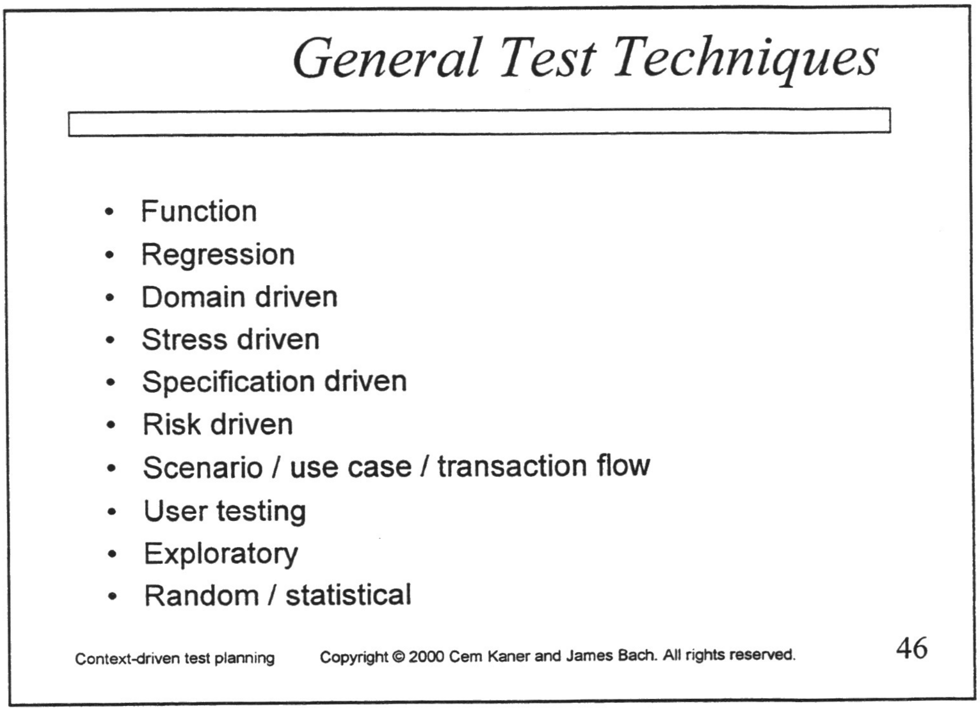

Analyze Potential Risks
Perform Appropriate Tests
Report & Resolve Problems
Analyze Actual Risks
pre-ship
post-ship
Improve Risk Analysis Process



Test Techniques

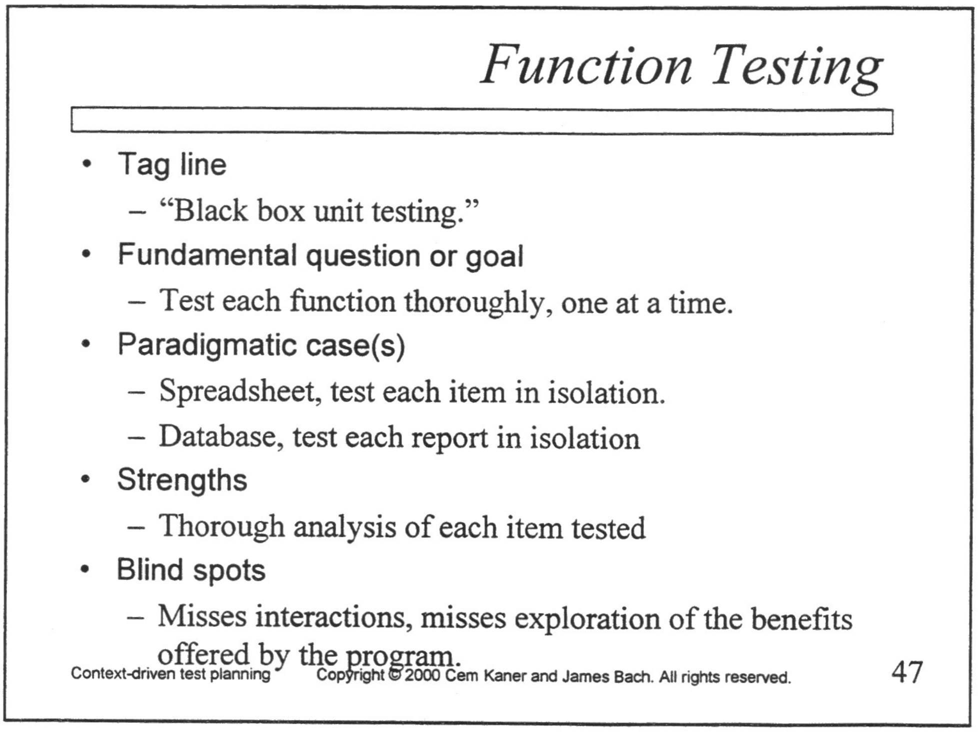
- Analyze the situation.
- Model the test space.
- Select what to cover.
- Determine test oracles.
- Configure the test system.
- Operate the test system.
- Observe the test system.
- Evaluate the test results.

A *test technique* is a recipe for performing these tasks that will reveal something worth reporting



General Test Techniques

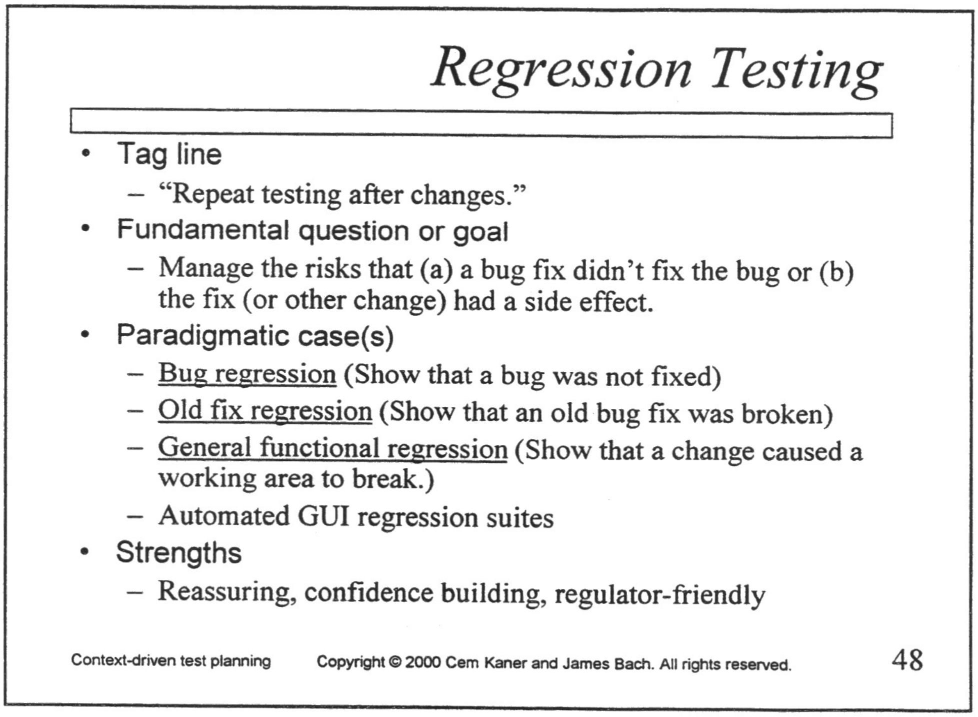
- Function
- Regression
- Domain driven
- Stress driven
- Specification driven
- Risk driven
- Scenario / use case / transaction flow
- User testing
- Exploratory
- Random / statistical



Function Testing

- Tag line
  - "Black box unit testing."
- Fundamental question or goal
  - Test each function thoroughly, one at a time.
- Paradigmatic case(s)
  - Spreadsheet, test each item in isolation.
  - Database, test each report in isolation
- Strengths
  - Thorough analysis of each item tested
- Blind spots
  - Misses interactions, misses exploration of the benefits offered by the program.

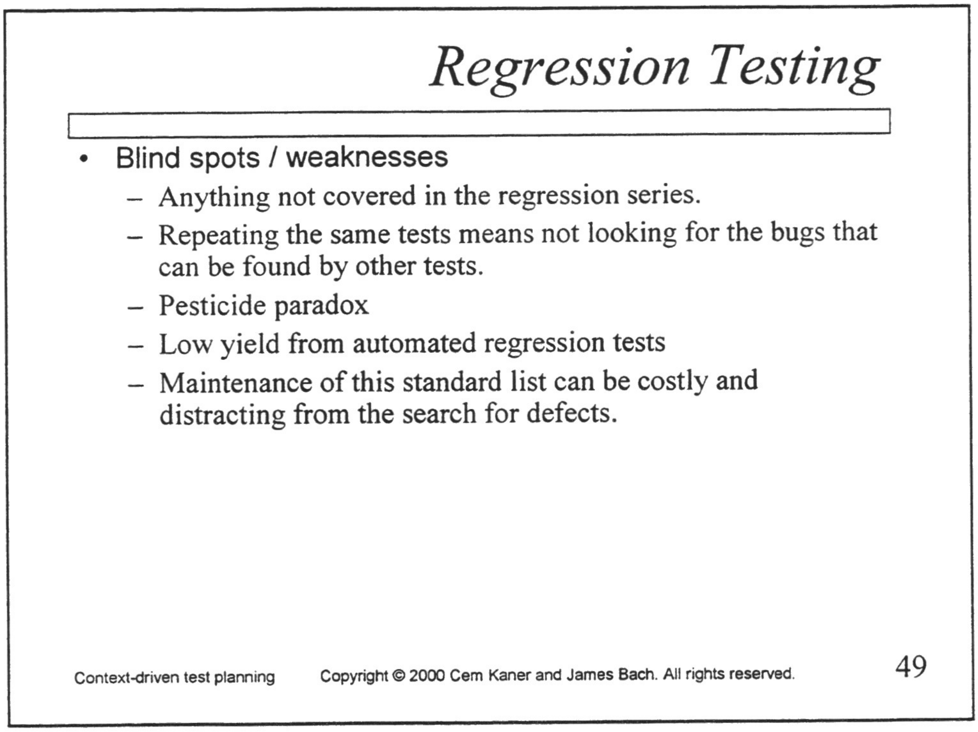
Context-driven test planning 47



Regression Testing

- Tag line
  - "Repeat testing after changes."
- Fundamental question or goal
  - Manage the risks that (a) a bug fix didn't fix the bug or (b) the fix (or other change) had a side effect.
- Paradigmatic case(s)
  - Bug regression (Show that a bug was not fixed)
  - Old fix regression (Show that an old bug fix was broken)
  - General functional regression (Show that a change caused a working area to break.)
  - Automated GUI regression suites
- Strengths
  - Reassuring, confidence building, regulator-friendly

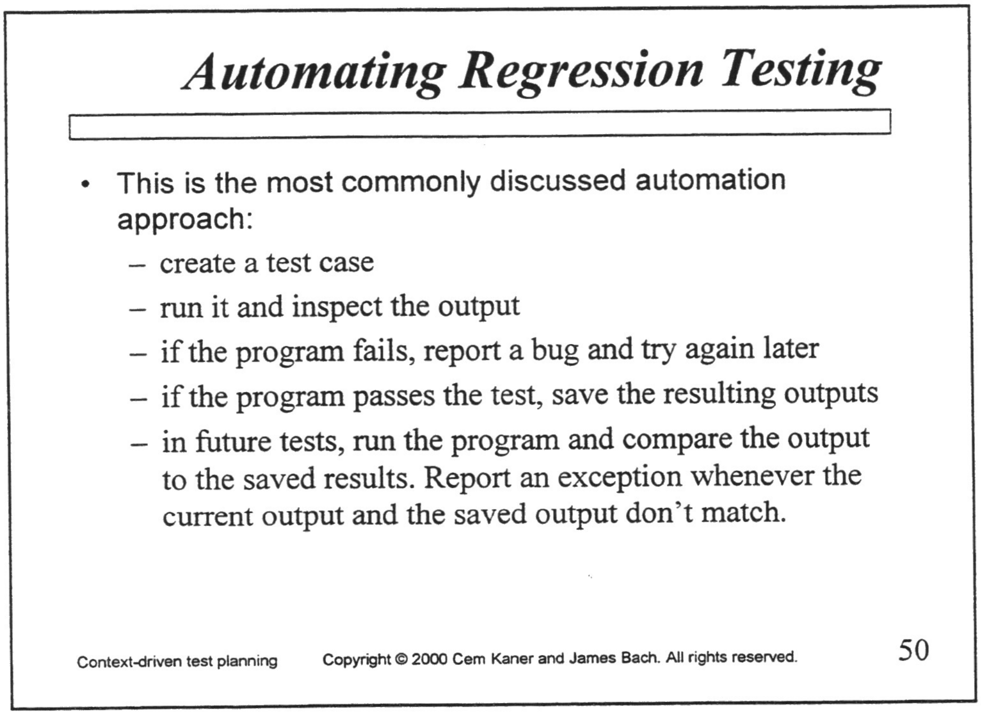
Context-driven test planning 48



Regression Testing

- Blind spots / weaknesses
  - Anything not covered in the regression series.
  - Repeating the same tests means not looking for the bugs that can be found by other tests.
  - Pesticide paradox
  - Low yield from automated regression tests
  - Maintenance of this standard list can be costly and distracting from the search for defects.

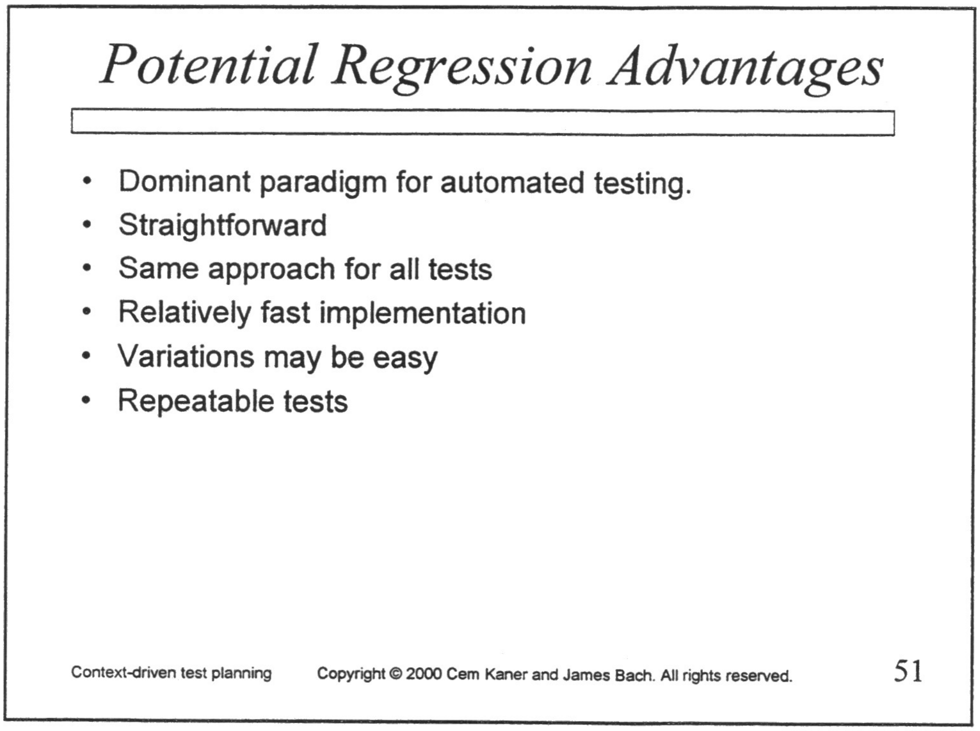
Context-driven test planning 49



Automating Regression Testing

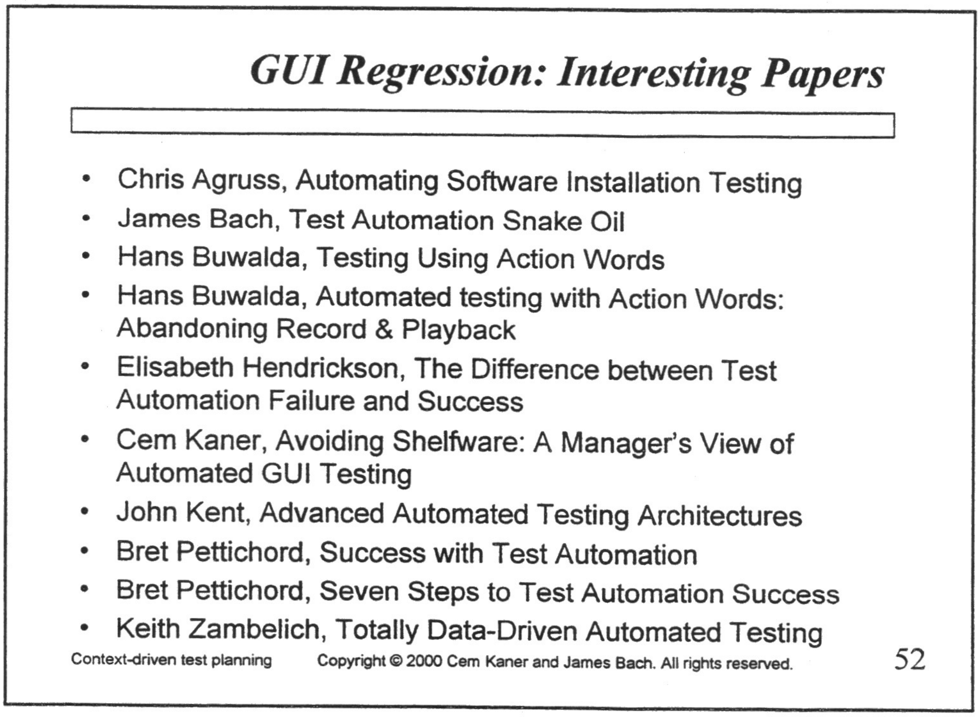
- This is the most commonly discussed automation approach:
  - create a test case
  - run it and inspect the output
  - if the program fails, report a bug and try again later
  - if the program passes the test, save the resulting outputs
  - in future tests, run the program and compare the output to the saved results. Report an exception whenever the current output and the saved output don't match.

Context-driven test planning 50



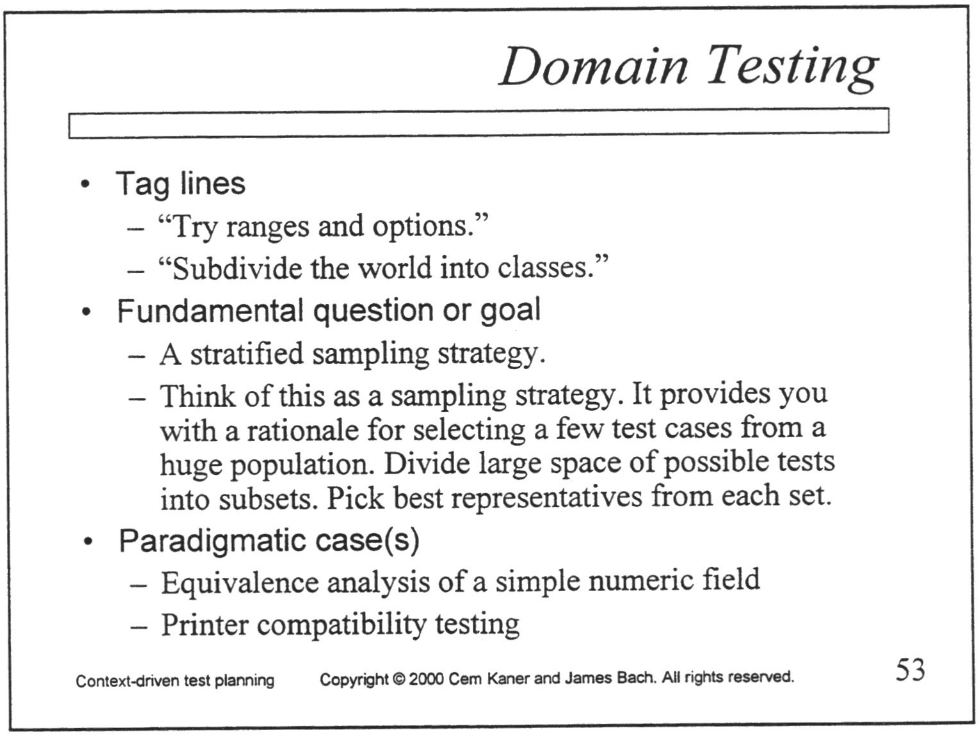
Potential Regression Advantages

- Dominant paradigm for automated testing.
- Straightforward
- Same approach for all tests
- Relatively fast implementation
- Variations may be easy
- Repeatable tests



GUI Regression: Interesting Papers

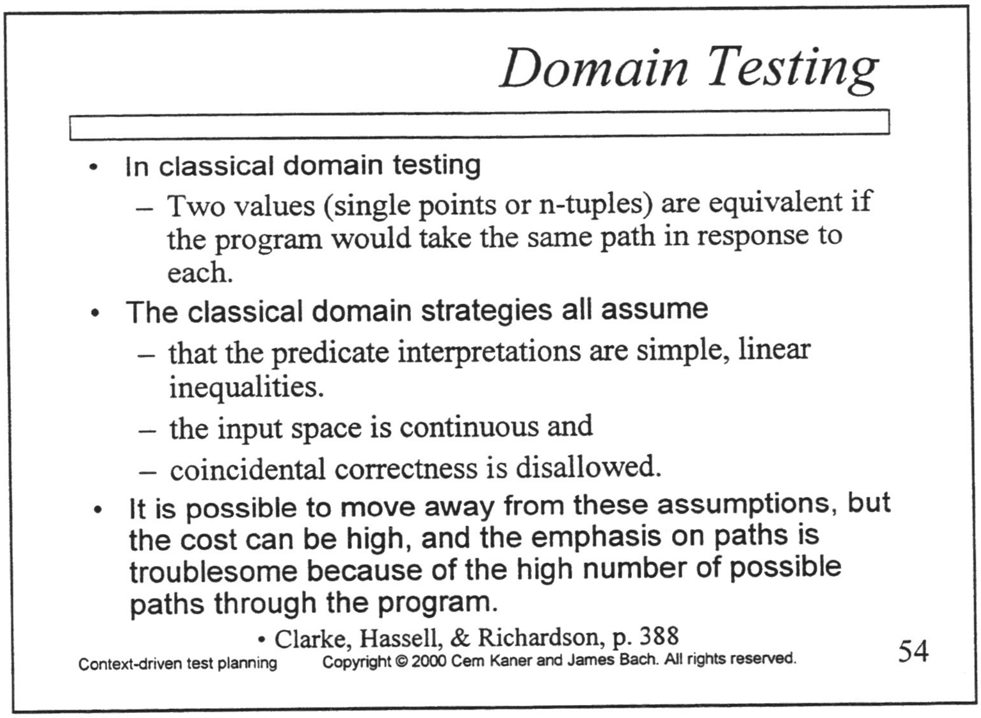
- Chris Agruss, Automating Software Installation Testing
- James Bach, Test Automation Snake Oil
- Hans Buwalda, Testing Using Action Words
- Hans Buwalda, Automated testing with Action Words: Abandoning Record & Playback
- Elisabeth Hendrickson, The Difference between Test Automation Failure and Success
- Cem Kaner, Avoiding Shelfware: A Manager's View of Automated GUI Testing
- John Kent, Advanced Automated Testing Architectures
- Bret Pettichord, Success with Test Automation
- Bret Pettichord, Seven Steps to Test Automation Success
- Keith Zambelich, Totally Data-Driven Automated Testing



Domain Testing

- Tag lines
  - “Try ranges and options.”
  - “Subdivide the world into classes.”
- Fundamental question or goal
  - A stratified sampling strategy.
  - Think of this as a sampling strategy. It provides you with a rationale for selecting a few test cases from a huge population. Divide large space of possible tests into subsets. Pick best representatives from each set.
- Paradigmatic case(s)
  - Equivalence analysis of a simple numeric field
  - Printer compatibility testing

Context-driven test planning Copyright © 2000 Cem Kaner and James Bach. All rights reserved. 53



Domain Testing

- In classical domain testing
  - Two values (single points or n-tuples) are equivalent if the program would take the same path in response to each.
- The classical domain strategies all assume
  - that the predicate interpretations are simple, linear inequalities.
  - the input space is continuous and
  - coincidental correctness is disallowed.
- It is possible to move away from these assumptions, but the cost can be high, and the emphasis on paths is troublesome because of the high number of possible paths through the program.
  - Clarke, Hassell, & Richardson, p. 388

Context-driven test planning Copyright © 2000 Cem Kaner and James Bach. All rights reserved. 54

Equivalence and Risk

Our working definition of equivalence:

> *Two test cases are equivalent if you expect the same result from each.*

This is fundamentally subjective. It depends on what you expect. And what you expect depends on what errors you can anticipate:

> *Two test cases can only be equivalent by reference to a specifiable risk.*

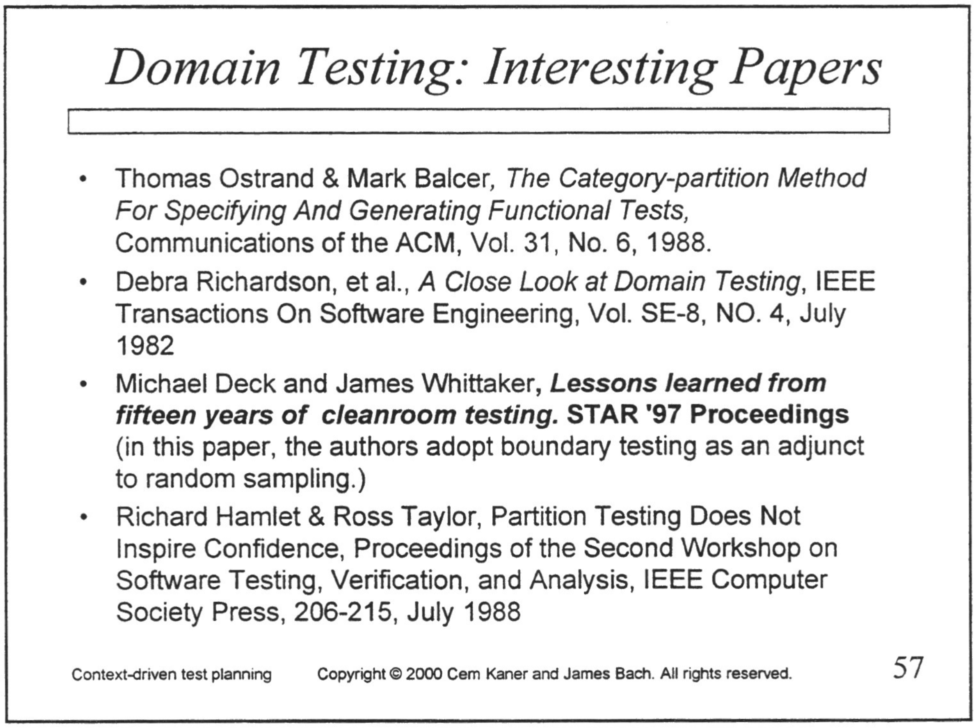
Two different testers will have different theories about how programs can fail, and therefore they will come up with different classes.

A boundary case in this system is a "best representative."

> *A best representative of an equivalence class is a test that is at least as likely to expose a fault as every other member of the class.*

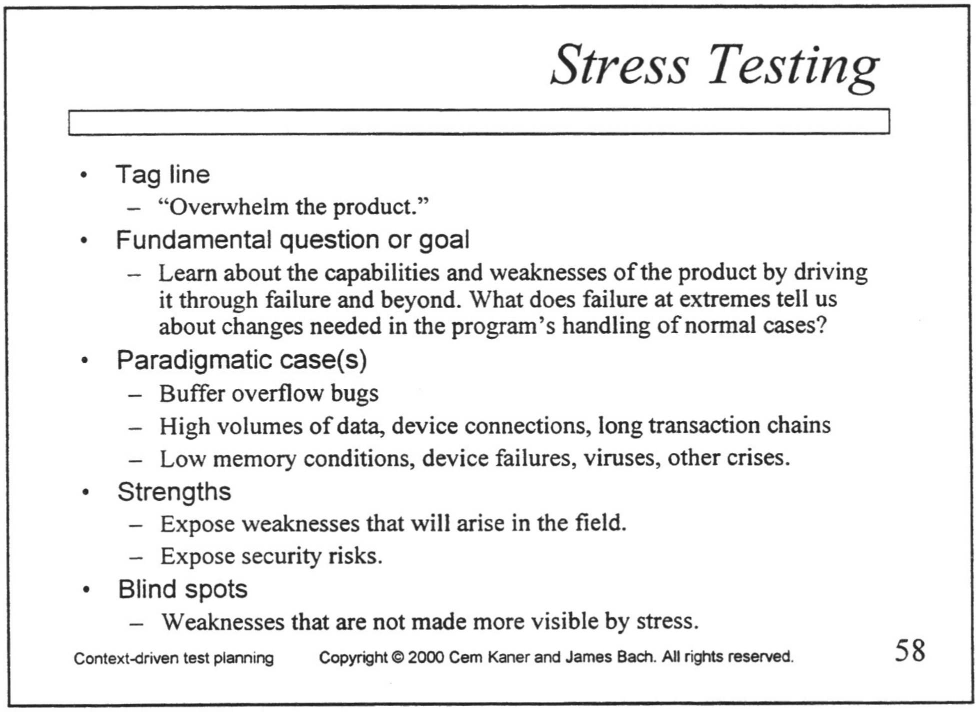
Domain Testing

- Strengths
  - Find highest probability errors with a relatively small set of tests.
  - Intuitively clear approach, generalizes well
- Blind spots
  - Errors that are not at boundaries or in obvious special cases.
  - Also, the actual domains are often unknowable.



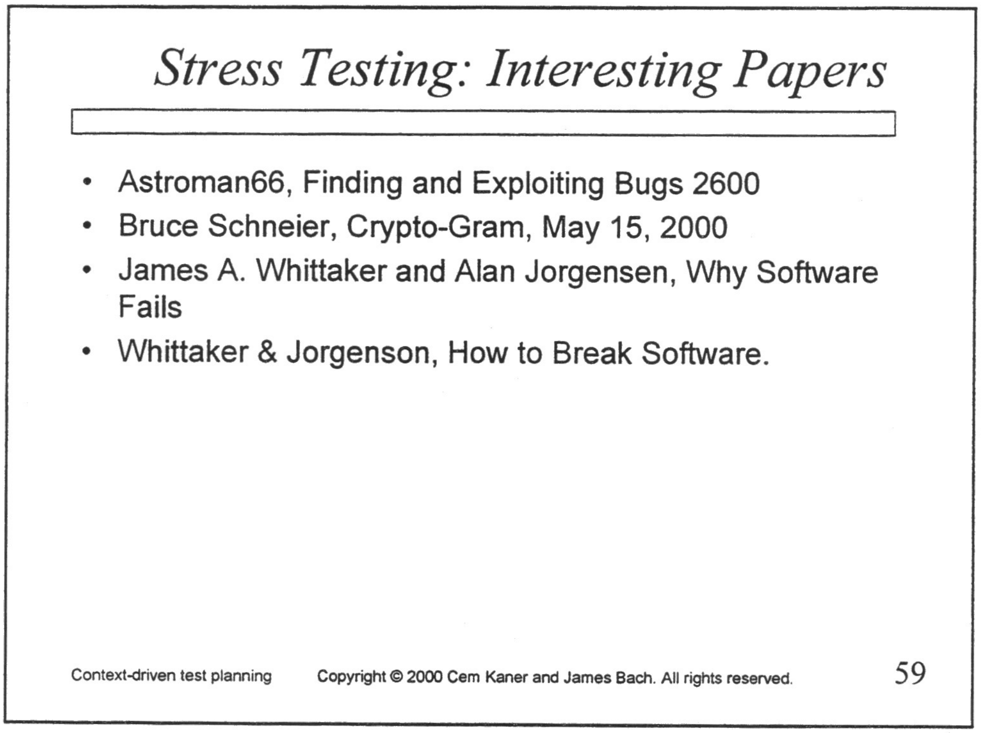
Domain Testing: Interesting Papers

- Thomas Ostrand & Mark Balcer, *The Category-partition Method For Specifying And Generating Functional Tests,* Communications of the ACM, Vol. 31, No. 6, 1988.
- Debra Richardson, et al., *A Close Look at Domain Testing*, IEEE Transactions On Software Engineering, Vol. SE-8, NO. 4, July 1982
- Michael Deck and James Whittaker, ***Lessons learned from fifteen years of cleanroom testing.*** **STAR '97 Proceedings** (in this paper, the authors adopt boundary testing as an adjunct to random sampling.)
- Richard Hamlet & Ross Taylor, Partition Testing Does Not Inspire Confidence, Proceedings of the Second Workshop on Software Testing, Verification, and Analysis, IEEE Computer Society Press, 206-215, July 1988



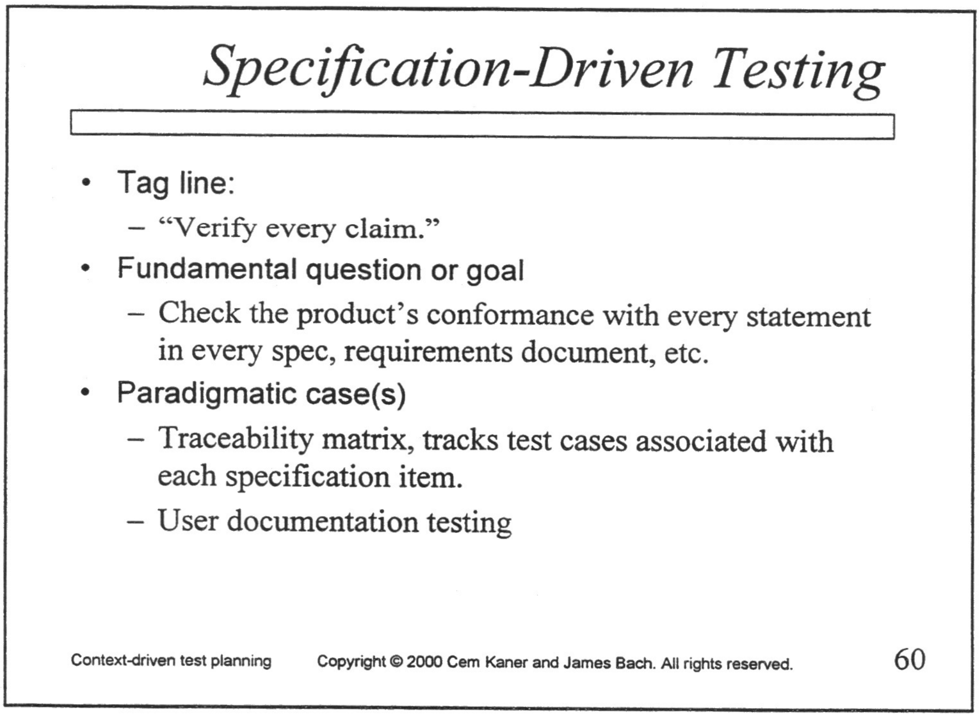
Stress Testing

- Tag line
  - "Overwhelm the product."
- Fundamental question or goal
  - Learn about the capabilities and weaknesses of the product by driving it through failure and beyond. What does failure at extremes tell us about changes needed in the program's handling of normal cases?
- Paradigmatic case(s)
  - Buffer overflow bugs
  - High volumes of data, device connections, long transaction chains
  - Low memory conditions, device failures, viruses, other crises.
- Strengths
  - Expose weaknesses that will arise in the field.
  - Expose security risks.
- Blind spots
  - Weaknesses that are not made more visible by stress.



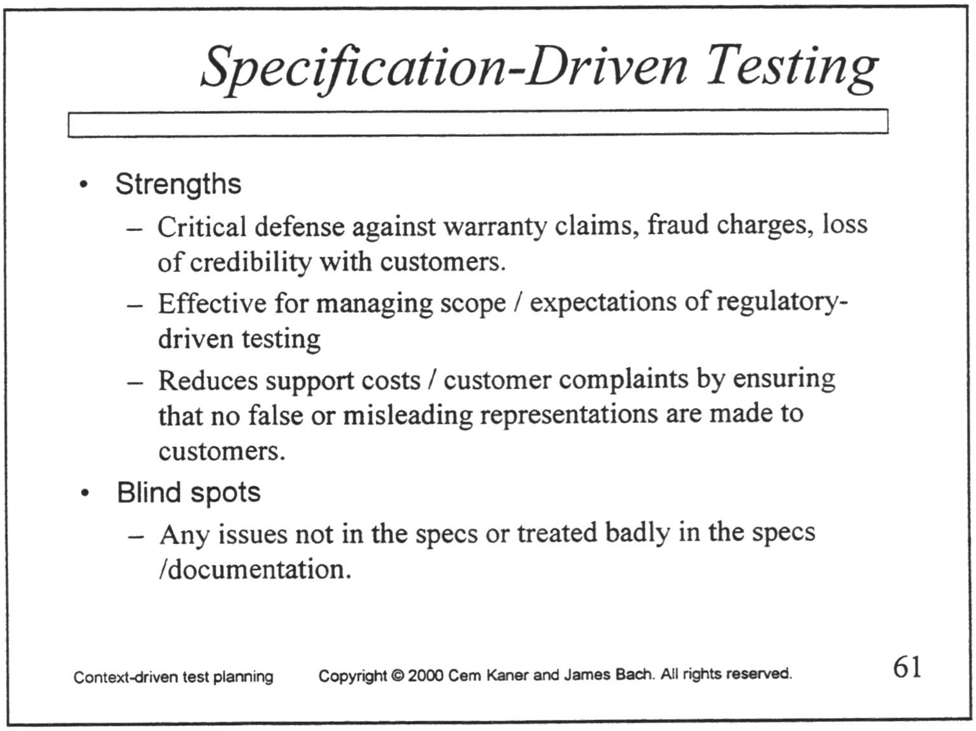
Stress Testing: Interesting Papers

- Astroman66, Finding and Exploiting Bugs 2600
- Bruce Schneier, Crypto-Gram, May 15, 2000
- James A. Whittaker and Alan Jorgensen, Why Software Fails
- Whittaker & Jorgenson, How to Break Software.



Specification-Driven Testing

- Tag line:
  - "Verify every claim."
- Fundamental question or goal
  - Check the product's conformance with every statement in every spec, requirements document, etc.
- Paradigmatic case(s)
  - Traceability matrix, tracks test cases associated with each specification item.
  - User documentation testing

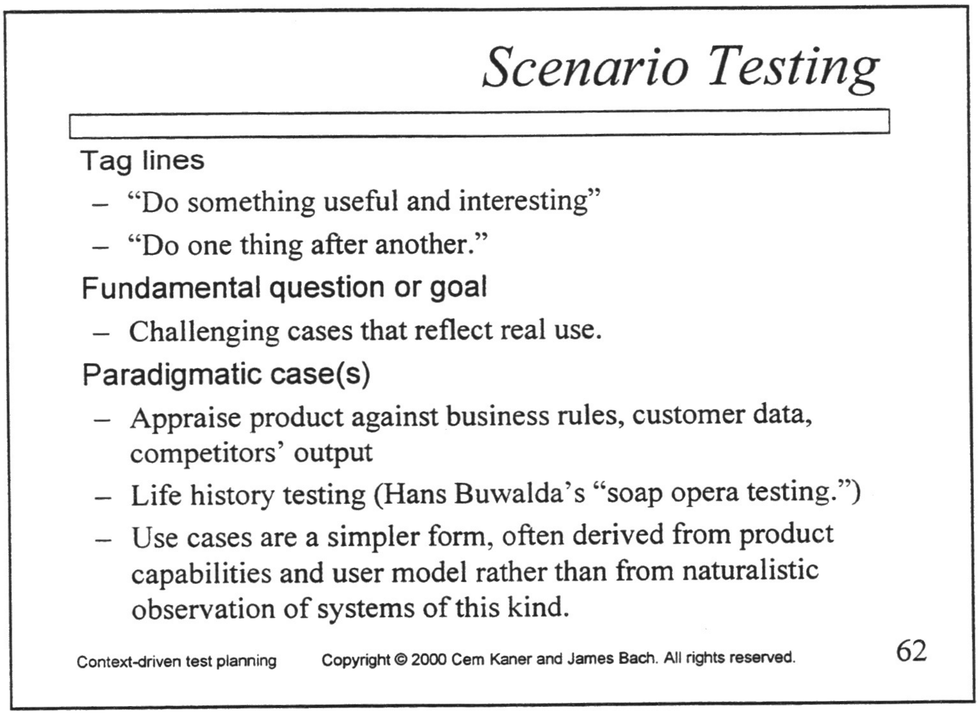


Specification-Driven Testing

- Strengths
  - Critical defense against warranty claims, fraud charges, loss of credibility with customers.
  - Effective for managing scope / expectations of regulatory-driven testing
  - Reduces support costs / customer complaints by ensuring that no false or misleading representations are made to customers.
- Blind spots
  - Any issues not in the specs or treated badly in the specs /documentation.

Context-driven test planning

61



Scenario Testing

Tag lines

- "Do something useful and interesting"
- "Do one thing after another."

Fundamental question or goal

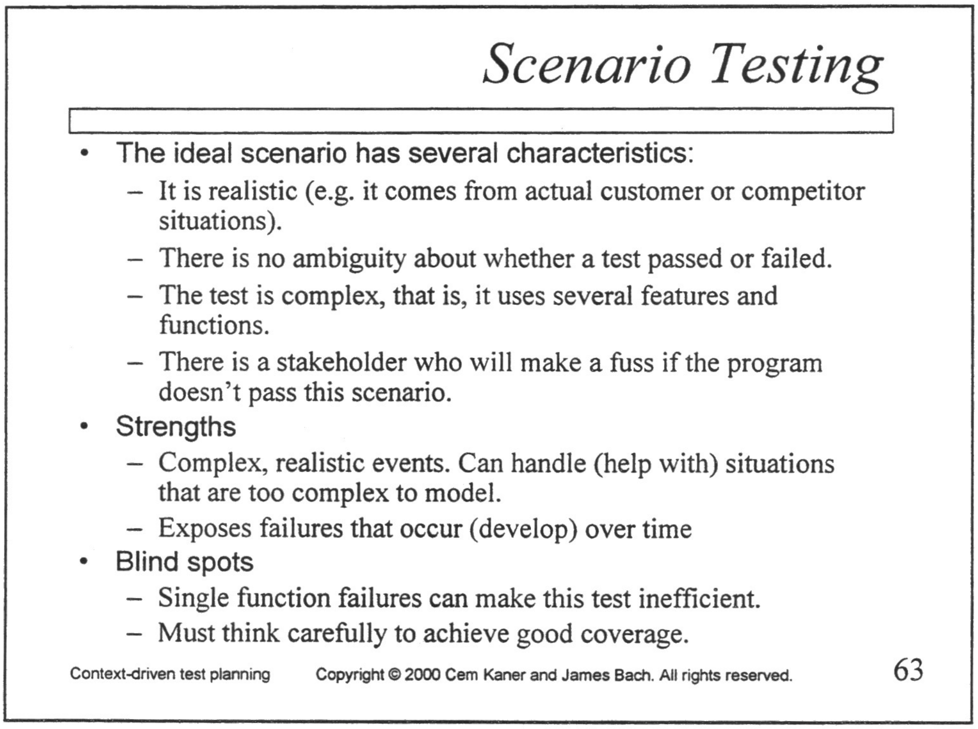
- Challenging cases that reflect real use.

Paradigmatic case(s)

- Appraise product against business rules, customer data, competitors' output
- Life history testing (Hans Buwalda's "soap opera testing.")
- Use cases are a simpler form, often derived from product capabilities and user model rather than from naturalistic observation of systems of this kind.

Context-driven test planning

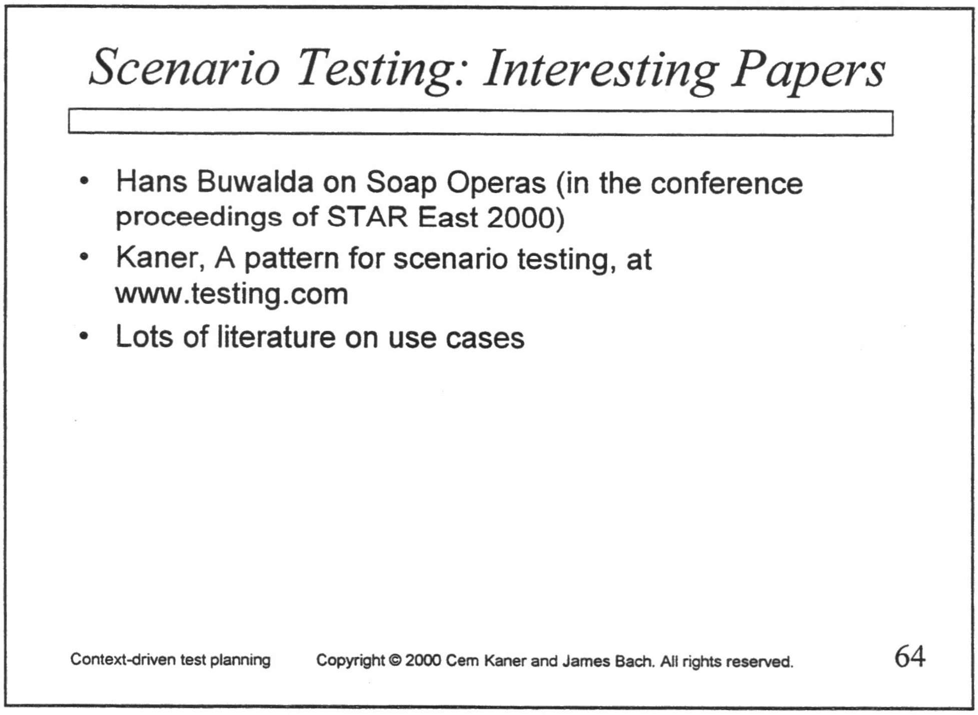
62



Scenario Testing

- The ideal scenario has several characteristics:
  - It is realistic (e.g. it comes from actual customer or competitor situations).
  - There is no ambiguity about whether a test passed or failed.
  - The test is complex, that is, it uses several features and functions.
  - There is a stakeholder who will make a fuss if the program doesn’t pass this scenario.
- Strengths
  - Complex, realistic events. Can handle (help with) situations that are too complex to model.
  - Exposes failures that occur (develop) over time
- Blind spots
  - Single function failures can make this test inefficient.
  - Must think carefully to achieve good coverage.

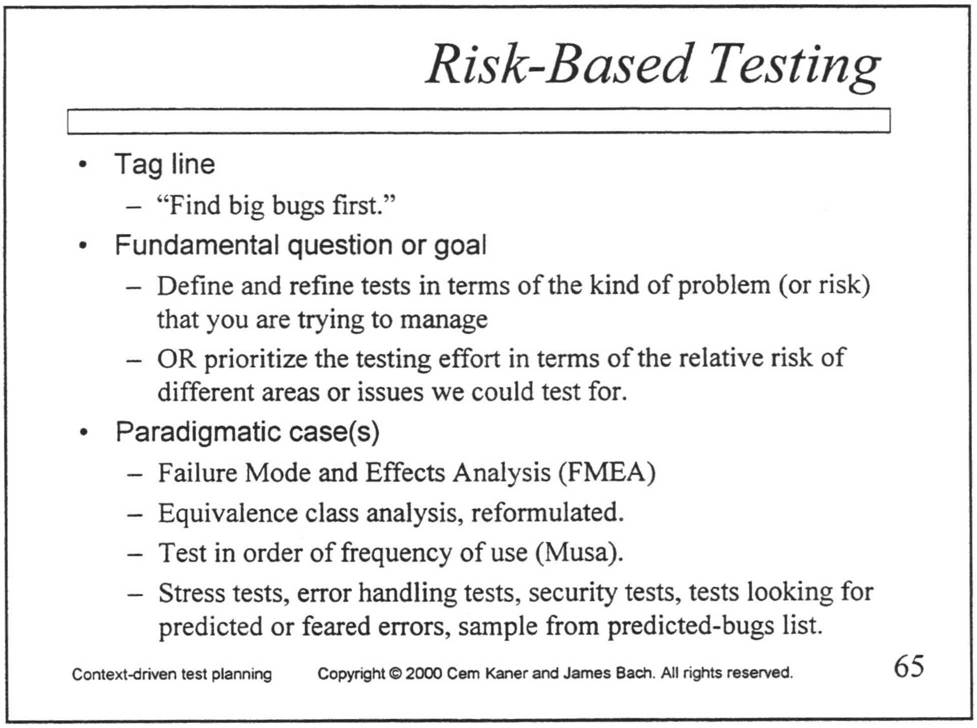
Context-driven test planning 63



Scenario Testing: Interesting Papers

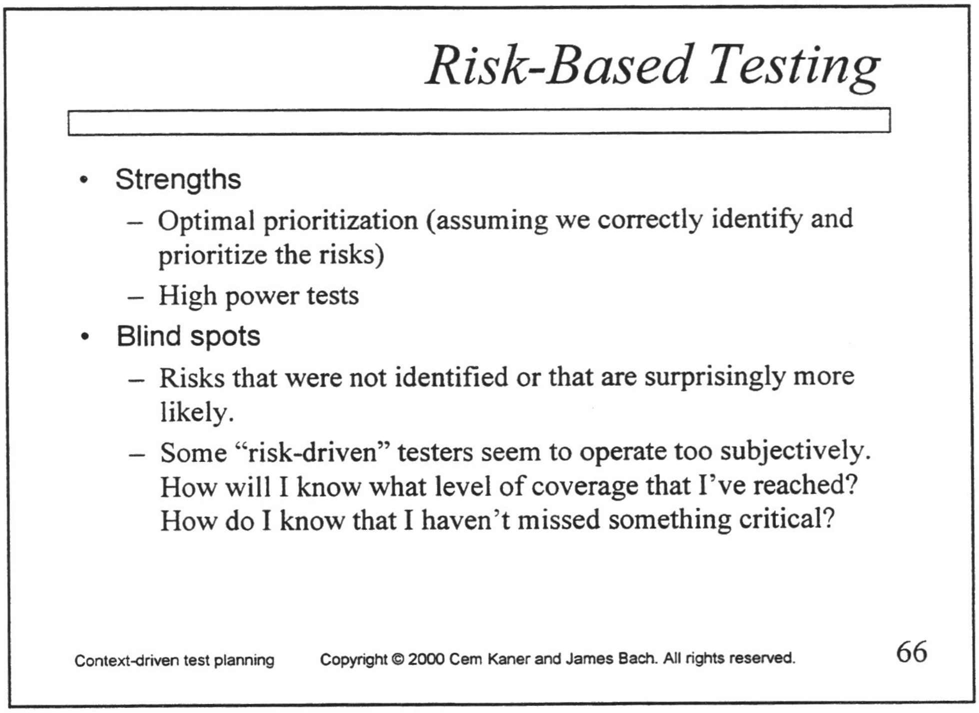
- Hans Buwalda on Soap Operas (in the conference proceedings of STAR East 2000)
- Kaner, A pattern for scenario testing, at www.testing.com
- Lots of literature on use cases

Context-driven test planning 64



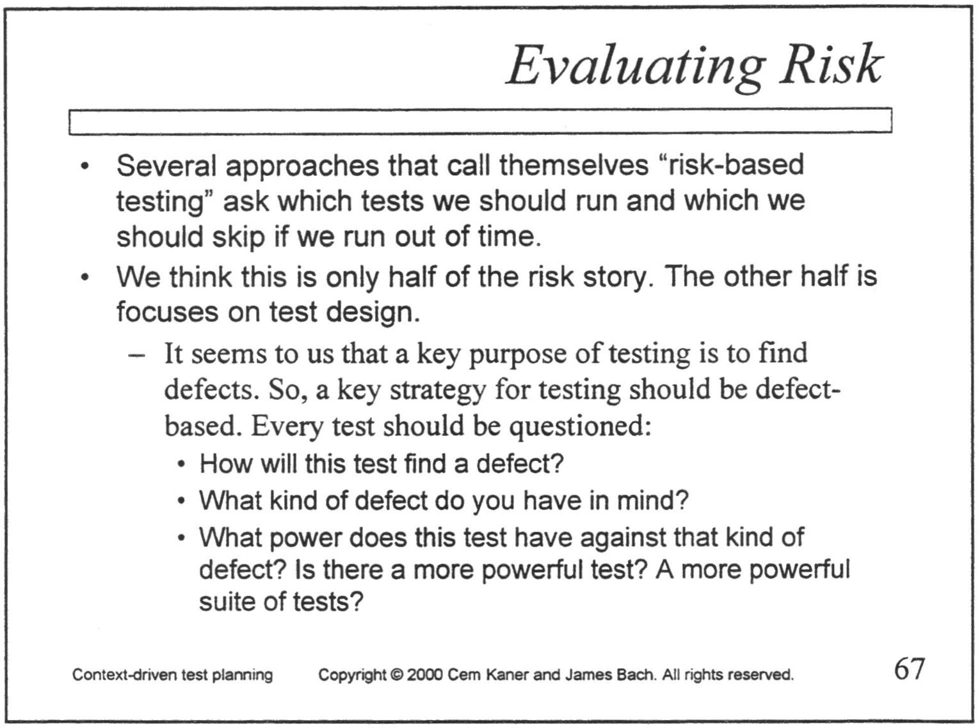
Risk-Based Testing

- Tag line
  - "Find big bugs first."
- Fundamental question or goal
  - Define and refine tests in terms of the kind of problem (or risk) that you are trying to manage
  - OR prioritize the testing effort in terms of the relative risk of different areas or issues we could test for.
- Paradigmatic case(s)
  - Failure Mode and Effects Analysis (FMEA)
  - Equivalence class analysis, reformulated.
  - Test in order of frequency of use (Musa).
  - Stress tests, error handling tests, security tests, tests looking for predicted or feared errors, sample from predicted-bugs list.



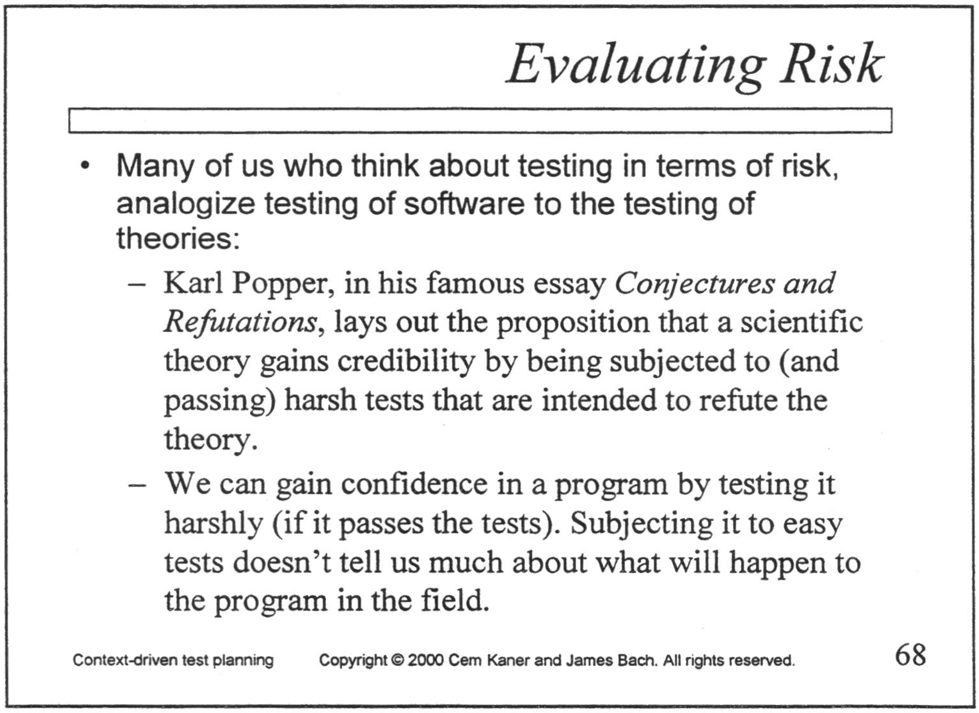
Risk-Based Testing

- Strengths
  - Optimal prioritization (assuming we correctly identify and prioritize the risks)
  - High power tests
- Blind spots
  - Risks that were not identified or that are surprisingly more likely.
  - Some "risk-driven" testers seem to operate too subjectively. How will I know what level of coverage that I've reached? How do I know that I haven't missed something critical?



Evaluating Risk

- Several approaches that call themselves "risk-based testing" ask which tests we should run and which we should skip if we run out of time.
- We think this is only half of the risk story. The other half is focuses on test design.
  - It seems to us that a key purpose of testing is to find defects. So, a key strategy for testing should be defect-based. Every test should be questioned:
    - How will this test find a defect?
    - What kind of defect do you have in mind?
    - What power does this test have against that kind of defect? Is there a more powerful test? A more powerful suite of tests?



Evaluating Risk

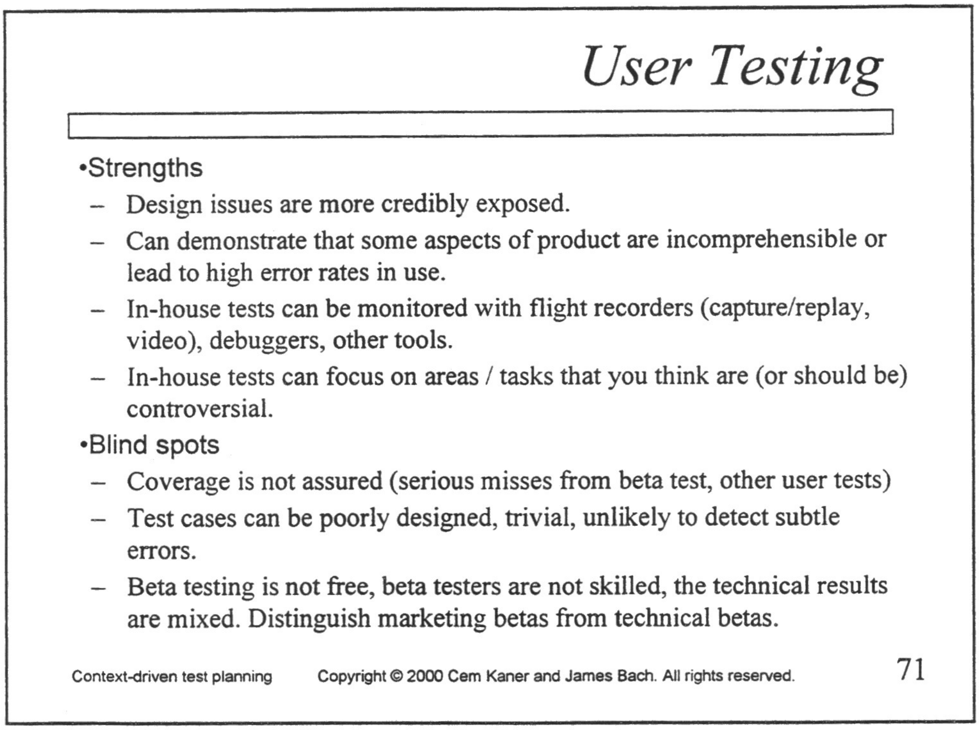
- Many of us who think about testing in terms of risk, analogize testing of software to the testing of theories:
  - Karl Popper, in his famous essay *Conjectures and Refutations*, lays out the proposition that a scientific theory gains credibility by being subjected to (and passing) harsh tests that are intended to refute the theory.
  - We can gain confidence in a program by testing it harshly (if it passes the tests). Subjecting it to easy tests doesn't tell us much about what will happen to the program in the field.

Risk-Based Testing: Interesting Papers

- Stale Amland, Risk Based Testing
- James Bach, Reframing Requirements Analysis
- James Bach, Risk and Requirements- Based Testing
- James Bach, James Bach on Risk-Based Testing
- Stale Amland & Hans Schaefer, Risk based testing, a response
- Carl Popper, Conjectures & Refutations

User Testing

- Tag line
  - Strive for realism
  - Let's try this with real humans (for a change).
- Fundamental question or goal
  - Identify failures that will arise in the hands of a person, i.e. breakdowns in the overall human/machine/software system.
- Paradigmatic case(s)
  - Beta testing
  - In-house experiments using a stratified sample of target market
  - Usability testing



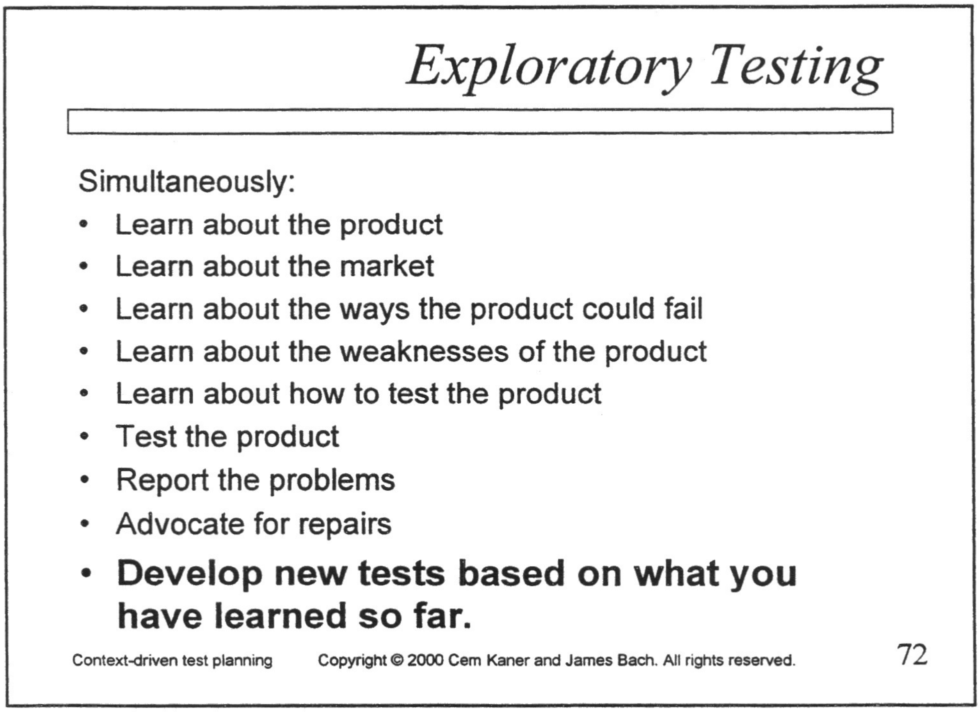
User Testing

•Strengths

- Design issues are more credibly exposed.
- Can demonstrate that some aspects of product are incomprehensible or lead to high error rates in use.
- In-house tests can be monitored with flight recorders (capture/replay, video), debuggers, other tools.
- In-house tests can focus on areas / tasks that you think are (or should be) controversial.

•Blind spots

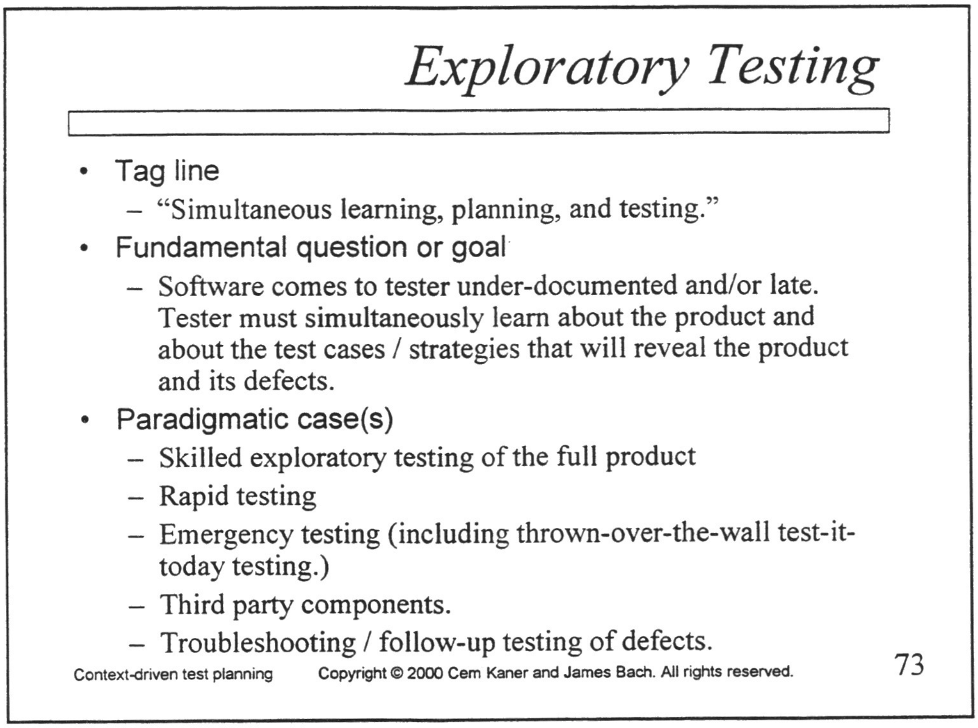
- Coverage is not assured (serious misses from beta test, other user tests)
- Test cases can be poorly designed, trivial, unlikely to detect subtle errors.
- Beta testing is not free, beta testers are not skilled, the technical results are mixed. Distinguish marketing betas from technical betas.



Exploratory Testing

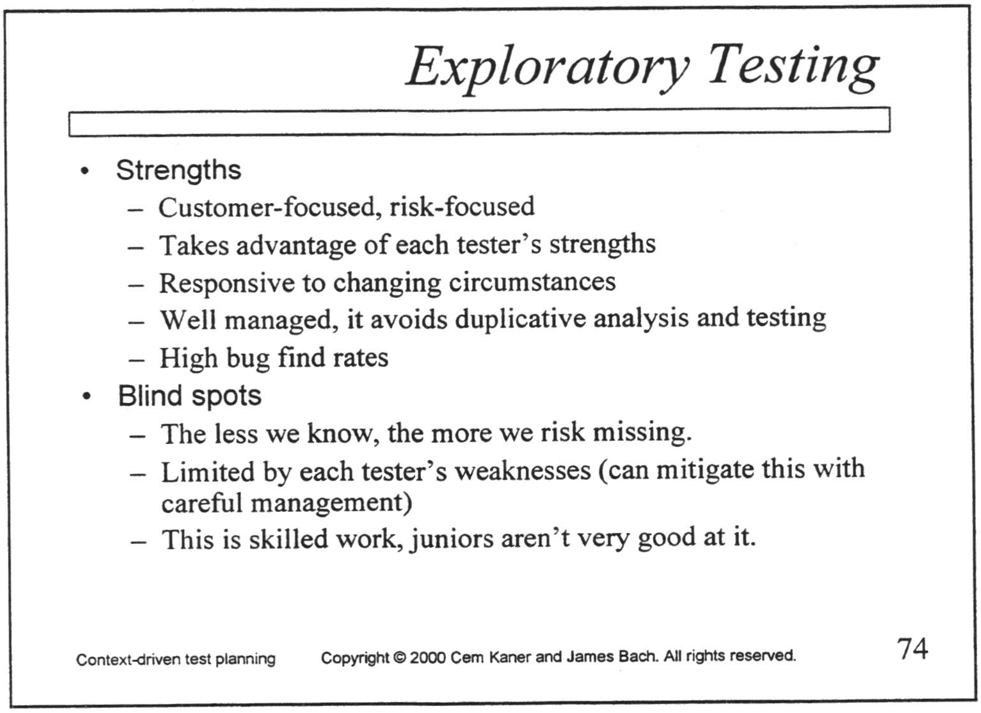
Simultaneously:

- Learn about the product
- Learn about the market
- Learn about the ways the product could fail
- Learn about the weaknesses of the product
- Learn about how to test the product
- Test the product
- Report the problems
- Advocate for repairs
- **Develop new tests based on what you have learned so far.**



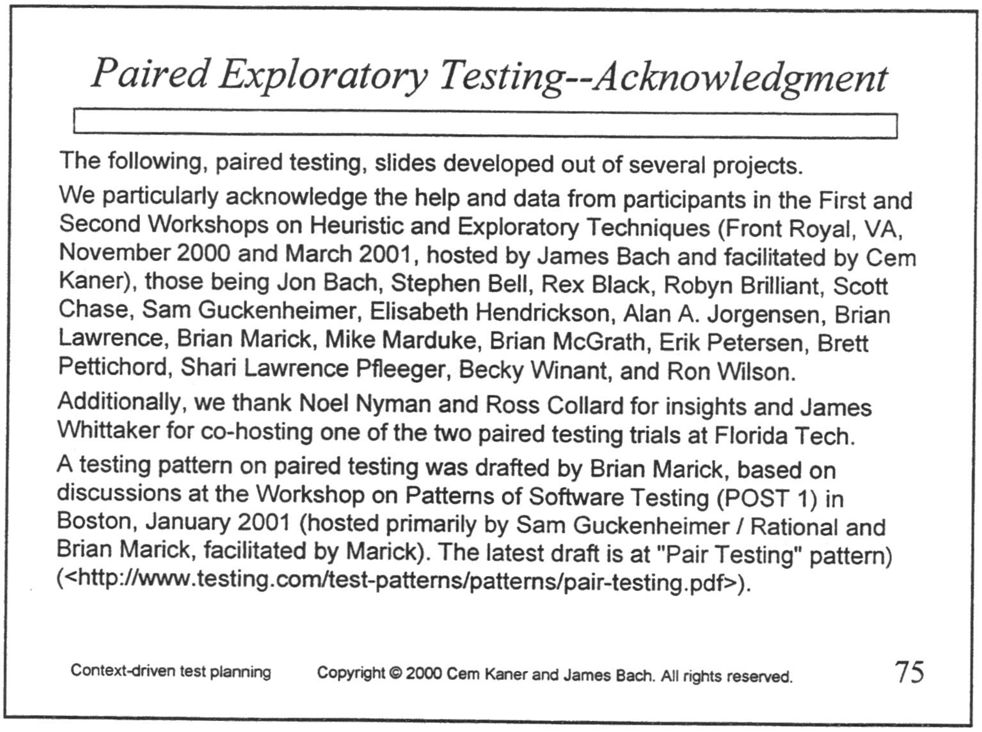
Exploratory Testing

- Tag line
  - "Simultaneous learning, planning, and testing."
- Fundamental question or goal
  - Software comes to tester under-documented and/or late. Tester must simultaneously learn about the product and about the test cases / strategies that will reveal the product and its defects.
- Paradigmatic case(s)
  - Skilled exploratory testing of the full product
  - Rapid testing
  - Emergency testing (including thrown-over-the-wall test-it-today testing.)
  - Third party components.
  - Troubleshooting / follow-up testing of defects.



Exploratory Testing

- Strengths
  - Customer-focused, risk-focused
  - Takes advantage of each tester's strengths
  - Responsive to changing circumstances
  - Well managed, it avoids duplicative analysis and testing
  - High bug find rates
- Blind spots
  - The less we know, the more we risk missing.
  - Limited by each tester's weaknesses (can mitigate this with careful management)
  - This is skilled work, juniors aren't very good at it.



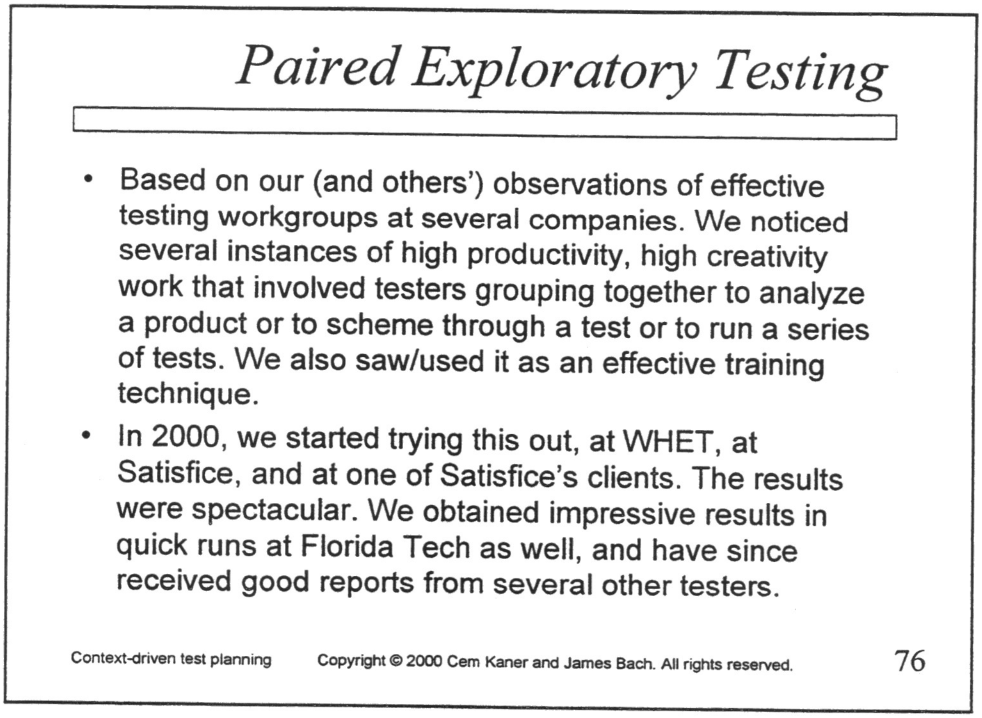
Paired Exploratory Testing--Acknowledgment

The following, paired testing, slides developed out of several projects.

We particularly acknowledge the help and data from participants in the First and Second Workshops on Heuristic and Exploratory Techniques (Front Royal, VA, November 2000 and March 2001, hosted by James Bach and facilitated by Cem Kaner), those being Jon Bach, Stephen Bell, Rex Black, Robyn Brilliant, Scott Chase, Sam Guckenheimer, Elisabeth Hendrickson, Alan A. Jorgensen, Brian Lawrence, Brian Marick, Mike Marduke, Brian McGrath, Erik Petersen, Brett Pettichord, Shari Lawrence Pfleeger, Becky Winant, and Ron Wilson.

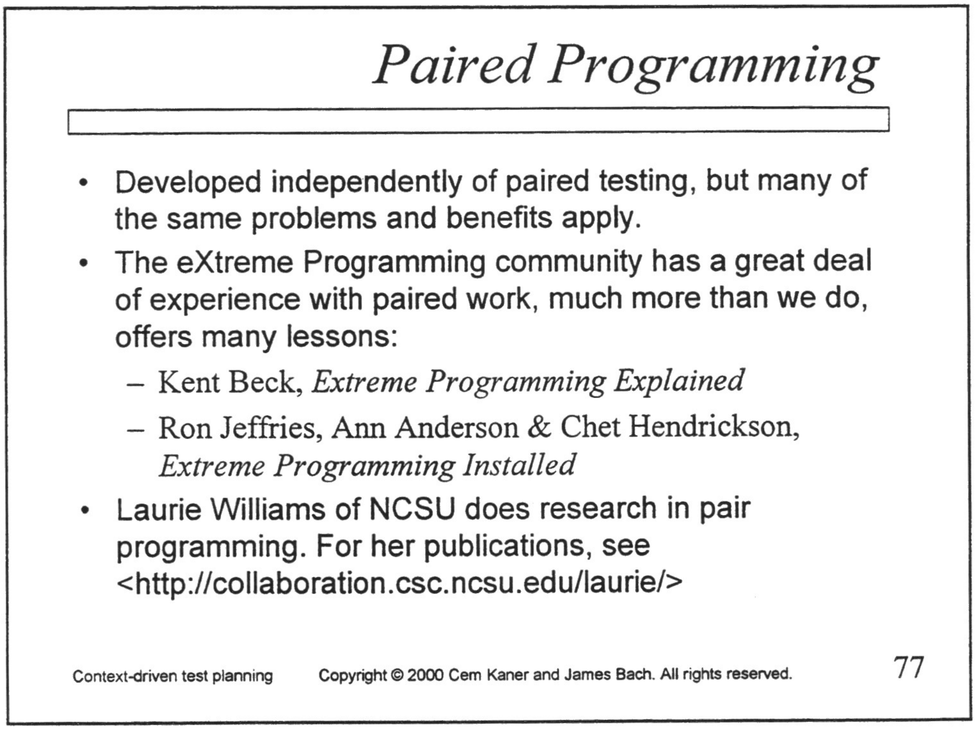
Additionally, we thank Noel Nyman and Ross Collard for insights and James Whittaker for co-hosting one of the two paired testing trials at Florida Tech.

A testing pattern on paired testing was drafted by Brian Marick, based on discussions at the Workshop on Patterns of Software Testing (POST 1) in Boston, January 2001 (hosted primarily by Sam Guckenheimer / Rational and Brian Marick, facilitated by Marick). The latest draft is at "Pair Testing" pattern) (<http://www.testing.com/test-patterns/patterns/pair-testing.pdf>).



Paired Exploratory Testing

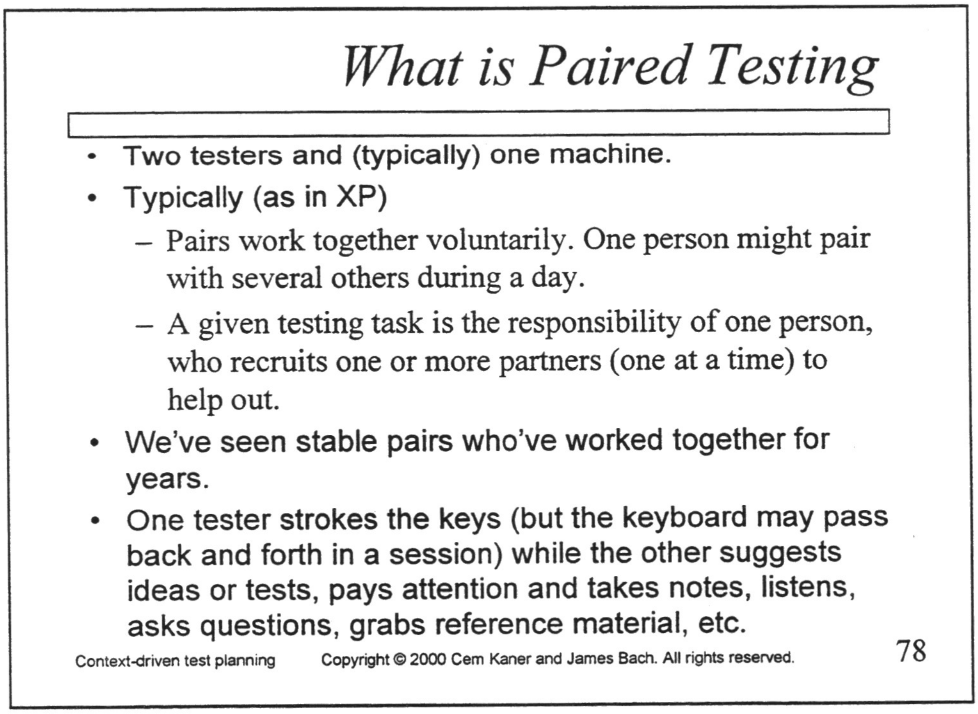
- Based on our (and others') observations of effective testing workgroups at several companies. We noticed several instances of high productivity, high creativity work that involved testers grouping together to analyze a product or to scheme through a test or to run a series of tests. We also saw/used it as an effective training technique.
- In 2000, we started trying this out, at WHET, at Satisfice, and at one of Satisfice's clients. The results were spectacular. We obtained impressive results in quick runs at Florida Tech as well, and have since received good reports from several other testers.



Paired Programming

- Developed independently of paired testing, but many of the same problems and benefits apply.
- The eXtreme Programming community has a great deal of experience with paired work, much more than we do, offers many lessons:
  - Kent Beck, *Extreme Programming Explained*
  - Ron Jeffries, Ann Anderson & Chet Hendrickson, *Extreme Programming Installed*
- Laurie Williams of NCSU does research in pair programming. For her publications, see <http://collaboration.csc.ncsu.edu/laurie/>

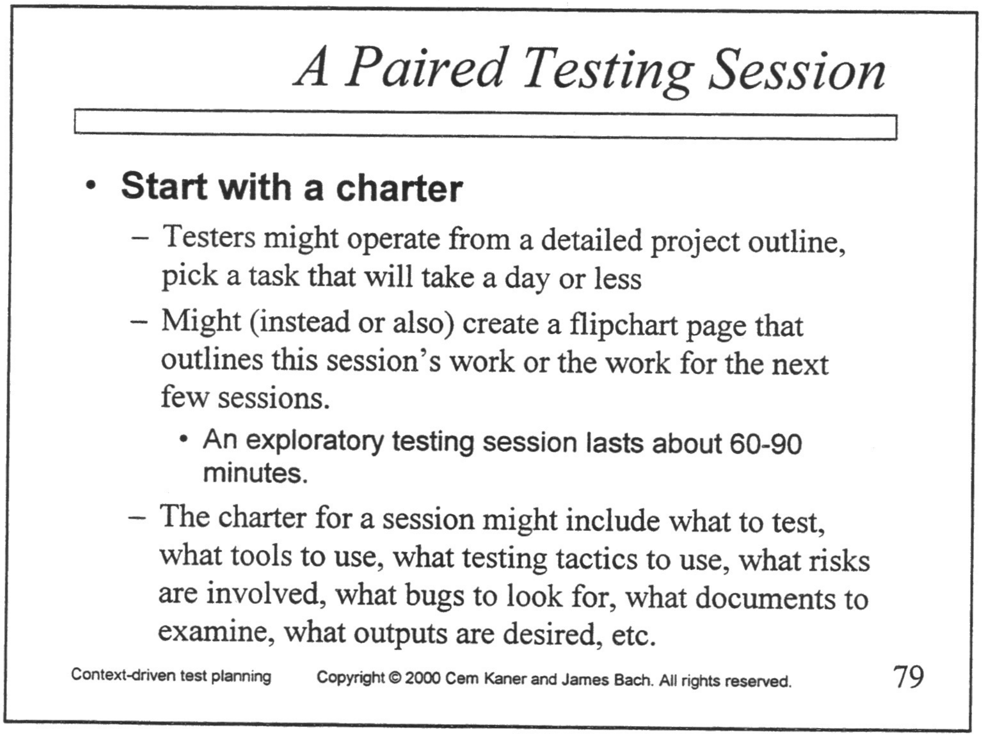
Context-driven test planning 77



What is Paired Testing

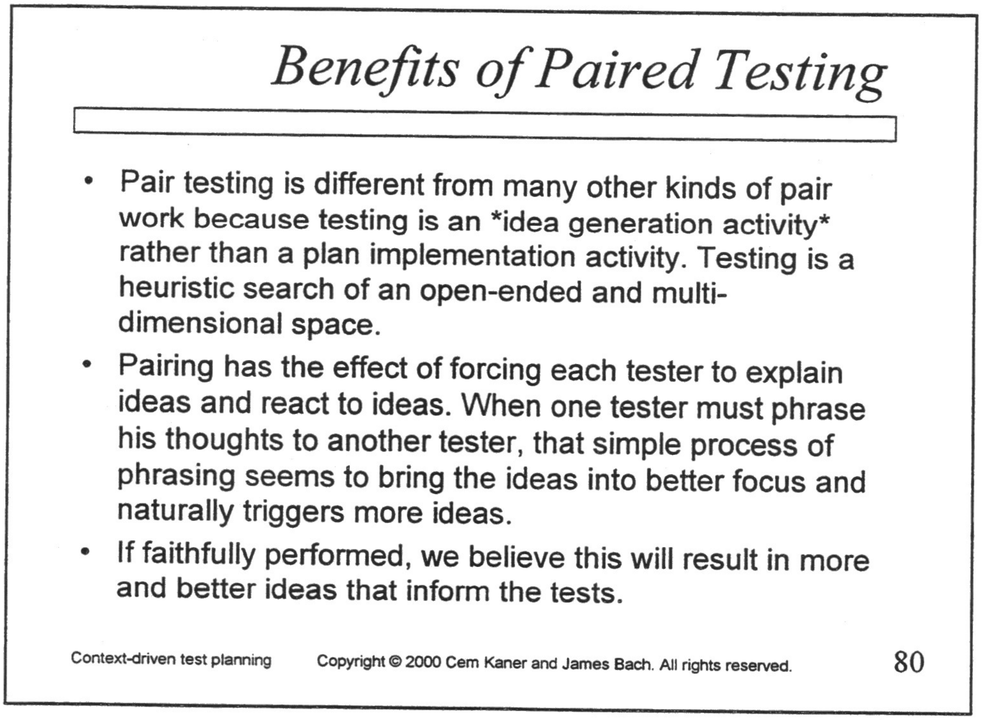
- Two testers and (typically) one machine.
- Typically (as in XP)
  - Pairs work together voluntarily. One person might pair with several others during a day.
  - A given testing task is the responsibility of one person, who recruits one or more partners (one at a time) to help out.
- We've seen stable pairs who've worked together for years.
- One tester strokes the keys (but the keyboard may pass back and forth in a session) while the other suggests ideas or tests, pays attention and takes notes, listens, asks questions, grabs reference material, etc.

Context-driven test planning 78



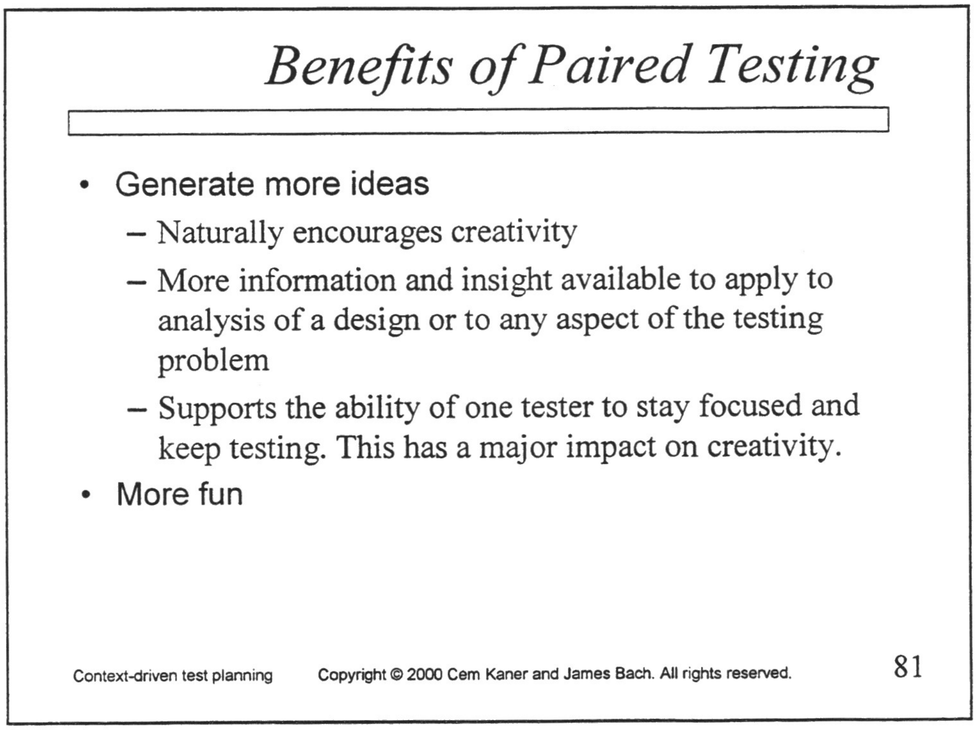
A Paired Testing Session

- **Start with a charter**
  - Testers might operate from a detailed project outline, pick a task that will take a day or less
  - Might (instead or also) create a flipchart page that outlines this session's work or the work for the next few sessions.
    - An exploratory testing session lasts about 60-90 minutes.
  - The charter for a session might include what to test, what tools to use, what testing tactics to use, what risks are involved, what bugs to look for, what documents to examine, what outputs are desired, etc.



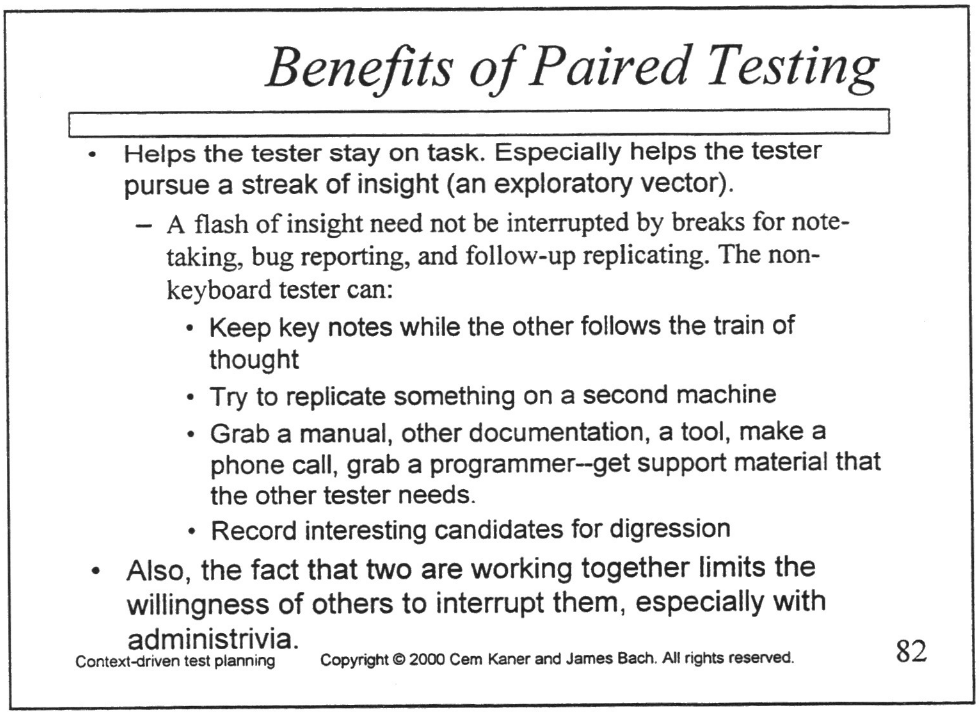
Benefits of Paired Testing

- Pair testing is different from many other kinds of pair work because testing is an *idea generation activity* rather than a plan implementation activity. Testing is a heuristic search of an open-ended and multi-dimensional space.
- Pairing has the effect of forcing each tester to explain ideas and react to ideas. When one tester must phrase his thoughts to another tester, that simple process of phrasing seems to bring the ideas into better focus and naturally triggers more ideas.
- If faithfully performed, we believe this will result in more and better ideas that inform the tests.



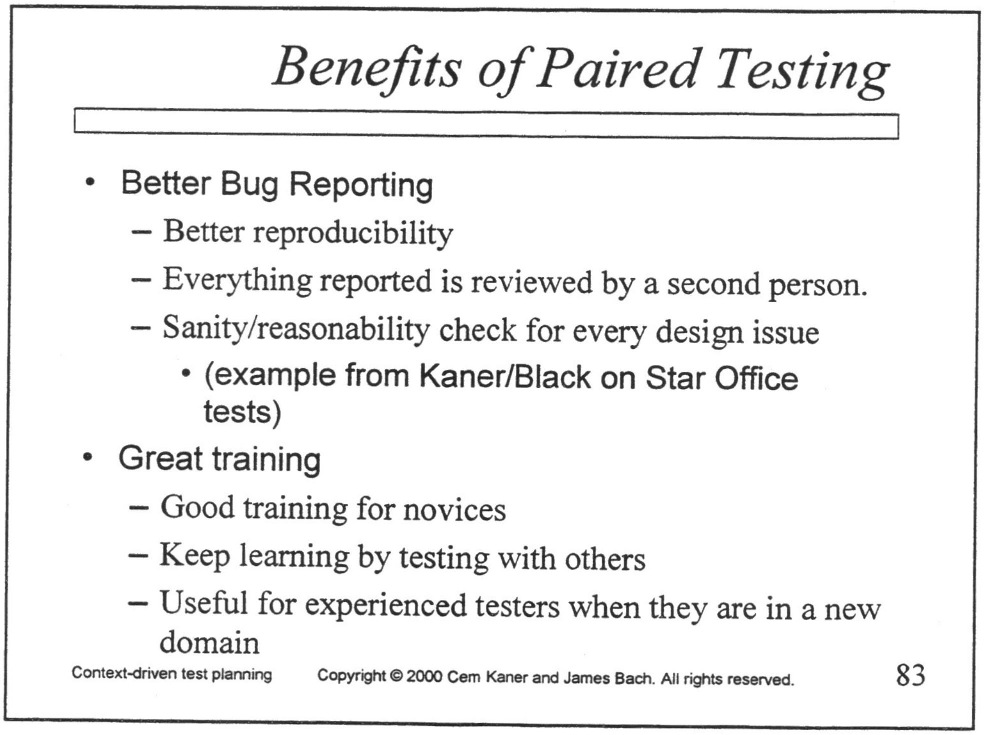
Benefits of Paired Testing

- Generate more ideas
  - – Naturally encourages creativity
  - – More information and insight available to apply to analysis of a design or to any aspect of the testing problem
  - – Supports the ability of one tester to stay focused and keep testing. This has a major impact on creativity.
- More fun



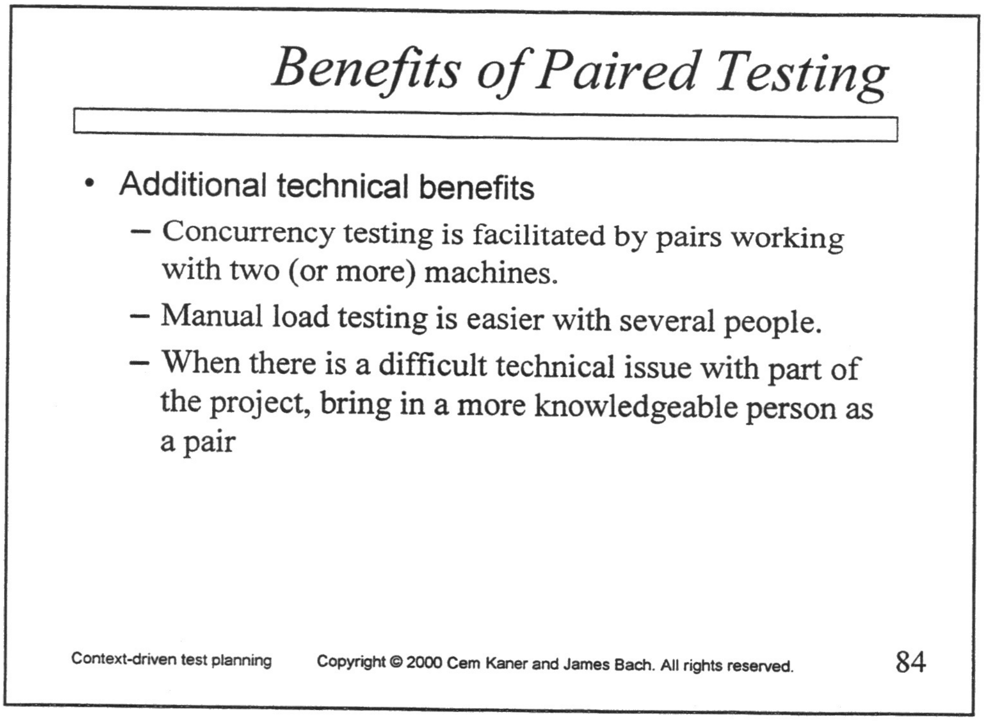
Benefits of Paired Testing

- Helps the tester stay on task. Especially helps the tester pursue a streak of insight (an exploratory vector).
  - – A flash of insight need not be interrupted by breaks for note-taking, bug reporting, and follow-up replicating. The non-keyboard tester can:
    - Keep key notes while the other follows the train of thought
    - Try to replicate something on a second machine
    - Grab a manual, other documentation, a tool, make a phone call, grab a programmer--get support material that the other tester needs.
    - Record interesting candidates for digression
- Also, the fact that two are working together limits the willingness of others to interrupt them, especially with administrivia.



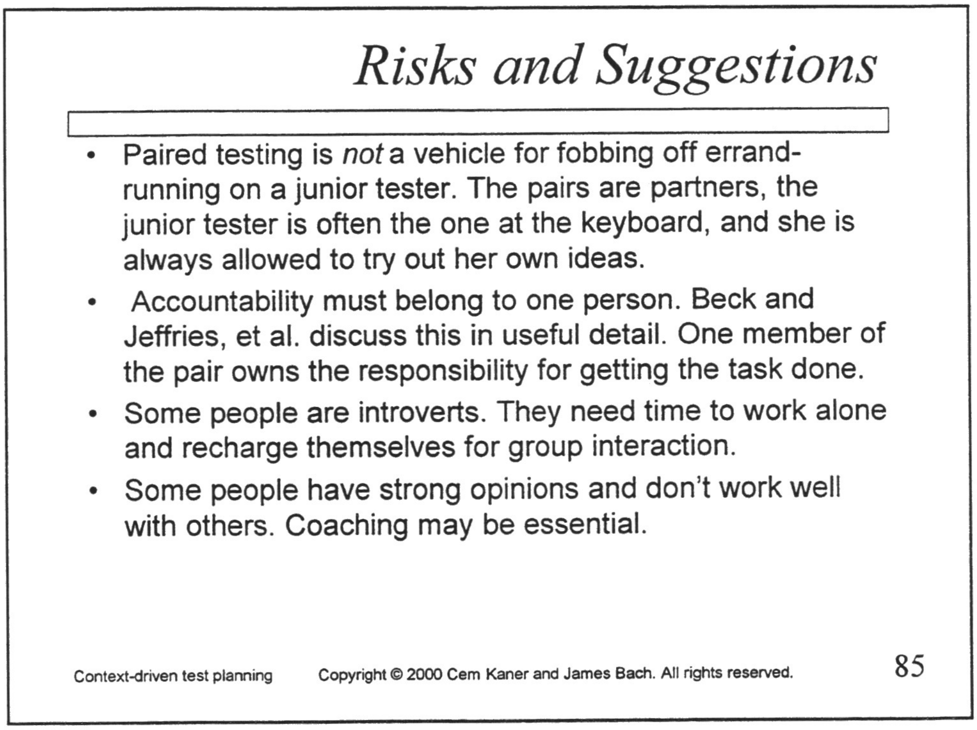
Benefits of Paired Testing

- Better Bug Reporting
  - Better reproducibility
  - Everything reported is reviewed by a second person.
  - Sanity/reasonability check for every design issue
    - (example from Kaner/Black on Star Office tests)
- Great training
  - Good training for novices
  - Keep learning by testing with others
  - Useful for experienced testers when they are in a new domain



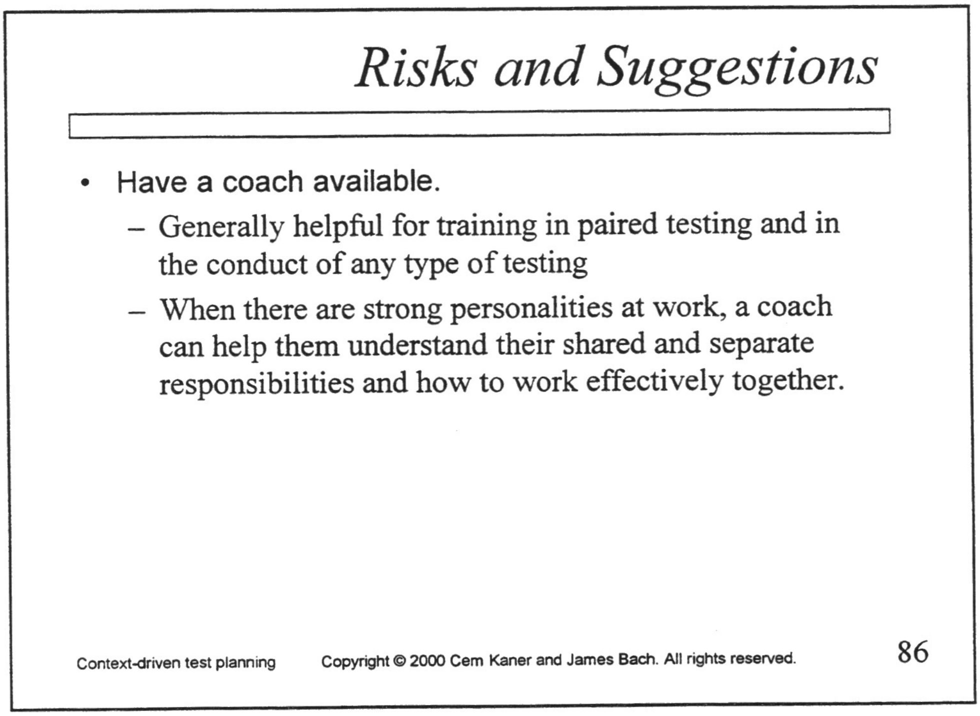
Benefits of Paired Testing

- Additional technical benefits
  - Concurrency testing is facilitated by pairs working with two (or more) machines.
  - Manual load testing is easier with several people.
  - When there is a difficult technical issue with part of the project, bring in a more knowledgeable person as a pair



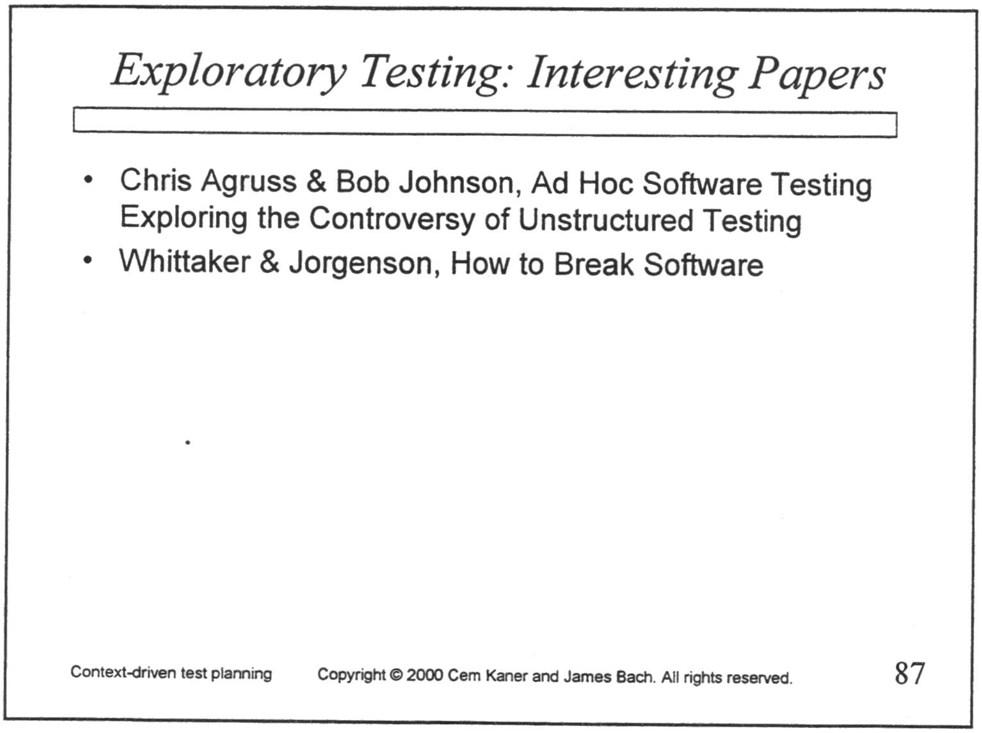
Risks and Suggestions

- Paired testing is *not* a vehicle for fobbing off errand-running on a junior tester. The pairs are partners, the junior tester is often the one at the keyboard, and she is always allowed to try out her own ideas.
- Accountability must belong to one person. Beck and Jeffries, et al. discuss this in useful detail. One member of the pair owns the responsibility for getting the task done.
- Some people are introverts. They need time to work alone and recharge themselves for group interaction.
- Some people have strong opinions and don't work well with others. Coaching may be essential.



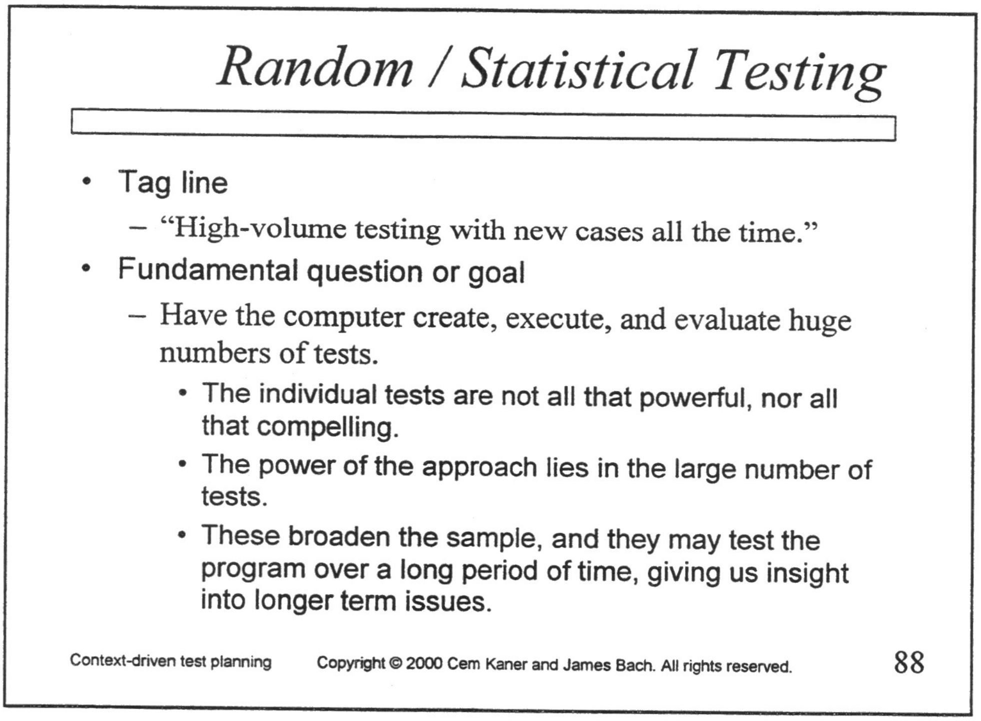
Risks and Suggestions

- Have a coach available.
  - Generally helpful for training in paired testing and in the conduct of any type of testing
  - When there are strong personalities at work, a coach can help them understand their shared and separate responsibilities and how to work effectively together.



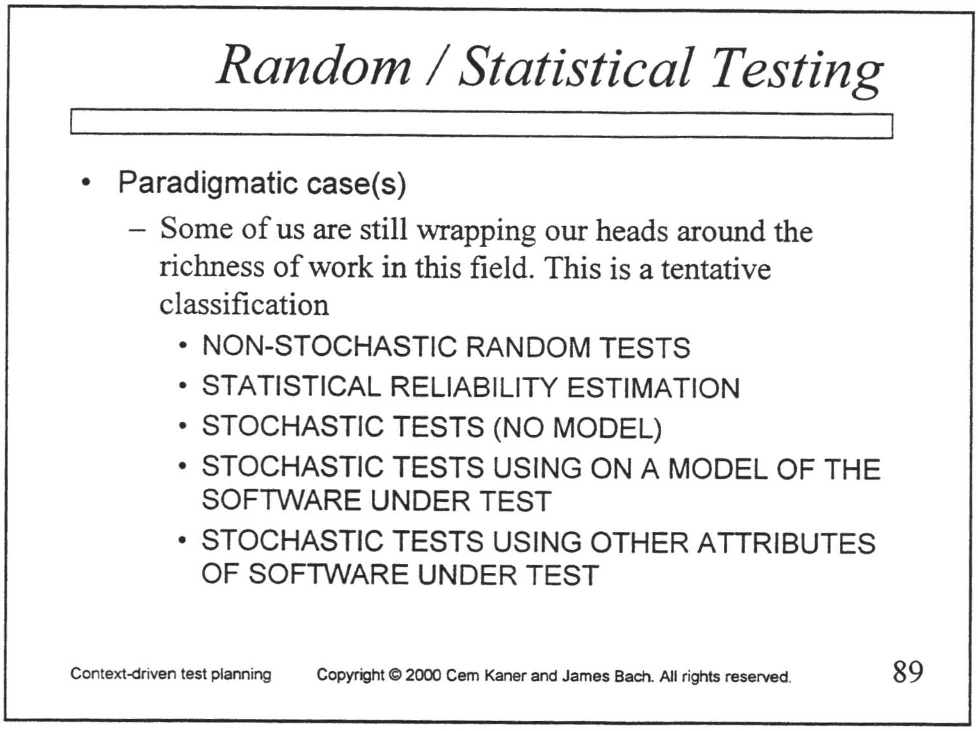
Exploratory Testing: Interesting Papers

- Chris Agruss & Bob Johnson, Ad Hoc Software Testing Exploring the Controversy of Unstructured Testing
- Whittaker & Jorgenson, How to Break Software



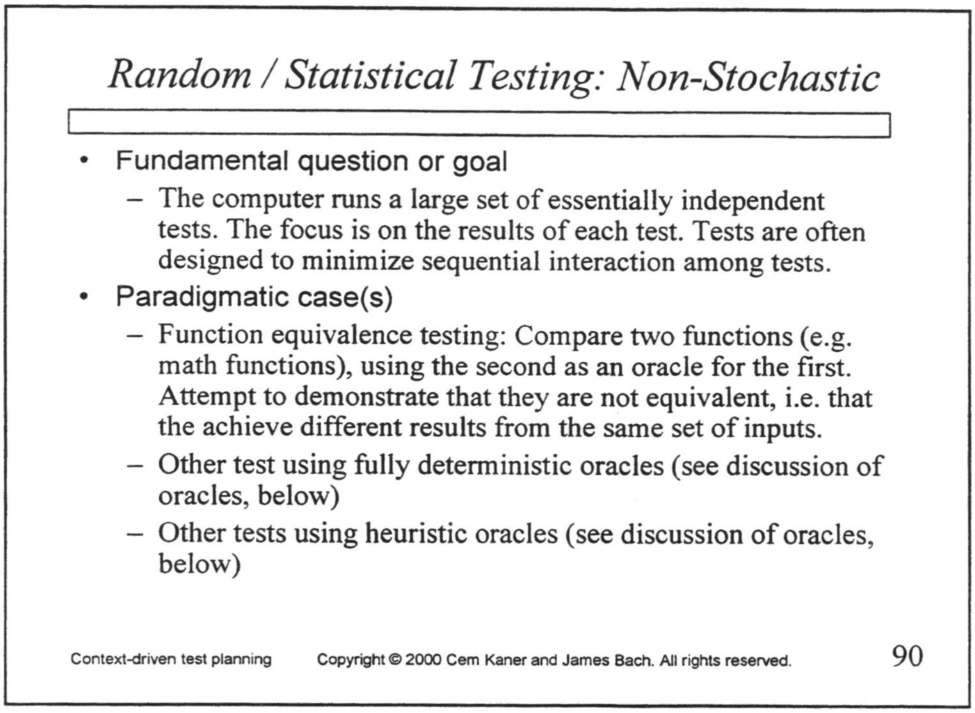
Random / Statistical Testing

- Tag line
  - "High-volume testing with new cases all the time."
- Fundamental question or goal
  - Have the computer create, execute, and evaluate huge numbers of tests.
    - The individual tests are not all that powerful, nor all that compelling.
    - The power of the approach lies in the large number of tests.
    - These broaden the sample, and they may test the program over a long period of time, giving us insight into longer term issues.



Random / Statistical Testing

- Paradigmatic case(s)
  - Some of us are still wrapping our heads around the richness of work in this field. This is a tentative classification
    - NON-STOCHASTIC RANDOM TESTS
    - STATISTICAL RELIABILITY ESTIMATION
    - STOCHASTIC TESTS (NO MODEL)
    - STOCHASTIC TESTS USING ON A MODEL OF THE SOFTWARE UNDER TEST
    - STOCHASTIC TESTS USING OTHER ATTRIBUTES OF SOFTWARE UNDER TEST



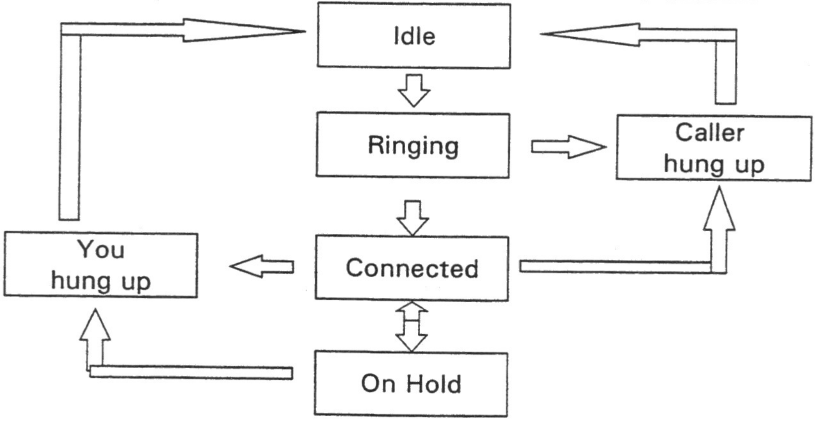
Random / Statistical Testing: Non-Stochastic

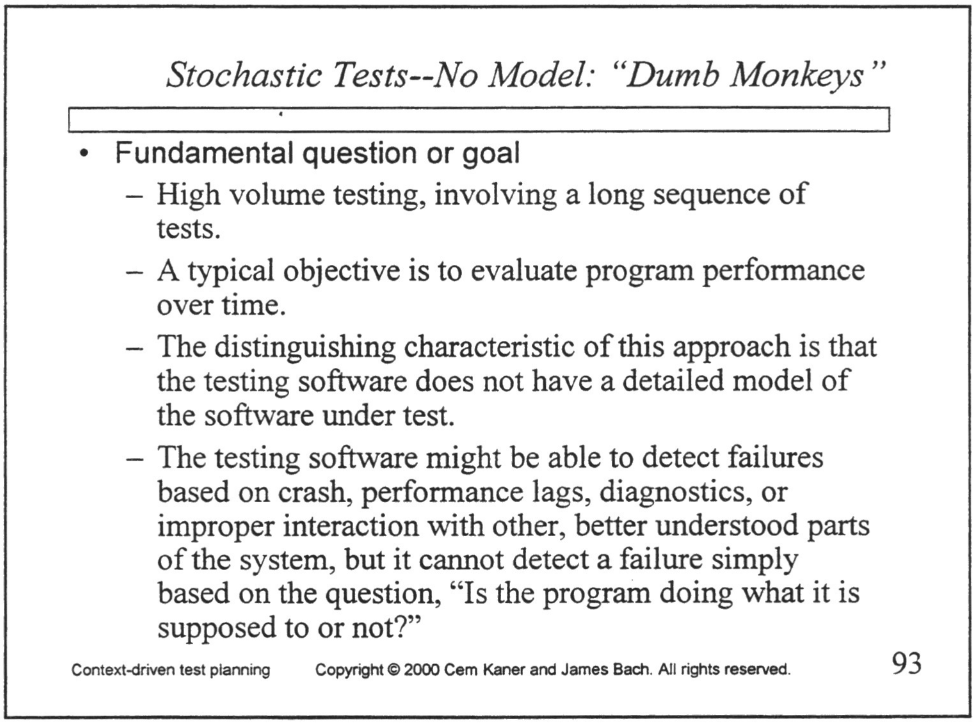
- Fundamental question or goal
  - The computer runs a large set of essentially independent tests. The focus is on the results of each test. Tests are often designed to minimize sequential interaction among tests.
- Paradigmatic case(s)
  - Function equivalence testing: Compare two functions (e.g. math functions), using the second as an oracle for the first. Attempt to demonstrate that they are not equivalent, i.e. that the achieve different results from the same set of inputs.
  - Other test using fully deterministic oracles (see discussion of oracles, below)
  - Other tests using heuristic oracles (see discussion of oracles, below)

Statistical Reliability Estimation

- Fundamental question or goal
  - Use random testing (possibly stochastic, possibly oracle-based) to estimate the stability or reliability of the software. Testing is being used primarily to qualify the software, rather than to find defects.
- Paradigmatic case(s)
  - Clean-room based approaches

The Need for Stochastic Testing: An Example

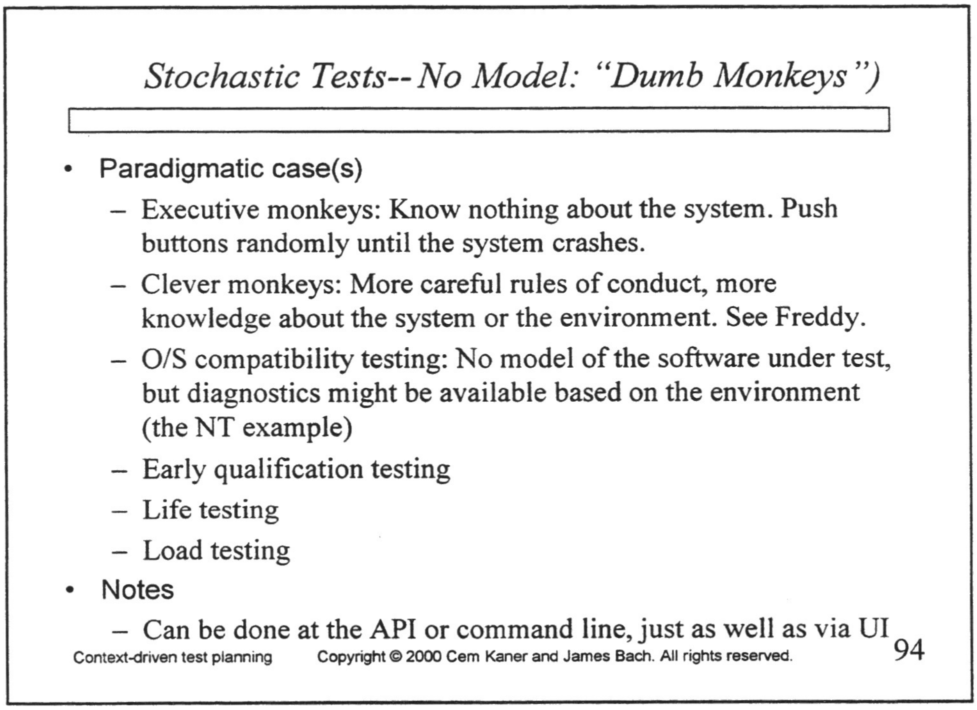




Stochastic Tests--No Model: "Dumb Monkeys"

- Fundamental question or goal
  - High volume testing, involving a long sequence of tests.
  - A typical objective is to evaluate program performance over time.
  - The distinguishing characteristic of this approach is that the testing software does not have a detailed model of the software under test.
  - The testing software might be able to detect failures based on crash, performance lags, diagnostics, or improper interaction with other, better understood parts of the system, but it cannot detect a failure simply based on the question, "Is the program doing what it is supposed to or not?"

Context-driven test planning Copyright © 2000 Cem Kaner and James Bach. All rights reserved. 93



Stochastic Tests-- No Model: "Dumb Monkeys")

- Paradigmatic case(s)
  - Executive monkeys: Know nothing about the system. Push buttons randomly until the system crashes.
  - Clever monkeys: More careful rules of conduct, more knowledge about the system or the environment. See Freddy.
  - O/S compatibility testing: No model of the software under test, but diagnostics might be available based on the environment (the NT example)
  - Early qualification testing
  - Life testing
  - Load testing
- Notes
  - Can be done at the API or command line, just as well as via UI

Context-driven test planning Copyright © 2000 Cem Kaner and James Bach. All rights reserved. 94

Stochastic, assert or diagnostics-based random tests

- Fundamental question or goal
  - High volume random testing using random sequence of fresh or pre-defined tests that may or may not self-check for pass/fail. The primary method for detecting pass/fail uses assertions (diagnostics built into the program) or other (e.g. system) diagnostics.
- Paradigmatic case(s)
  - Telephone example (asserts)
  - Embedded software example (diagnostics)

Random Testing: Stochastic, Regression-Based

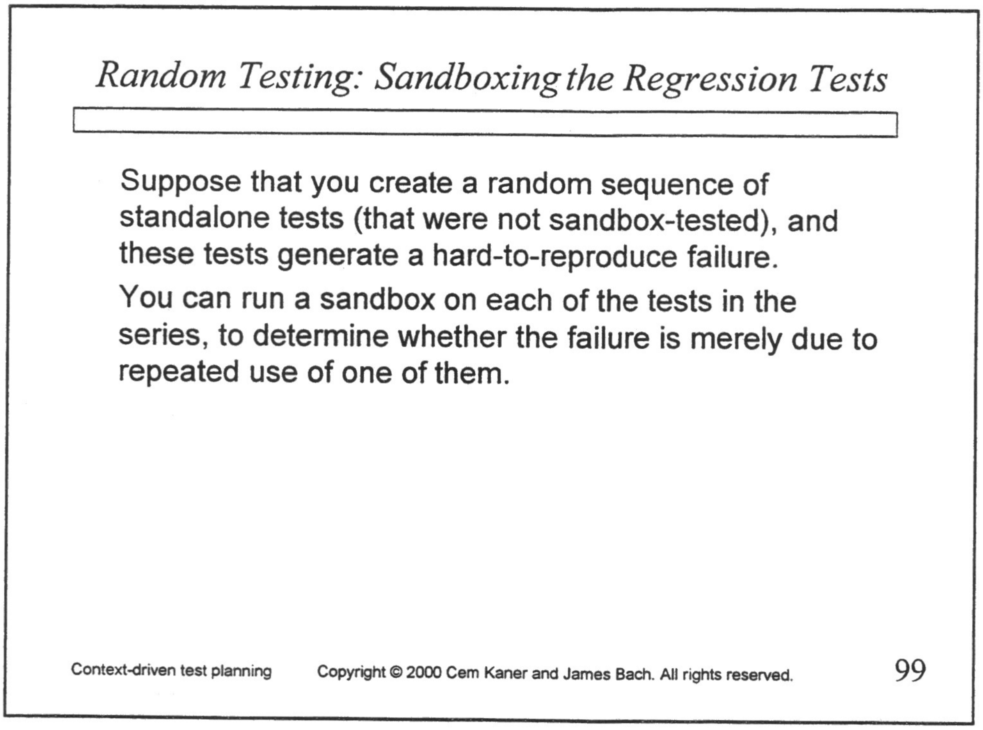
- Fundamental question or goal
  - High volume random testing using random sequence of pre-defined tests that can self-check for pass/fail.
- Paradigmatic case(s)
  - Life testing
  - Search for specific types of long-sequence defects.

Random Testing: Stochastic, Regression-Based

- Notes
  - Create a series of regression tests. Design them so that they don't reinitialize the system or force it to a standard starting state that would erase history. The tests are designed so that the automation can identify failures. Run the tests in random order over a long sequence.
  - This is a low-mental-overhead alternative to model-based testing. You get pass/fail info for every test, but without having to achieve the same depth of understanding of the software. Of course, you probably have worse coverage, less awareness of your actual coverage, and less opportunity to stumble over bugs.
  - Unless this is very carefully managed, there is a serious risk of non-reproduceability of failures.

Random Testing: Sandboxing the Regression Tests

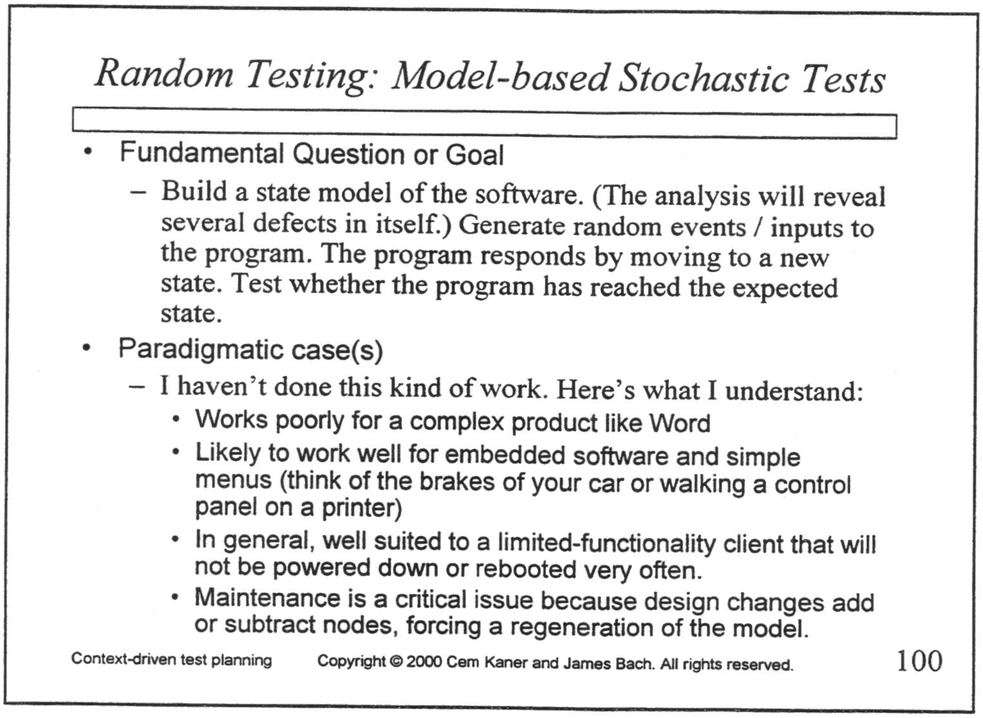
- In a random sequence of standalone tests, we might want to qualify each test, T1, T2, etc, as able to run on its own. Then, when we test a sequence of these tests, we know that errors are due to interactions among them rather than merely to cumulative effects of repetition of a single test.
- Therefore, for each Ti, we run the test on its own many times in one long series, randomly switching as many other environmental or systematic variables during this random sequence as our tools allow.
- We call this the "sandbox" series—Ti is forced to play in its own sandbox until it "proves" that it can behave properly on its own. (This is an 80/20 rule operation. We do want to avoid creating a big random test series that crashes only because one test doesn't like being run or that fails after a few runs under low memory. We want to weed out these simple causes of failure. But we don't want to spend a fortune trying to control this risk.)



Random Testing: Sandboxing the Regression Tests

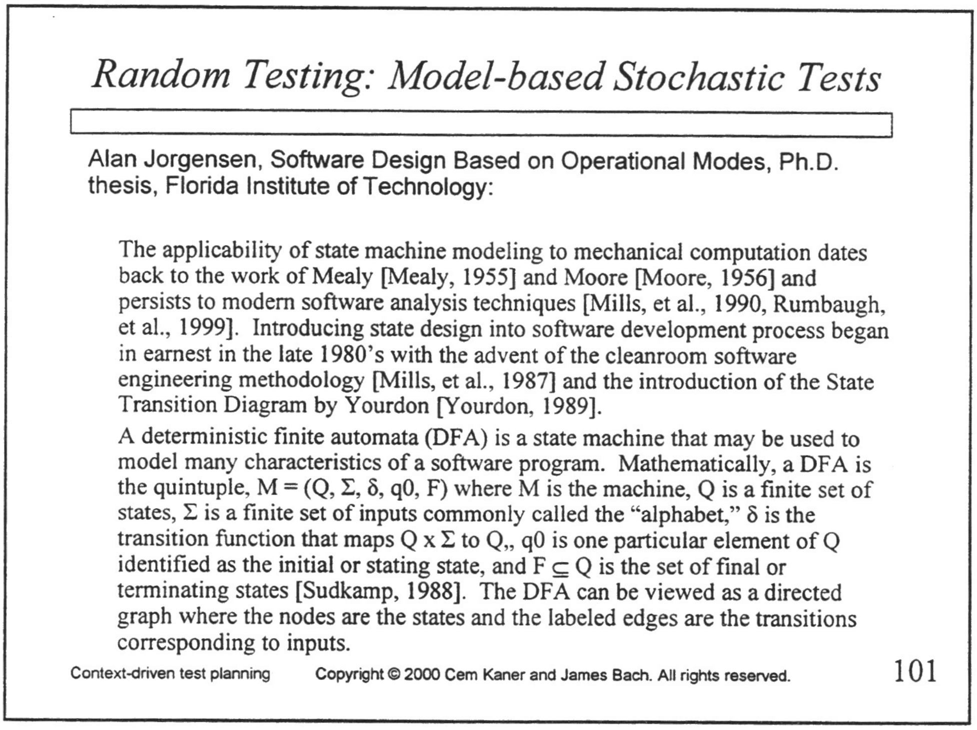
Suppose that you create a random sequence of standalone tests (that were not sandbox-tested), and these tests generate a hard-to-reproduce failure.

You can run a sandbox on each of the tests in the series, to determine whether the failure is merely due to repeated use of one of them.



Random Testing: Model-based Stochastic Tests

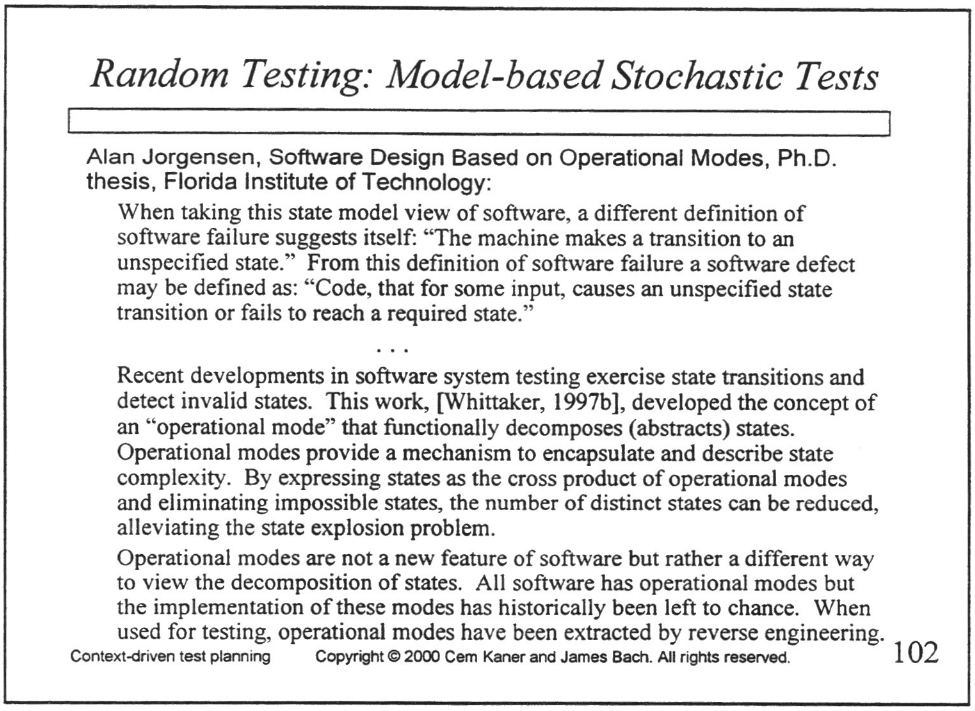
- Fundamental Question or Goal
  - Build a state model of the software. (The analysis will reveal several defects in itself.) Generate random events / inputs to the program. The program responds by moving to a new state. Test whether the program has reached the expected state.
- Paradigmatic case(s)
  - I haven't done this kind of work. Here's what I understand:
    - Works poorly for a complex product like Word
    - Likely to work well for embedded software and simple menus (think of the brakes of your car or walking a control panel on a printer)
    - In general, well suited to a limited-functionality client that will not be powered down or rebooted very often.
    - Maintenance is a critical issue because design changes add or subtract nodes, forcing a regeneration of the model.



Random Testing: Model-based Stochastic Tests

Alan Jorgensen, Software Design Based on Operational Modes, Ph.D. thesis, Florida Institute of Technology:

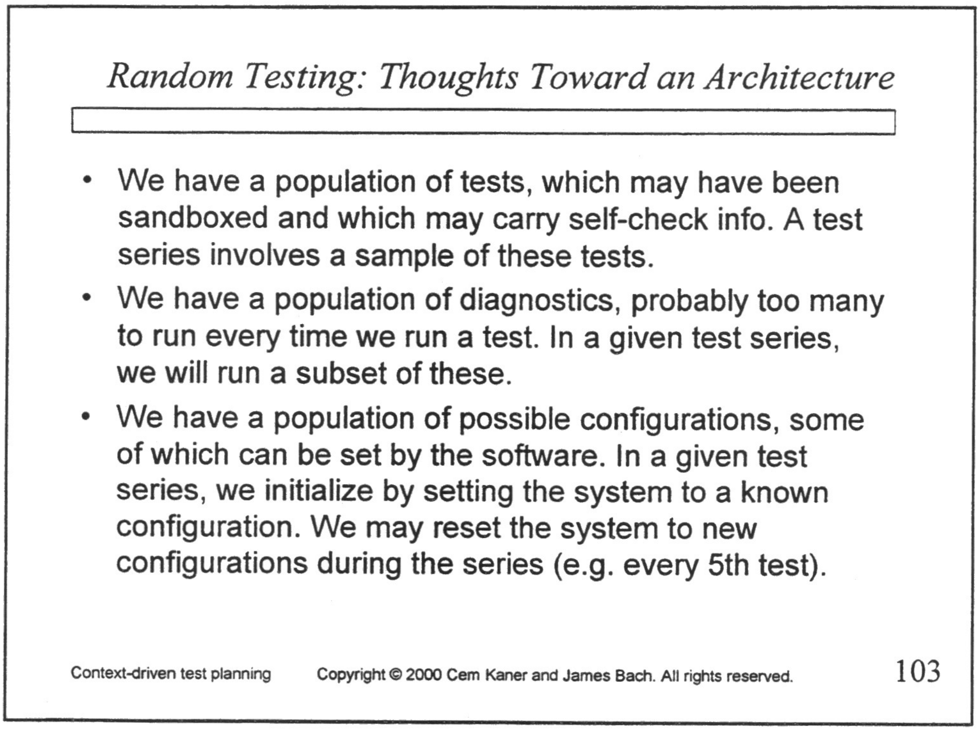
> The applicability of state machine modeling to mechanical computation dates back to the work of Mealy [Mealy, 1955] and Moore [Moore, 1956] and persists to modern software analysis techniques [Mills, et al., 1990, Rumbaugh, et al., 1999]. Introducing state design into software development process began in earnest in the late 1980's with the advent of the cleanroom software engineering methodology [Mills, et al., 1987] and the introduction of the State Transition Diagram by Yourdon [Yourdon, 1989].
>
> A deterministic finite automata (DFA) is a state machine that may be used to model many characteristics of a software program. Mathematically, a DFA is the quintuple, M = (Q, Σ, δ, q0, F) where M is the machine, Q is a finite set of states, Σ is a finite set of inputs commonly called the "alphabet," δ is the transition function that maps Q x Σ to Q,, q0 is one particular element of Q identified as the initial or stating state, and F ⊆ Q is the set of final or terminating states [Sudkamp, 1988]. The DFA can be viewed as a directed graph where the nodes are the states and the labeled edges are the transitions corresponding to inputs.



Random Testing: Model-based Stochastic Tests

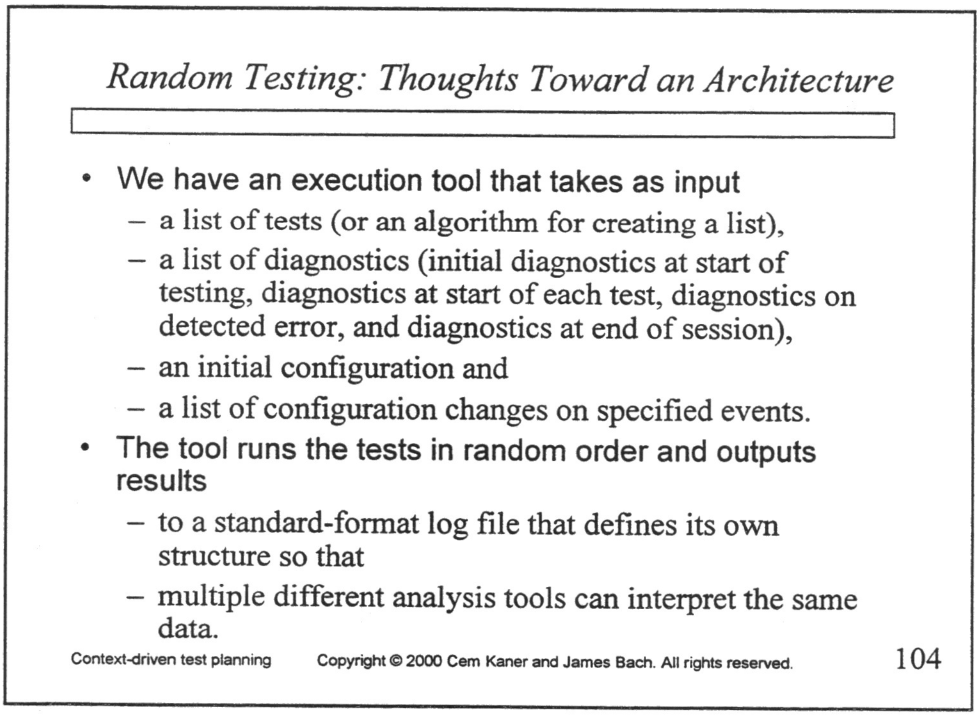
Alan Jorgensen, Software Design Based on Operational Modes, Ph.D. thesis, Florida Institute of Technology:

> When taking this state model view of software, a different definition of software failure suggests itself: "The machine makes a transition to an unspecified state." From this definition of software failure a software defect may be defined as: "Code, that for some input, causes an unspecified state transition or fails to reach a required state."
>
> . . .
>
> Recent developments in software system testing exercise state transitions and detect invalid states. This work, [Whittaker, 1997b], developed the concept of an "operational mode" that functionally decomposes (abstracts) states. Operational modes provide a mechanism to encapsulate and describe state complexity. By expressing states as the cross product of operational modes and eliminating impossible states, the number of distinct states can be reduced, alleviating the state explosion problem.
>
> Operational modes are not a new feature of software but rather a different way to view the decomposition of states. All software has operational modes but the implementation of these modes has historically been left to chance. When used for testing, operational modes have been extracted by reverse engineering.



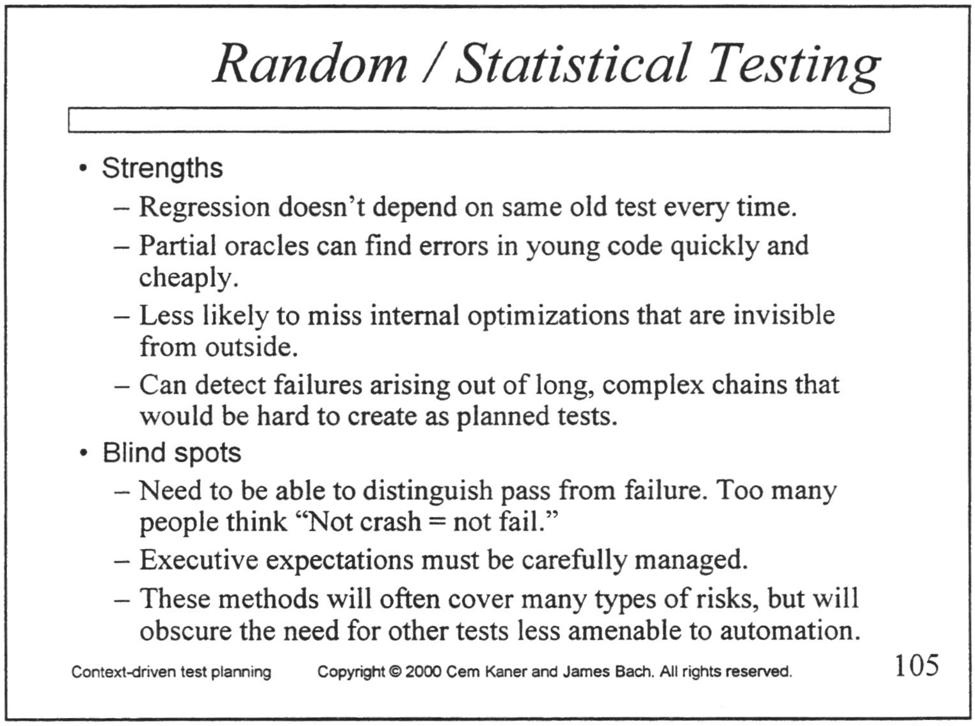
Random Testing: Thoughts Toward an Architecture

- We have a population of tests, which may have been sandboxed and which may carry self-check info. A test series involves a sample of these tests.
- We have a population of diagnostics, probably too many to run every time we run a test. In a given test series, we will run a subset of these.
- We have a population of possible configurations, some of which can be set by the software. In a given test series, we initialize by setting the system to a known configuration. We may reset the system to new configurations during the series (e.g. every 5th test).



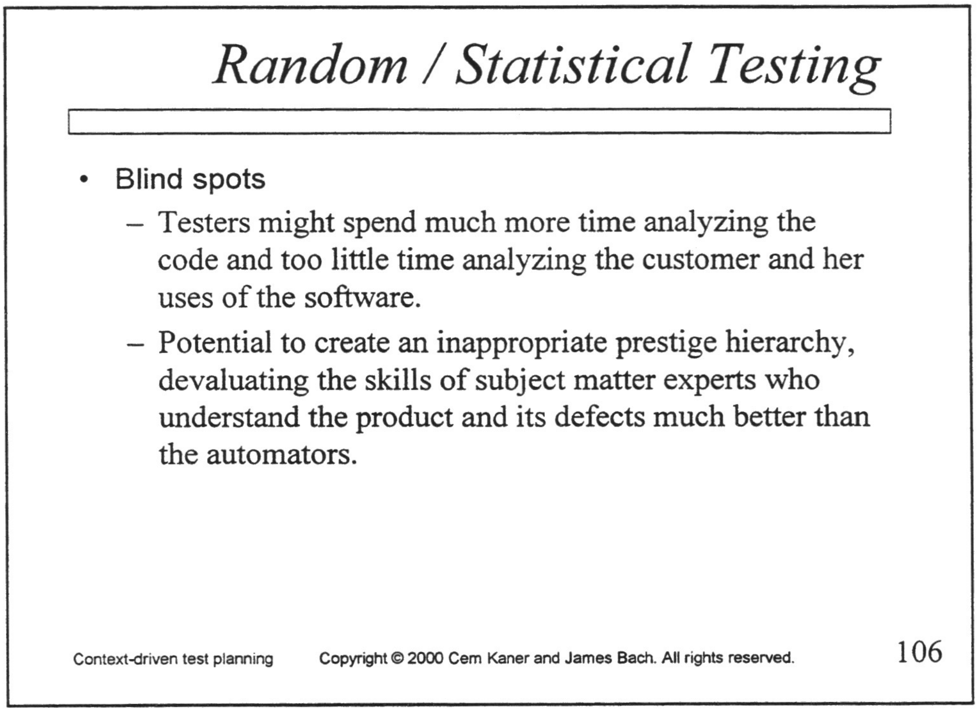
Random Testing: Thoughts Toward an Architecture

- We have an execution tool that takes as input
  - a list of tests (or an algorithm for creating a list),
  - a list of diagnostics (initial diagnostics at start of testing, diagnostics at start of each test, diagnostics on detected error, and diagnostics at end of session),
  - an initial configuration and
  - a list of configuration changes on specified events.
- The tool runs the tests in random order and outputs results
  - to a standard-format log file that defines its own structure so that
  - multiple different analysis tools can interpret the same data.



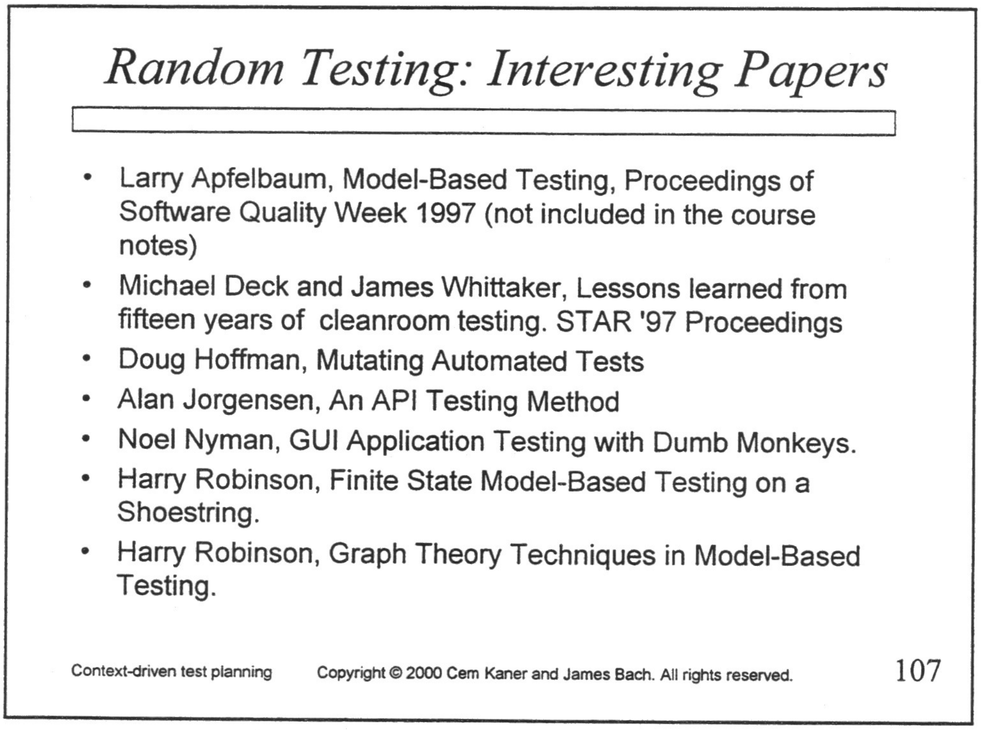
Random / Statistical Testing

- Strengths
  - – Regression doesn't depend on same old test every time.
  - – Partial oracles can find errors in young code quickly and cheaply.
  - – Less likely to miss internal optimizations that are invisible from outside.
  - – Can detect failures arising out of long, complex chains that would be hard to create as planned tests.
- Blind spots
  - – Need to be able to distinguish pass from failure. Too many people think "Not crash = not fail."
  - – Executive expectations must be carefully managed.
  - – These methods will often cover many types of risks, but will obscure the need for other tests less amenable to automation.



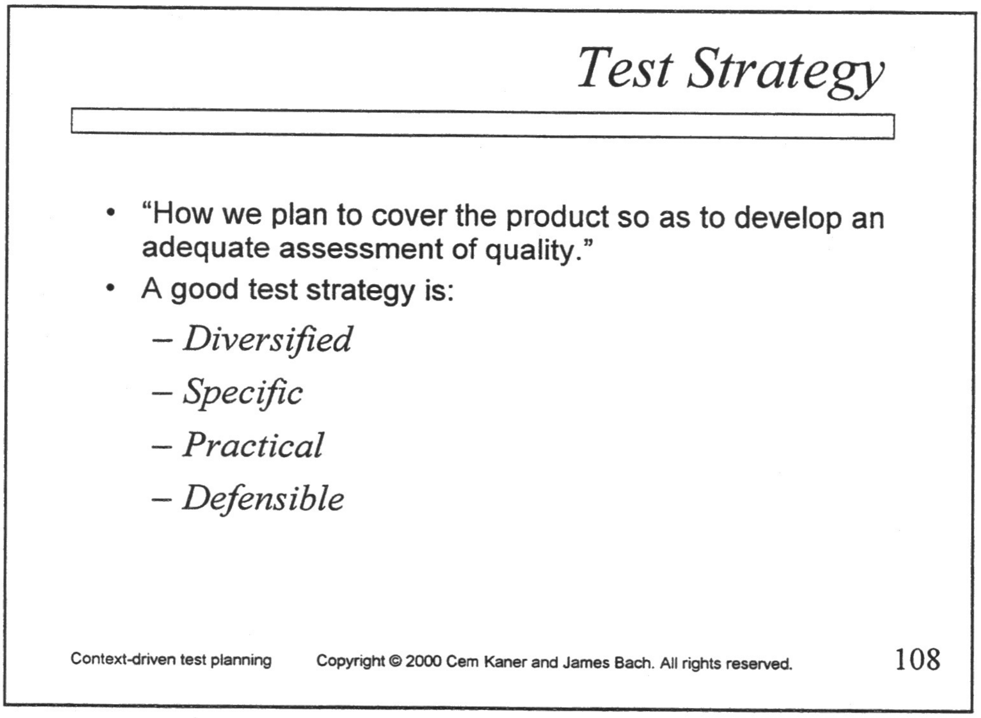
Random / Statistical Testing

- Blind spots
  - – Testers might spend much more time analyzing the code and too little time analyzing the customer and her uses of the software.
  - – Potential to create an inappropriate prestige hierarchy, devaluating the skills of subject matter experts who understand the product and its defects much better than the automators.



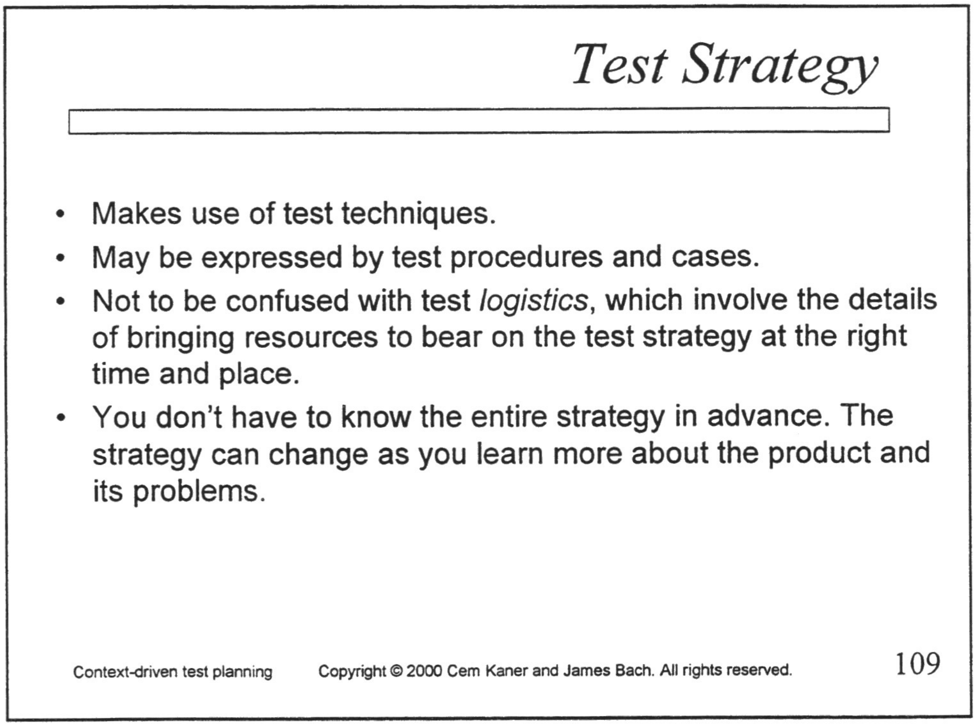
Random Testing: Interesting Papers

- Larry Apfelbaum, Model-Based Testing, Proceedings of Software Quality Week 1997 (not included in the course notes)
- Michael Deck and James Whittaker, Lessons learned from fifteen years of cleanroom testing. STAR '97 Proceedings
- Doug Hoffman, Mutating Automated Tests
- Alan Jorgensen, An API Testing Method
- Noel Nyman, GUI Application Testing with Dumb Monkeys.
- Harry Robinson, Finite State Model-Based Testing on a Shoestring.
- Harry Robinson, Graph Theory Techniques in Model-Based Testing.



Test Strategy

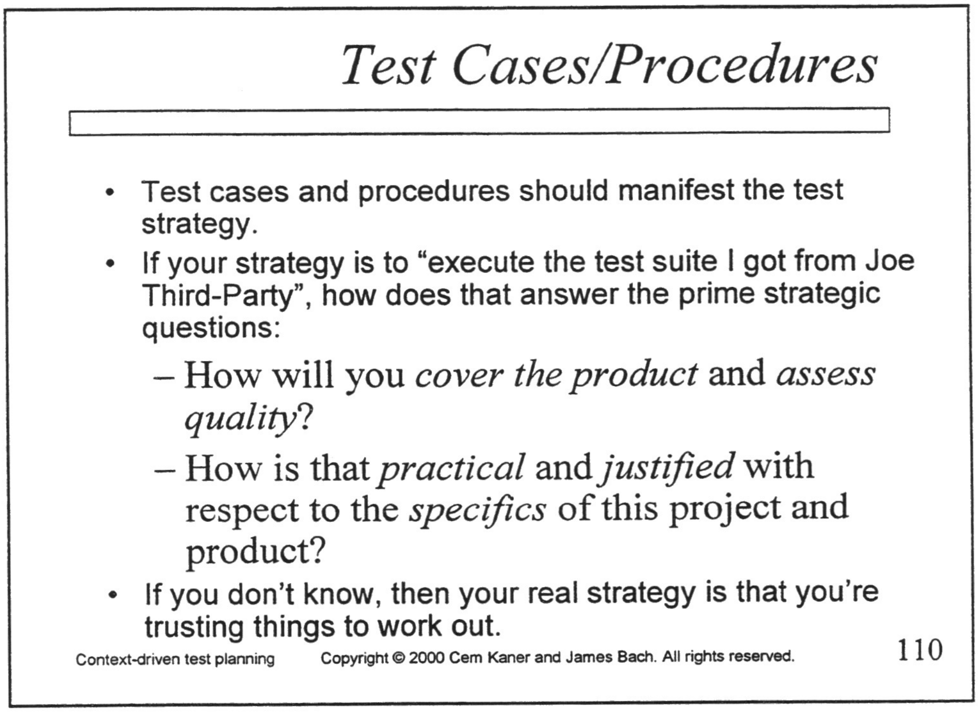
- "How we plan to cover the product so as to develop an adequate assessment of quality."
- A good test strategy is:
  - *Diversified*
  - *Specific*
  - *Practical*
  - *Defensible*



Test Strategy

- Makes use of test techniques.
- May be expressed by test procedures and cases.
- Not to be confused with test *logistics*, which involve the details of bringing resources to bear on the test strategy at the right time and place.
- You don't have to know the entire strategy in advance. The strategy can change as you learn more about the product and its problems.

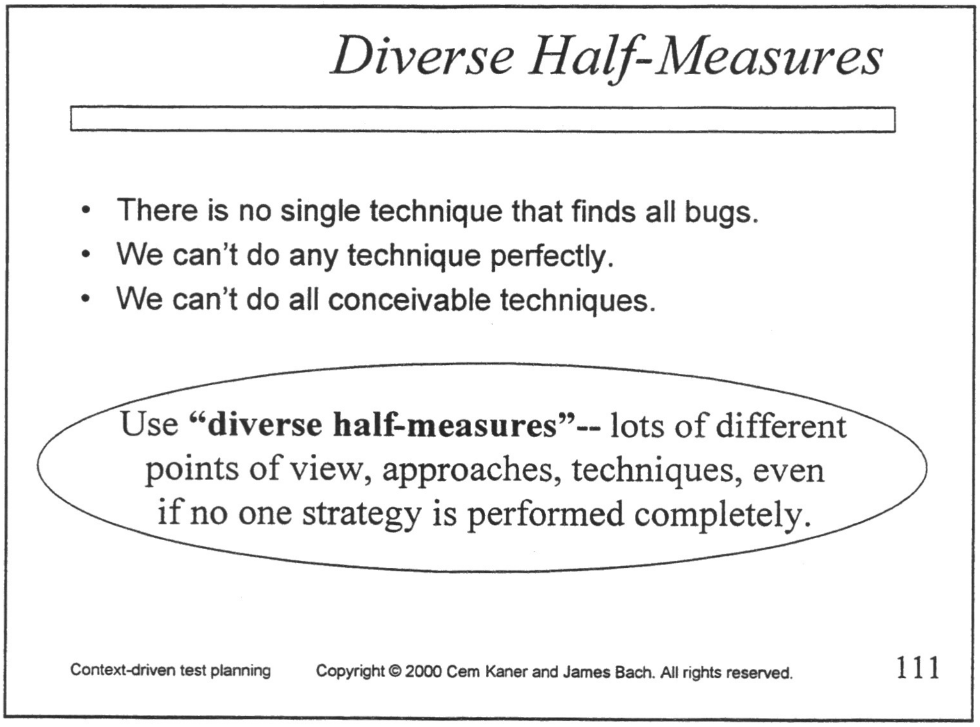
Context-driven test planning 109



Test Cases/Procedures

- Test cases and procedures should manifest the test strategy.
- If your strategy is to "execute the test suite I got from Joe Third-Party", how does that answer the prime strategic questions:
  - How will you *cover the product* and *assess quality*?
  - How is that *practical* and *justified* with respect to the *specifics* of this project and product?
- If you don't know, then your real strategy is that you're trusting things to work out.

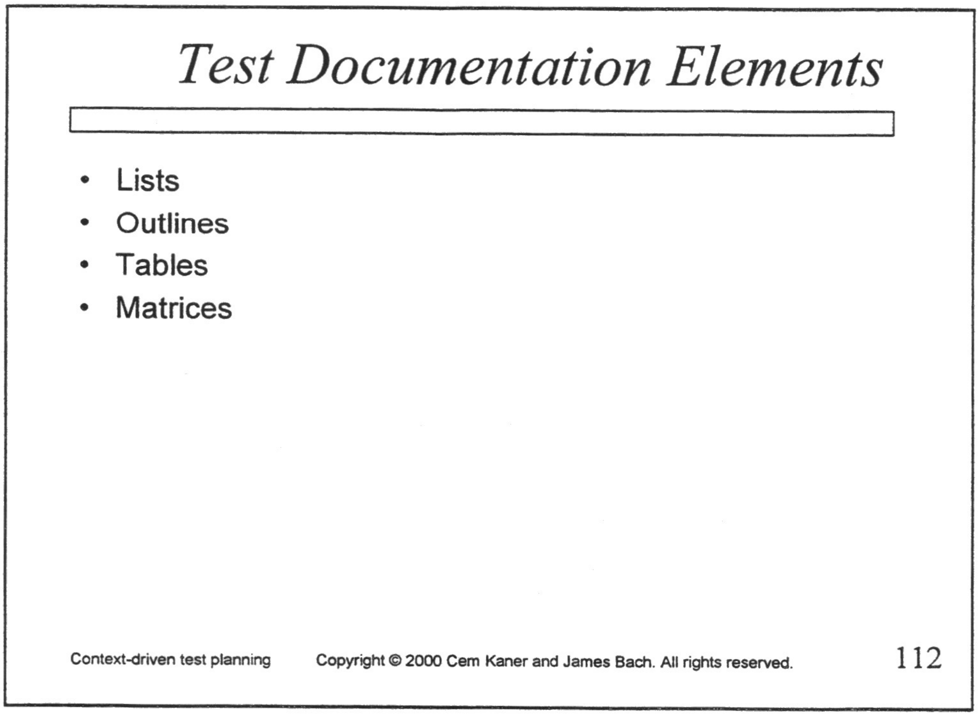
Context-driven test planning 110



Diverse Half-Measures

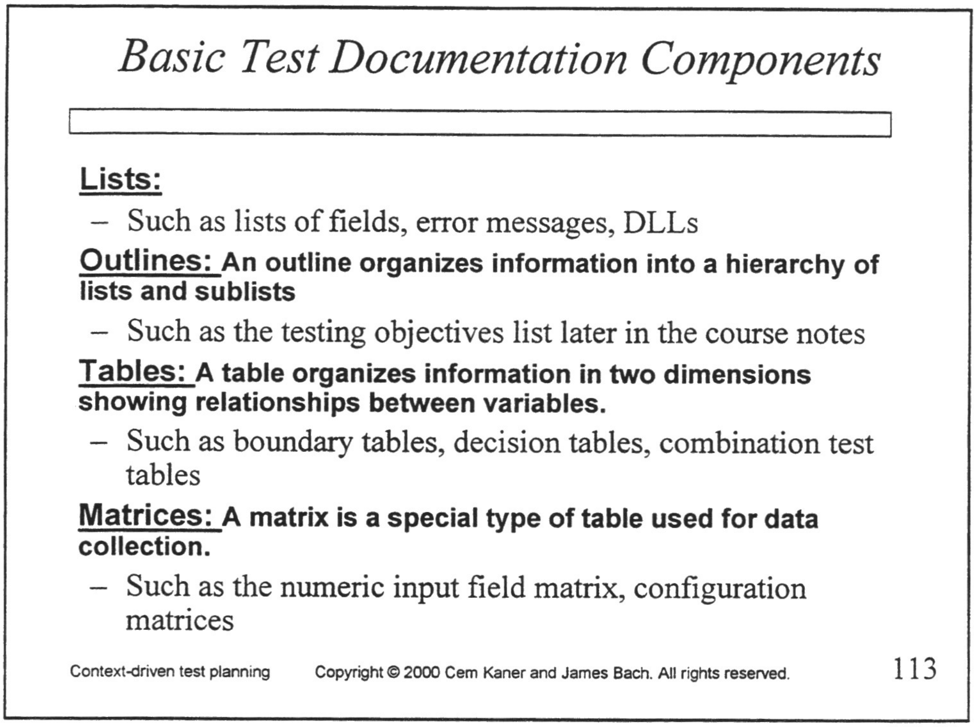
- There is no single technique that finds all bugs.
- We can't do any technique perfectly.
- We can't do all conceivable techniques.

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



Test Documentation Elements

- Lists
- Outlines
- Tables
- Matrices



Basic Test Documentation Components

Lists:

- Such as lists of fields, error messages, DLLs

Outlines: **An outline organizes information into a hierarchy of lists and sublists**

- Such as the testing objectives list later in the course notes

Tables: **A table organizes information in two dimensions showing relationships between variables.**

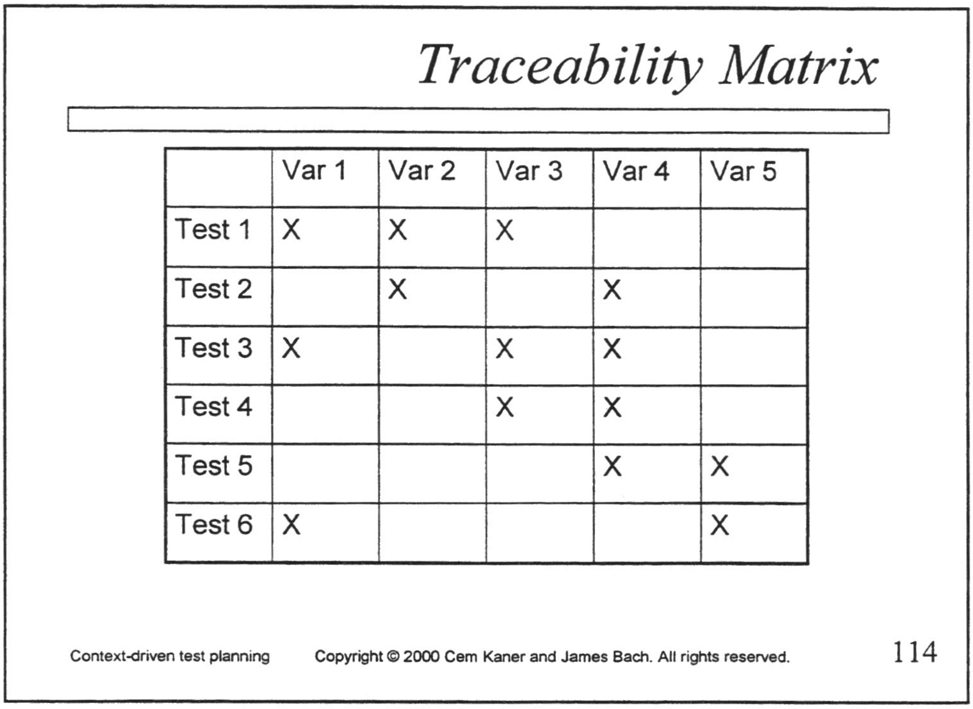
- Such as boundary tables, decision tables, combination test tables

Matrices: **A matrix is a special type of table used for data collection.**

- Such as the numeric input field matrix, configuration matrices

Context-driven test planning

113

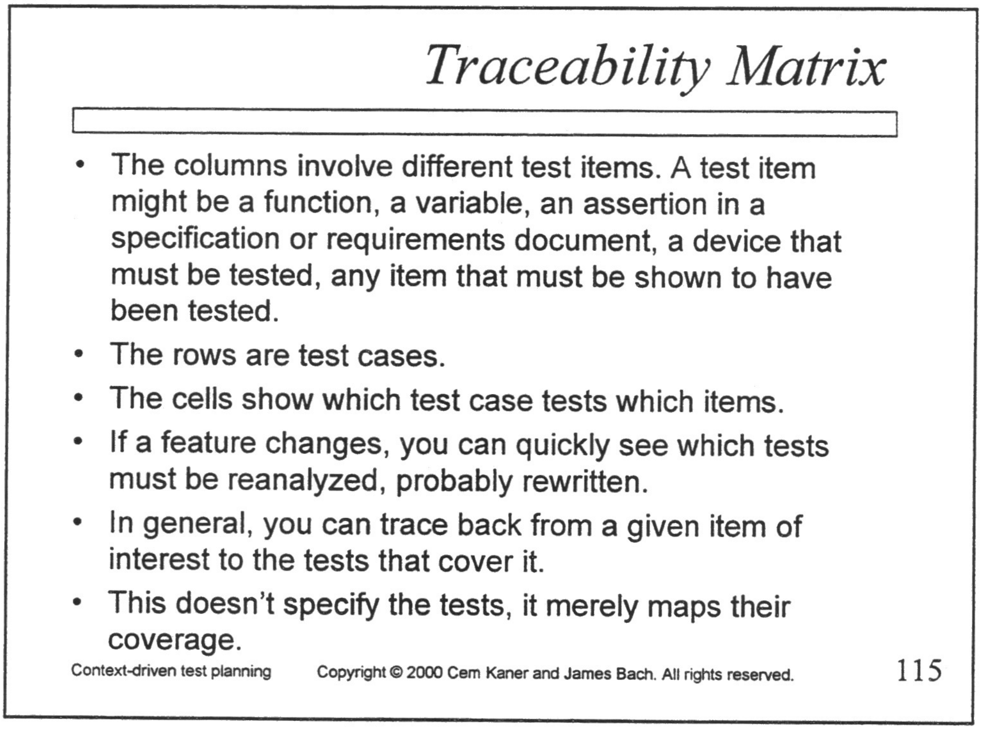


Traceability Matrix

	Var 1	Var 2	Var 3	Var 4	Var 5
Test 1	X	X	X		
Test 2		X		X	
Test 3	X		X	X	
Test 4			X	X	
Test 5				X	X
Test 6	X				X

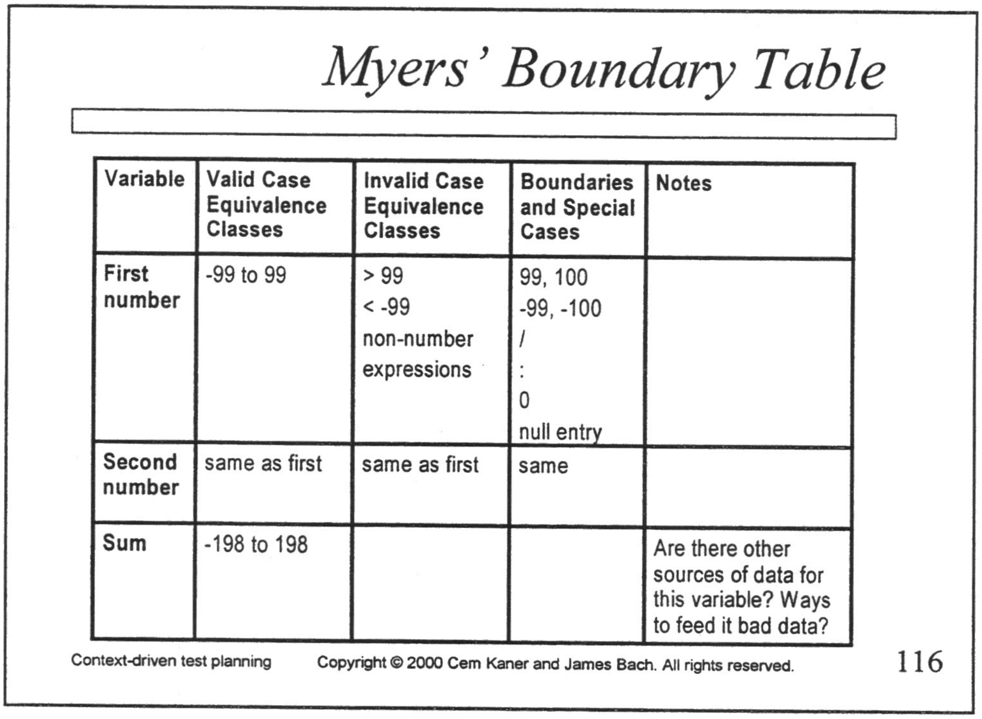
Context-driven test planning

114



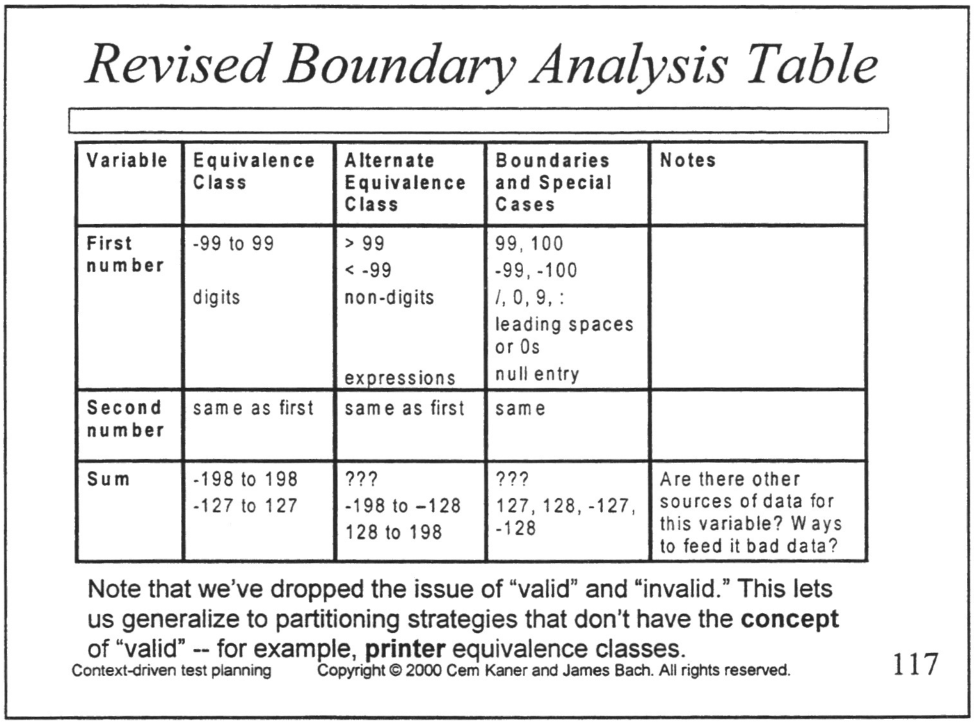
Traceability Matrix

- The columns involve different test items. A test item might be a function, a variable, an assertion in a specification or requirements document, a device that must be tested, any item that must be shown to have been tested.
- The rows are test cases.
- The cells show which test case tests which items.
- If a feature changes, you can quickly see which tests must be reanalyzed, probably rewritten.
- In general, you can trace back from a given item of interest to the tests that cover it.
- This doesn't specify the tests, it merely maps their coverage.



Myers' Boundary Table

Variable	Valid Case Equivalence Classes	Invalid Case Equivalence Classes	Boundaries and Special Cases	Notes
First number	-99 to 99	> 99 < -99 non-number expressions	99, 100 -99, -100 / : 0 null entry	
Second number	same as first	same as first	same	
Sum	-198 to 198			Are there other sources of data for this variable? Ways to feed it bad data?

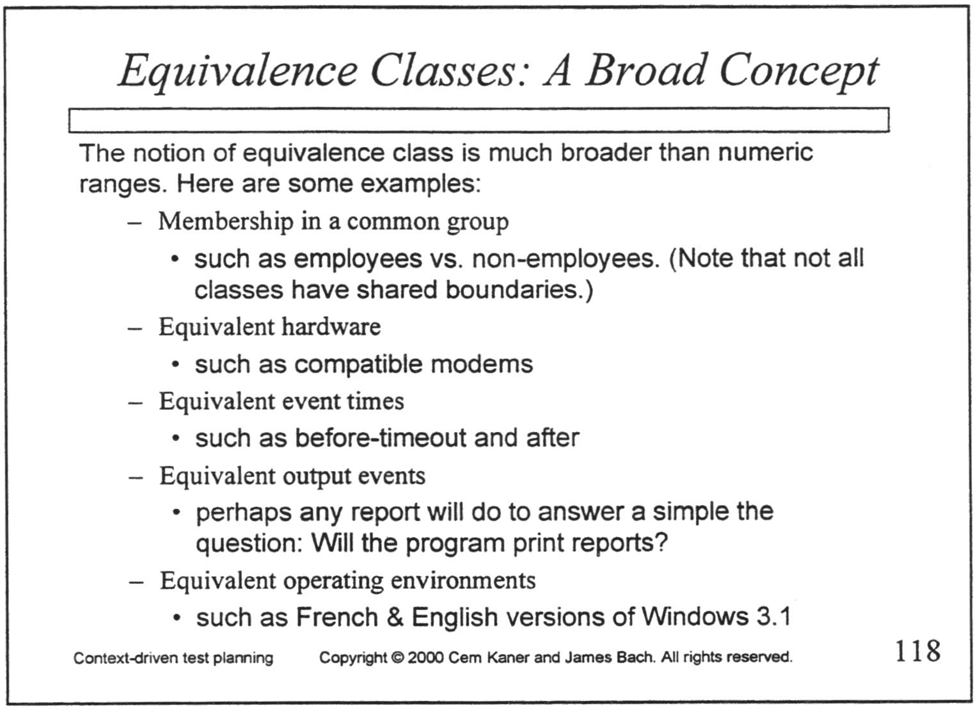


Revised Boundary Analysis Table

Variable	Equivalence Class	Alternate Equivalence Class	Boundaries and Special Cases	Notes
First number	-99 to 99 digits	> 99 < -99 non-digits expressions	99, 100 -99, -100 /, 0, 9, : leading spaces or 0s null entry	
Second number	same as first	same as first	same	
Sum	-198 to 198 -127 to 127	??? -198 to –128 128 to 198	??? 127, 128, -127, -128	Are there other sources of data for this variable? Ways to feed it bad data?

Note that we've dropped the issue of "valid" and "invalid." This lets us generalize to partitioning strategies that don't have the **concept** of "valid" -- for example, **printer** equivalence classes.

Context-driven test planning 117

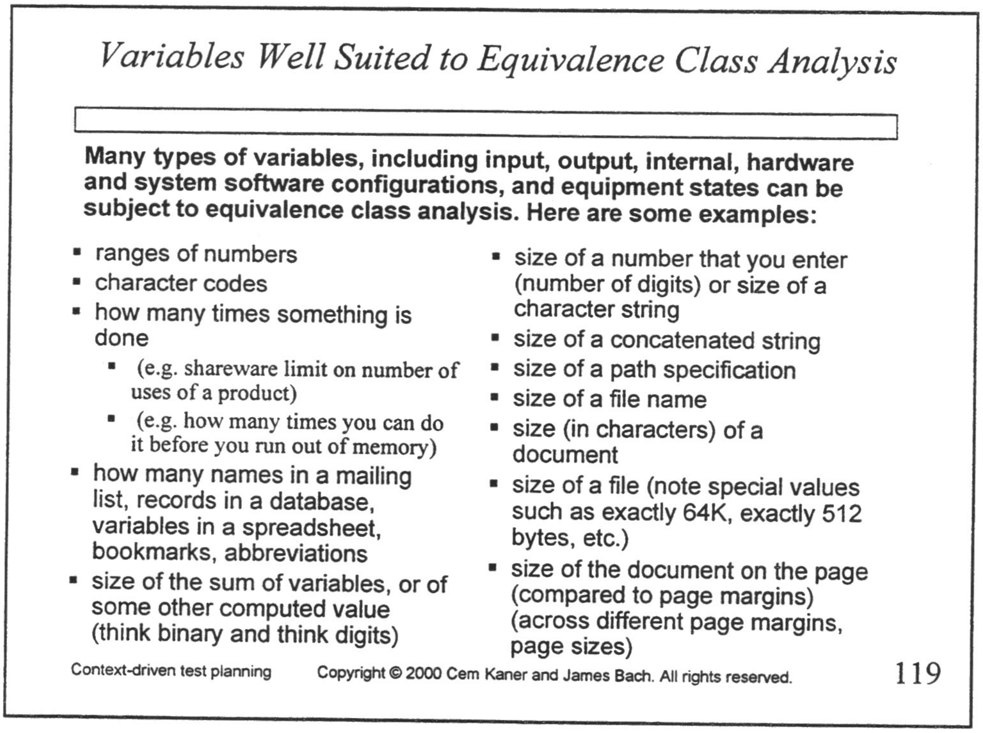


Equivalence Classes: A Broad Concept

The notion of equivalence class is much broader than numeric ranges. Here are some examples:

- Membership in a common group
  - such as employees vs. non-employees. (Note that not all classes have shared boundaries.)
- Equivalent hardware
  - such as compatible modems
- Equivalent event times
  - such as before-timeout and after
- Equivalent output events
  - perhaps any report will do to answer a simple the question: Will the program print reports?
- Equivalent operating environments
  - such as French & English versions of Windows 3.1

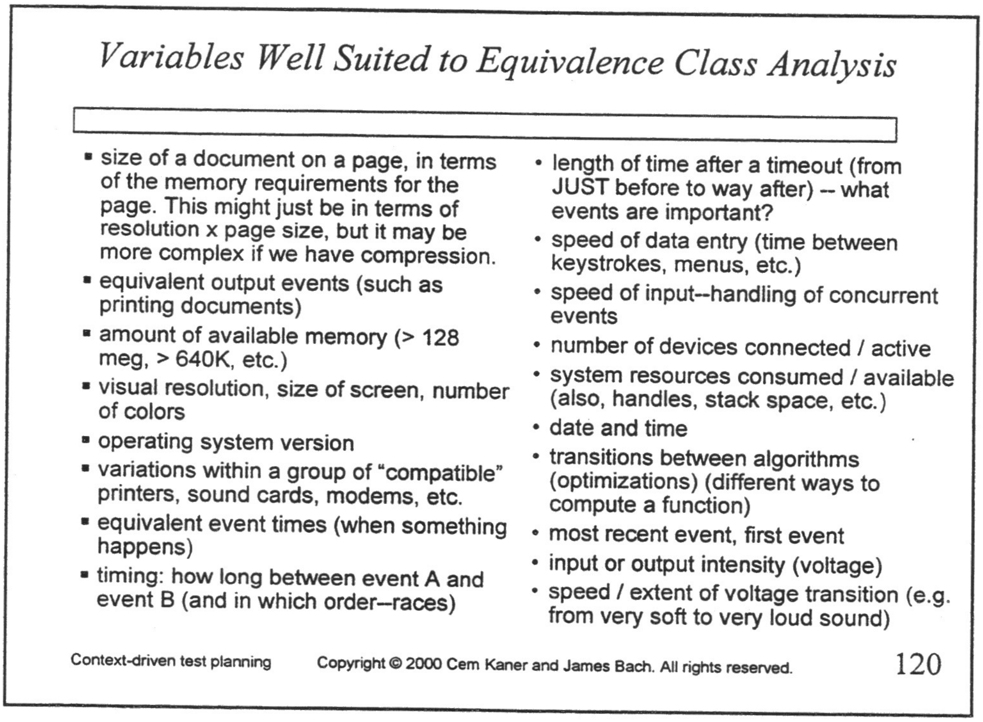
Context-driven test planning 118



Variables Well Suited to Equivalence Class Analysis

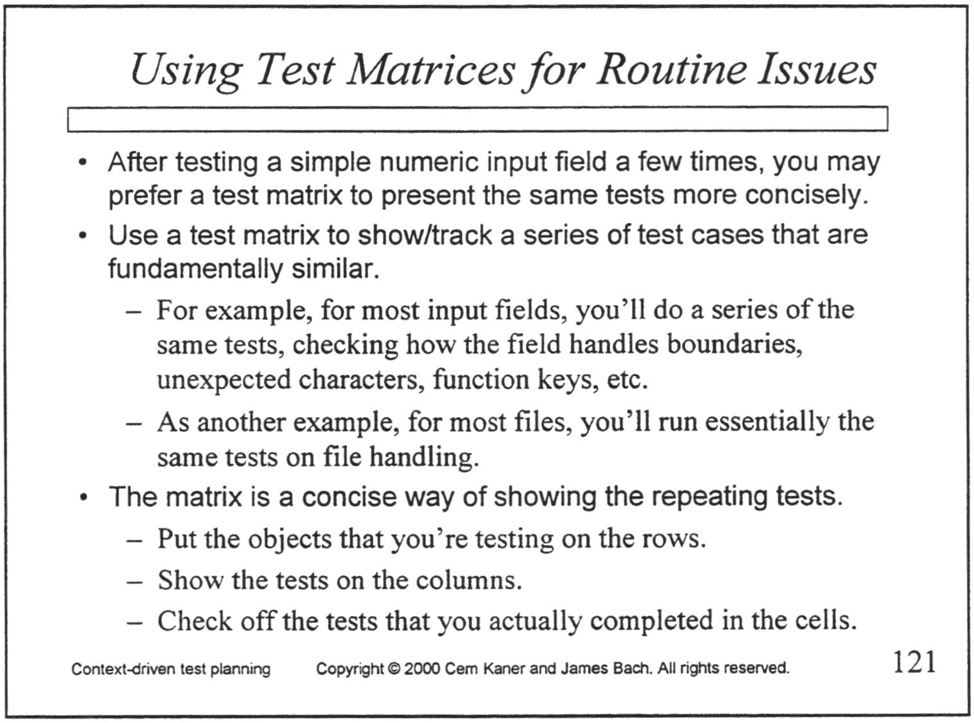
Many types of variables, including input, output, internal, hardware and system software configurations, and equipment states can be subject to equivalence class analysis. Here are some examples:

- ranges of numbers
- character codes
- how many times something is done
  - (e.g. shareware limit on number of uses of a product)
  - (e.g. how many times you can do it before you run out of memory)
- how many names in a mailing list, records in a database, variables in a spreadsheet, bookmarks, abbreviations
- size of the sum of variables, or of some other computed value (think binary and think digits)
- size of a number that you enter (number of digits) or size of a character string
- size of a concatenated string
- size of a path specification
- size of a file name
- size (in characters) of a document
- size of a file (note special values such as exactly 64K, exactly 512 bytes, etc.)
- size of the document on the page (compared to page margins) (across different page margins, page sizes)



Variables Well Suited to Equivalence Class Analysis

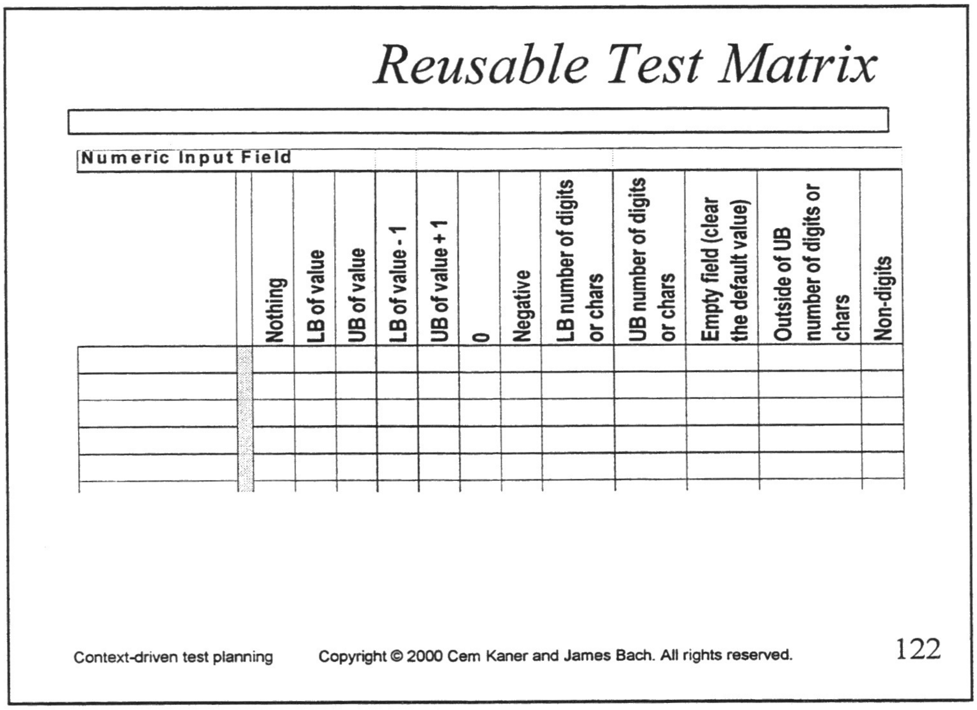
- size of a document on a page, in terms of the memory requirements for the page. This might just be in terms of resolution x page size, but it may be more complex if we have compression.
- equivalent output events (such as printing documents)
- amount of available memory (> 128 meg, > 640K, etc.)
- visual resolution, size of screen, number of colors
- operating system version
- variations within a group of "compatible" printers, sound cards, modems, etc.
- equivalent event times (when something happens)
- timing: how long between event A and event B (and in which order--races)
- length of time after a timeout (from JUST before to way after) -- what events are important?
- speed of data entry (time between keystrokes, menus, etc.)
- speed of input--handling of concurrent events
- number of devices connected / active
- system resources consumed / available (also, handles, stack space, etc.)
- date and time
- transitions between algorithms (optimizations) (different ways to compute a function)
- most recent event, first event
- input or output intensity (voltage)
- speed / extent of voltage transition (e.g. from very soft to very loud sound)



Using Test Matrices for Routine Issues

- After testing a simple numeric input field a few times, you may prefer a test matrix to present the same tests more concisely.
- Use a test matrix to show/track a series of test cases that are fundamentally similar.
  - For example, for most input fields, you'll do a series of the same tests, checking how the field handles boundaries, unexpected characters, function keys, etc.
  - As another example, for most files, you'll run essentially the same tests on file handling.
- The matrix is a concise way of showing the repeating tests.
  - Put the objects that you're testing on the rows.
  - Show the tests on the columns.
  - Check off the tests that you actually completed in the cells.

Context-driven test planning 121

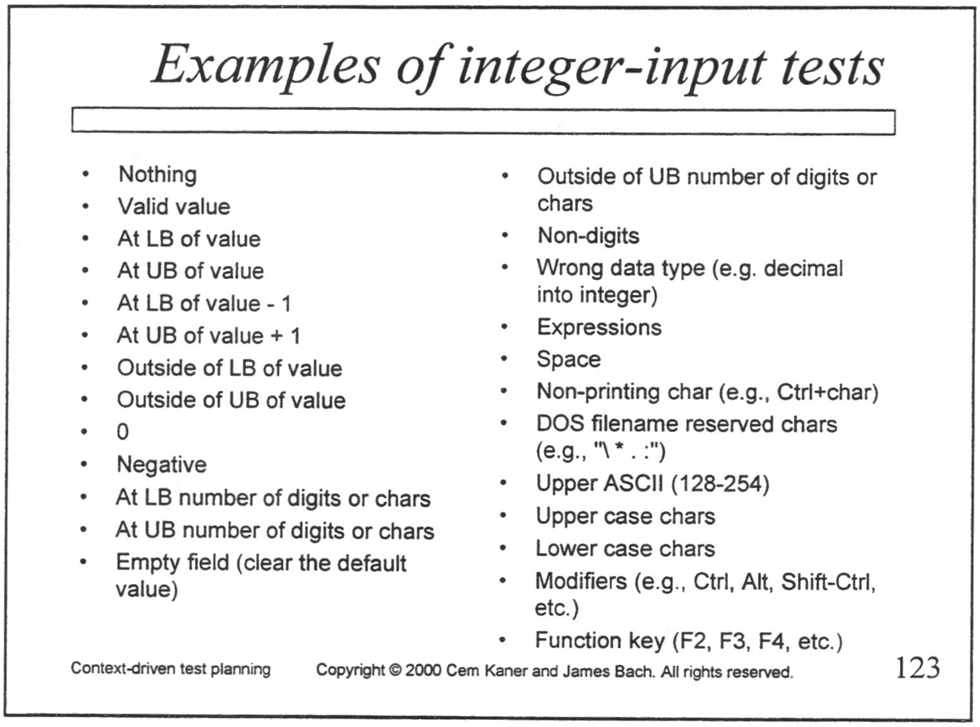


Reusable Test Matrix

Numeric Input Field

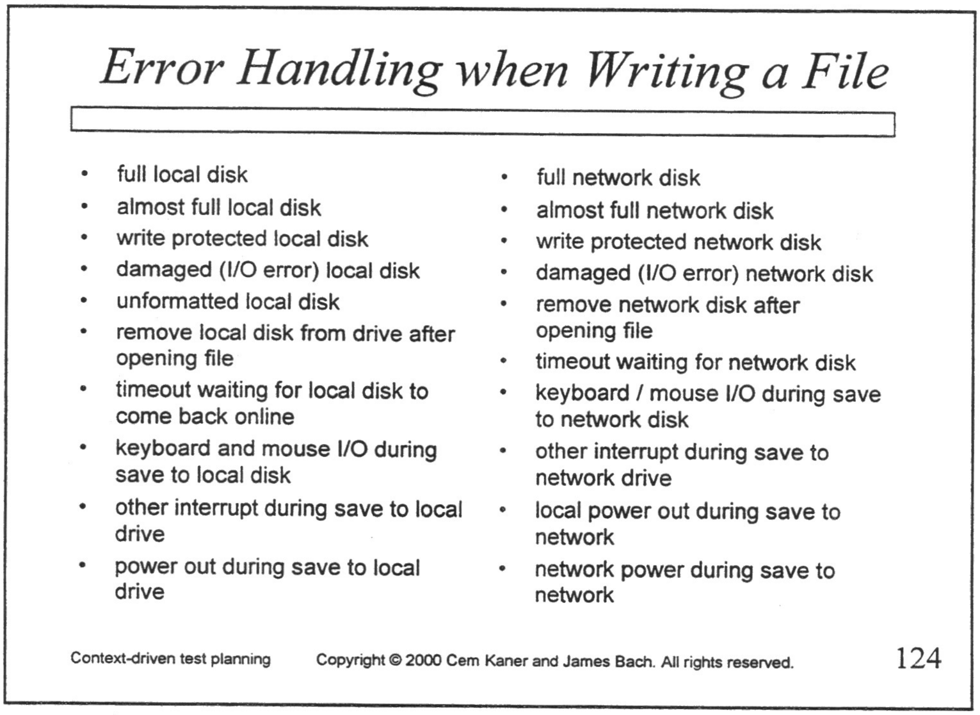
		Nothing	LB of value	UB of value	LB of value - 1	UB of value + 1	0	Negative	LB number of digits or chars	UB number of digits or chars	Empty field (clear the default value)	Outside of UB number of digits or chars	Non-digits

Context-driven test planning 122



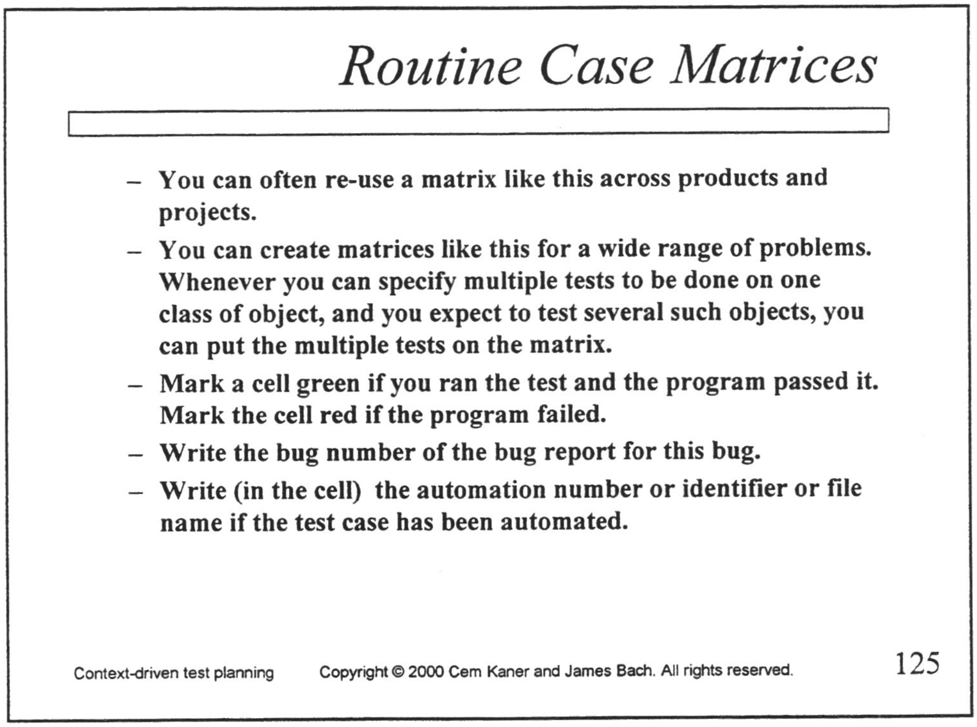
Examples of integer-input tests

- Nothing
- Valid value
- At LB of value
- At UB of value
- At LB of value - 1
- At UB of value + 1
- Outside of LB of value
- Outside of UB of value
- 0
- Negative
- At LB number of digits or chars
- At UB number of digits or chars
- Empty field (clear the default value)
- Outside of UB number of digits or chars
- Non-digits
- Wrong data type (e.g. decimal into integer)
- Expressions
- Space
- Non-printing char (e.g., Ctrl+char)
- DOS filename reserved chars (e.g., "\ * . :")
- Upper ASCII (128-254)
- Upper case chars
- Lower case chars
- Modifiers (e.g., Ctrl, Alt, Shift-Ctrl, etc.)
- Function key (F2, F3, F4, etc.)



Error Handling when Writing a File

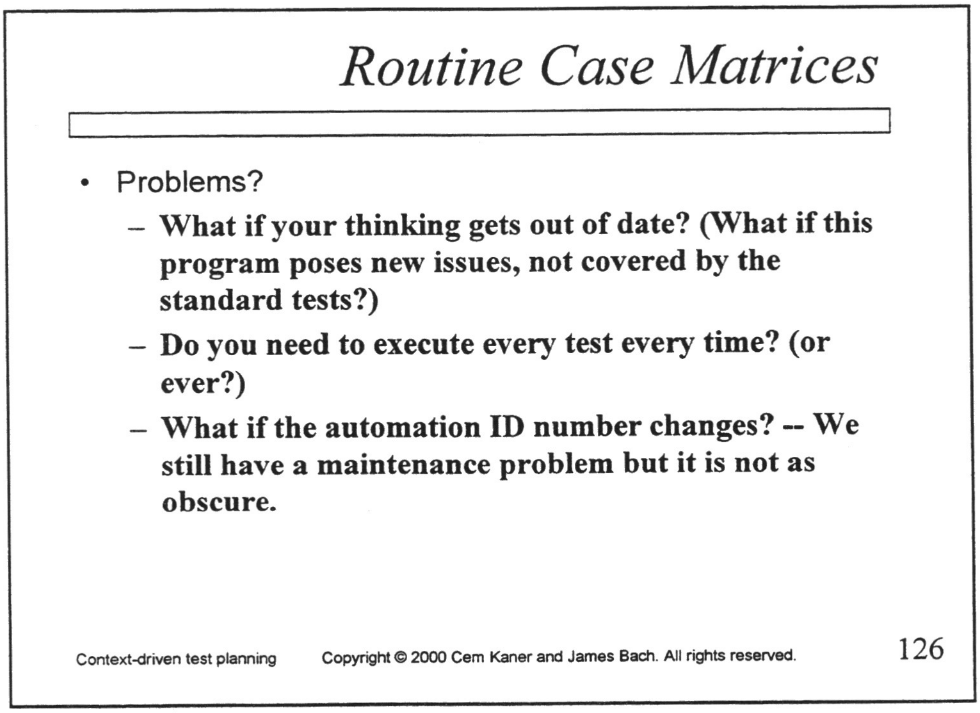
- full local disk
- almost full local disk
- write protected local disk
- damaged (I/O error) local disk
- unformatted local disk
- remove local disk from drive after opening file
- timeout waiting for local disk to come back online
- keyboard and mouse I/O during save to local disk
- other interrupt during save to local drive
- power out during save to local drive
- full network disk
- almost full network disk
- write protected network disk
- damaged (I/O error) network disk
- remove network disk after opening file
- timeout waiting for network disk
- keyboard / mouse I/O during save to network disk
- other interrupt during save to network drive
- local power out during save to network
- network power during save to network



Routine Case Matrices

- You can often re-use a matrix like this across products and projects.
- You can create matrices like this for a wide range of problems. Whenever you can specify multiple tests to be done on one class of object, and you expect to test several such objects, you can put the multiple tests on the matrix.
- Mark a cell green if you ran the test and the program passed it. Mark the cell red if the program failed.
- Write the bug number of the bug report for this bug.
- Write (in the cell) the automation number or identifier or file name if the test case has been automated.

Context-driven test planning 125

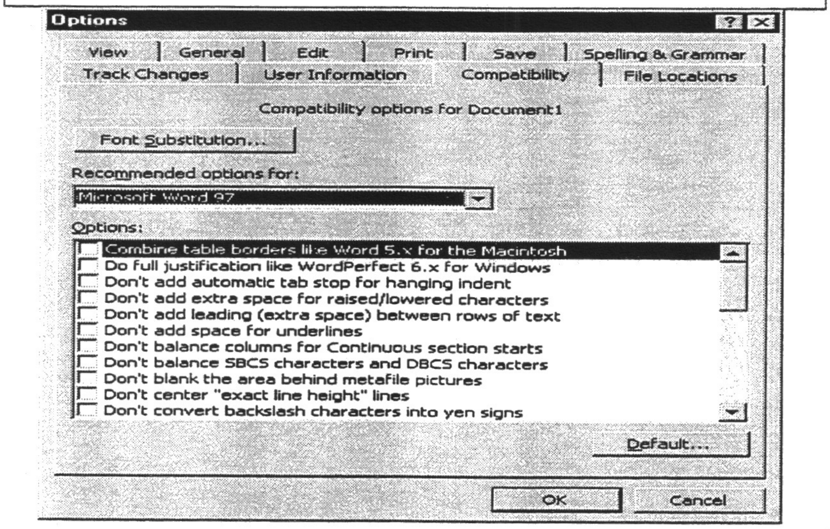


Routine Case Matrices

- Problems?
  - What if your thinking gets out of date? (What if this program poses new issues, not covered by the standard tests?)
  - Do you need to execute every test every time? (or ever?)
  - What if the automation ID number changes? -- We still have a maintenance problem but it is not as obscure.

Context-driven test planning 126

Complex Data Relationships



Compatibility options for Document1

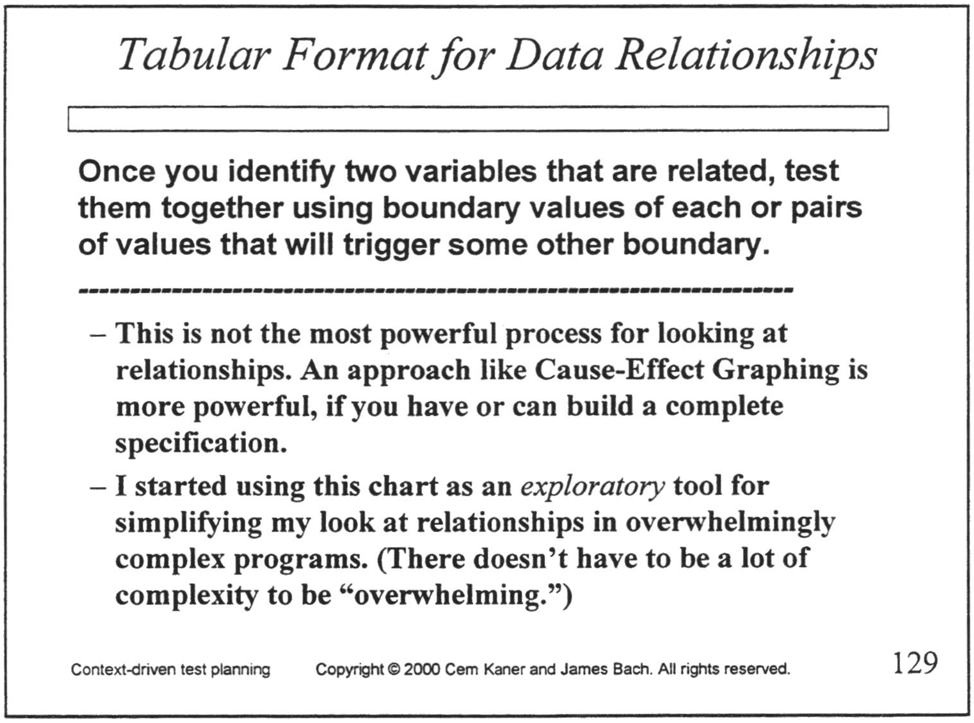
Recommended options for: Microsoft Word 97

Options:

- □ Combine table borders like Word 5.x for the Macintosh
- □ Do full justification like WordPerfect 6.x for Windows
- □ Don't add automatic tab stop for hanging indent
- □ Don't add extra space for raised/lowered characters
- □ Don't add leading (extra space) between rows of text
- □ Don't add space for underlines
- □ Don't balance columns for Continuous section starts
- □ Don't balance SBCS characters and DBCS characters
- □ Don't blank the area behind metafile pictures
- □ Don't center "exact line height" lines
- □ Don't convert backslash characters into yen signs

Tabular Format for Data Relationships

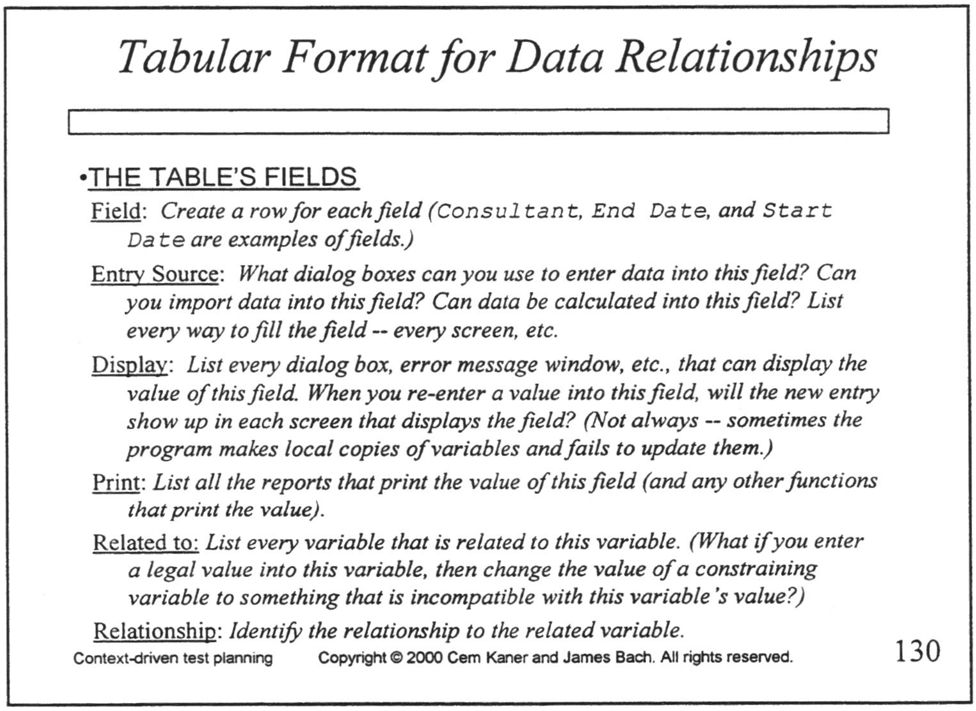
Field	Entry source	Display	Print	Related variable	Relationship



Tabular Format for Data Relationships

Once you identify two variables that are related, test them together using boundary values of each or pairs of values that will trigger some other boundary.

- **This is not the most powerful process for looking at relationships. An approach like Cause-Effect Graphing is more powerful, if you have or can build a complete specification.**
- **I started using this chart as an** *exploratory* **tool for simplifying my look at relationships in overwhelmingly complex programs. (There doesn't have to be a lot of complexity to be "overwhelming.")**



Tabular Format for Data Relationships

•THE TABLE'S FIELDS

Field: *Create a row for each field (`Consultant`, `End Date`, and `Start Date` are examples of fields.)*

Entry Source: *What dialog boxes can you use to enter data into this field? Can you import data into this field? Can data be calculated into this field? List every way to fill the field -- every screen, etc.*

Display: *List every dialog box, error message window, etc., that can display the value of this field. When you re-enter a value into this field, will the new entry show up in each screen that displays the field? (Not always -- sometimes the program makes local copies of variables and fails to update them.)*

Print: *List all the reports that print the value of this field (and any other functions that print the value).*

Related to: *List every variable that is related to this variable. (What if you enter a legal value into this variable, then change the value of a constraining variable to something that is incompatible with this variable's value?)*

Relationship: *Identify the relationship to the related variable.*

Tabular Format for Data Relationships

Many relationships among data:

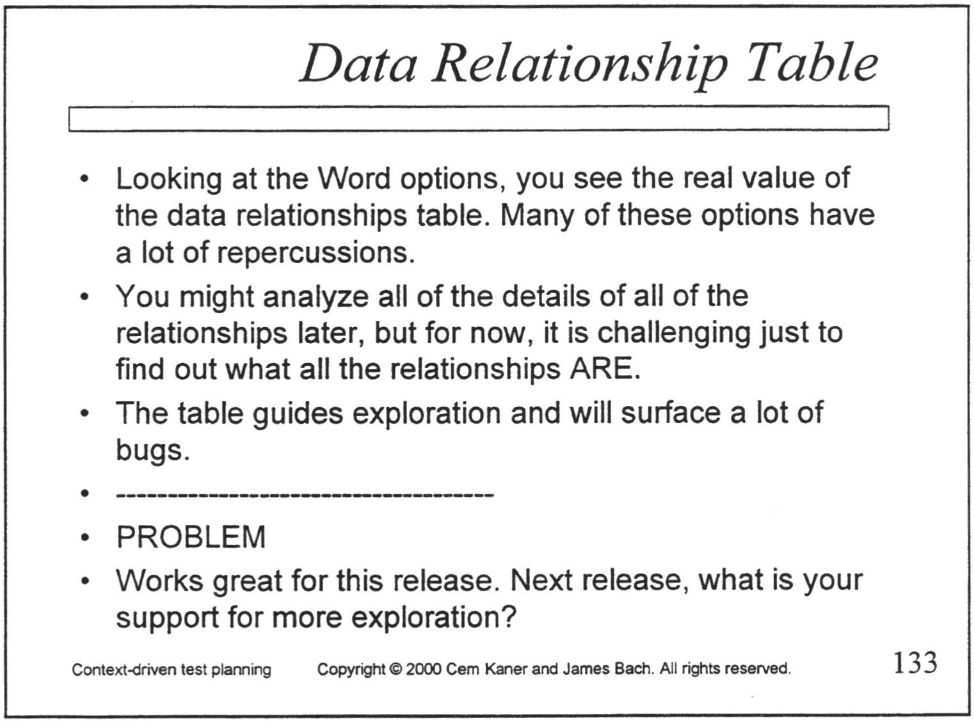
- **Independence**
  - **Varying one has no effect on the value or permissible values of the other.**
- **Causal determination**
  - **By changing the value of one, we determine the value of the other.**
  - **For example, in MS Word, the extent of shading of an area depends on the object selected. The shading differs depending on Table vs. Paragraph.**
- **Constrained to a range**
  - **For example, the width of a line has to be less than the width of the page.**
  - **In a date field, the permissible dates are determined by the month (and the year, if February).**
- **Selection of rules**
  - **Example, hyphenation rules depend on the language you choose.**

Tabular Format for Data Relationships

Many relationships among data:

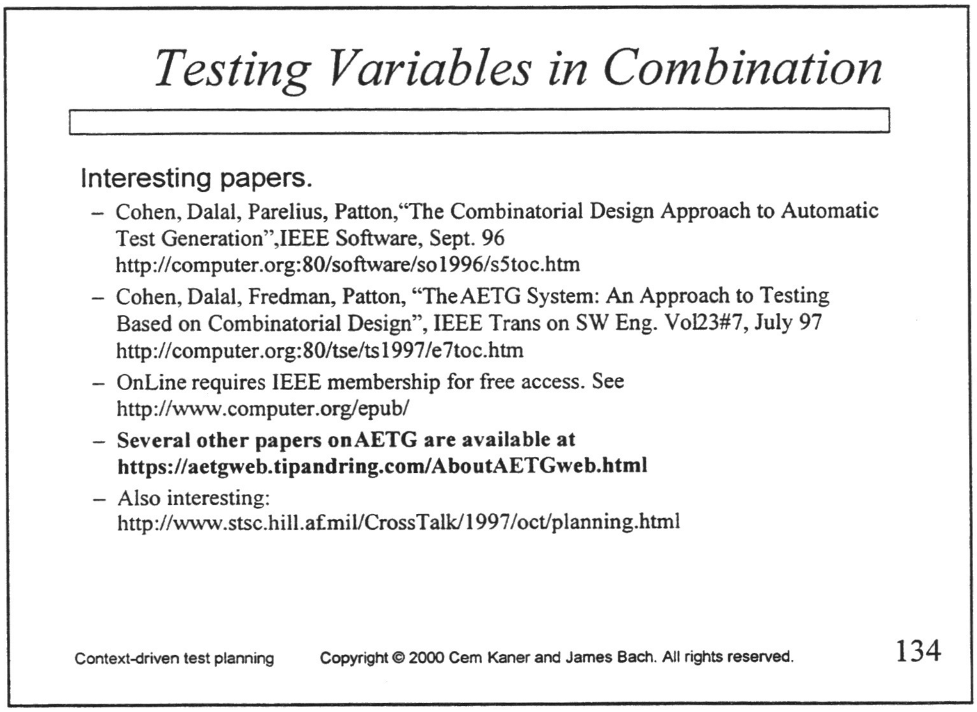
- **Logical selection from a list**
  - **processes the value you entered and then figures out what value to use for the next variable. Example: timeouts in phone dialing:**
    - **0 on complete call 555-1212 but 95551212?**
    - **10 on ambiguous completion, 955-5121**
    - **30 seconds incomplete 555-121**
- **Logical selection *of* a list:**
  - **For example, in printer setup, choose:**
    - **<u>OfficeJet,</u> get Graphics Quality, Paper Type, and Color Options**
    - **<u>LaserJet 4</u>, get Economode, Resolution, and Half-toning.**

Look at Marick *(Craft of Software Testing)* for discussion of catalogs of tests for data relationships.



Data Relationship Table

- Looking at the Word options, you see the real value of the data relationships table. Many of these options have a lot of repercussions.
- You might analyze all of the details of all of the relationships later, but for now, it is challenging just to find out what all the relationships ARE.
- The table guides exploration and will surface a lot of bugs.
- -----------------------------------
- PROBLEM
- Works great for this release. Next release, what is your support for more exploration?



Testing Variables in Combination

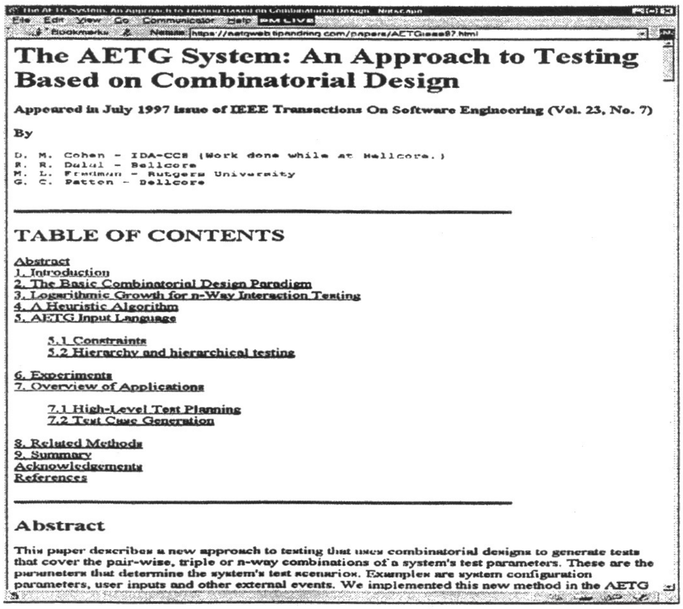
Interesting papers.

- Cohen, Dalal, Parelius, Patton,“The Combinatorial Design Approach to Automatic Test Generation”,IEEE Software, Sept. 96 http://computer.org:80/software/so1996/s5toc.htm
- Cohen, Dalal, Fredman, Patton, “TheAETG System: An Approach to Testing Based on Combinatorial Design”, IEEE Trans on SW Eng. Vol23#7, July 97 http://computer.org:80/tse/ts1997/e7toc.htm
- OnLine requires IEEE membership for free access. See http://www.computer.org/epub/
- **Several other papers onAETG are available at https://aetgweb.tipandring.com/AboutAETGweb.html**
- Also interesting: http://www.stsc.hill.af.mil/CrossTalk/1997/oct/planning.html

Combination Chart

	Var 1	Var 2	Var 3	Var 4	Var 5
Test 1	Value 11	Value 12	Value 13	Value 14	Value 15
Test 2	Value 21	Value 22	Value 23	Value 24	Value 25
Test 3	Value 31	Value 32	Value 33	Value 34	Value 35
Test 4	Value 41	Value 42	Value 43	Value 44	Value 45
Test 5	Value 51	Value 52	Value 53	Value 54	Value 55
Test 6	Value 61	Value 62	Value 63	Value 64	Value 65

Context-driven test planning 135


The AETG System: An Approach to Testing Based on Combinatorial Design - Netscape
File Edit View Go Communicator Help PM LIVE
Bookmarks
Netsite: https://aetgweb.tipandring.com/papers/AETGieee97.html

The AETG System: An Approach to Testing Based on Combinatorial Design

Appeared in July 1997 issue of IEEE Transactions On Software Engineering (Vol. 23, No. 7)

By

```
D. M. Cohen - IDA-CCS (Work done while at Bellcore.)
S. R. Dalal - Bellcore
M. L. Fredman - Rutgers University
G. C. Patton - Bellcore
```

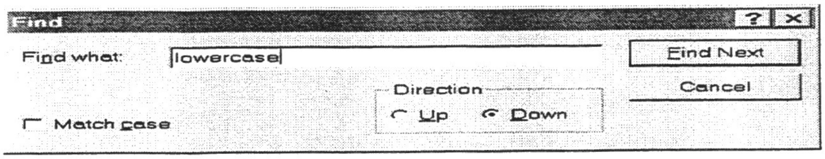
TABLE OF CONTENTS

Abstract

This paper describes a new approach to testing that uses combinatorial designs to generate tests that cover the pair-wise, triple or n-way combinations of a system's test parameters. These are the parameters that determine the system's test scenarios. Examples are system configuration parameters, user inputs and other external events. We implemented this new method in the AETG

136

Combinations Exercise / Illustration



Find what: lowercase

Direction ○ Up ● Down

□ Match case

- Here is a simple Find dialog. It takes three inputs:
  - `Find what:` a text string
  - `Match case:` yes or no
  - `Direction:` up or down
- Simplify this by considering only three values for the text string, "lowercase" and "Mixed Cases" and "CAPITALS".
- (Note: To do a better job, we'd also choose input documents that would yield a "find" and a "don't find" for each case. The input document would be another variable or, really, the intended result (Find / Don't) would be the variable. We'll think about that again after the exercise.)

Combinations Exercise

1 How many combinations of these three variables are possible?

2 List ALL the combinations of these three variables.

3 Now create combination tests that cover all possible pairs of values, but don't try to cover all possible triplets. List one such set.

4 How many test cases are in this set?

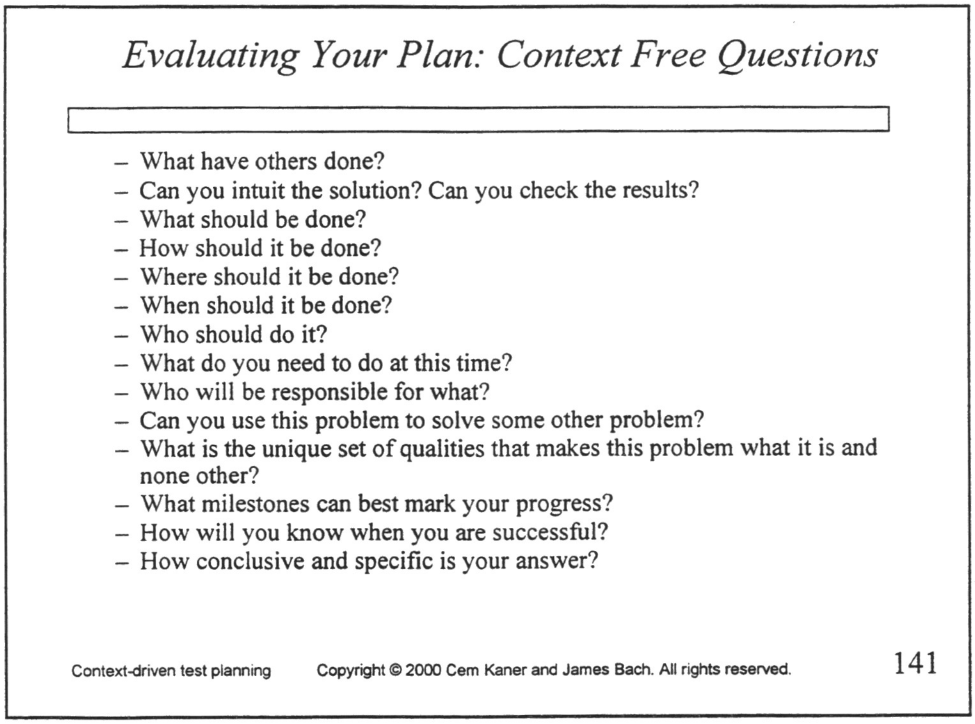
Charts: References

You can find plenty of example charts in Bill Perry's books, such as Effective Methods for Software Testing (2nd Ed., Wiley). Several of these will probably be useful, though (like the charts in these notes) you'll have to adapt them to your circumstances.

Evaluating Your Plan: Context Free Questions

Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140) and Bach's Evaluation Strategies (Rapid Testing Course notes)*

– Can you solve the whole problem? Part of the problem?
– What would you like the resolution to be? Can you picture it?
– How much of the unknown can you determine?
– What reference data are you using (if any)?
– What product output will you evaluate?
– How will you do the evaluation?
– Can you derive something useful from the information you have?
– Have you used all the information?
– Have you taken into account all essential notions in the problem?
– Can you separate the steps in the problem-solving process? Can you determine the correctness of each step?
– What creative thinking techniques can you use to generate ideas? How many different techniques?
– Can you see the result? How many different kinds of results can you see?
– How many different ways have you tried to solve the problem?



Evaluating Your Plan: Context Free Questions

- What have others done?
- Can you intuit the solution? Can you check the results?
- What should be done?
- How should it be done?
- Where should it be done?
- When should it be done?
- Who should do it?
- What do you need to do at this time?
- Who will be responsible for what?
- Can you use this problem to solve some other problem?
- What is the unique set of qualities that makes this problem what it is and none other?
- What milestones can best mark your progress?
- How will you know when you are successful?
- How conclusive and specific is your answer?

Context-driven test planning

141

QW2001 Workshop W2

Mr. John Paul
(Minjoh Technology Solutions)

Automating Software Testing: A Life-Cycle Methodology

Key Points

- Best practices for test automation
- A Case Study will be presented that covers how the ATLM was implemented on one particular project. This case study will address each phases of the Automated Testing Life-cycle. Students can bring their own project specific problems, which can be addressed during the tutorial.
- Acquiring management support
- Test tool evaluation and selection
- The automated testing introduction process
- Various tools used during the various life-cycle phases
- Automated and manual test planning and preparation
- Test procedure development guidelines
- Automation reuse analysis and reuse library

Presentation Abstract

Automating Software Testing: A Life-Cycle Methodology
This tutorial outlines the Automated Test Life-cycle Methodology, a structured process for designing, developing, executing and managing testing that parallels the System Development Life-cycle. It is based on the book titled "Automated Software Testing" co-authored by the instructor and published by AWL, ISBN 0-201-43287-0. Automated Testing Life-Cycle Methodology

How test teams introduce an automated software test tool on a new project is nearly as important as the selection of the most appropriate test tool for the project. A tool is only as good as the process being used to implement the tool. Over the last several years test teams have largely implemented automated testing tools on projects, without having a process or strategy in place describing in detail the steps involved in using the test tool productively. This approach commonly results in the development of test scripts that are not reusable, meaning that the test script serves a single test string but cannot be applied to a subsequent release of the software application. In the case of incremental software builds and as a result of software changes, these test scripts need to be recreated repeatedly and must be adjusted multiple times to accommodate minor software changes. This approach increases the testing effort and brings subsequent schedule increases and cost overruns.

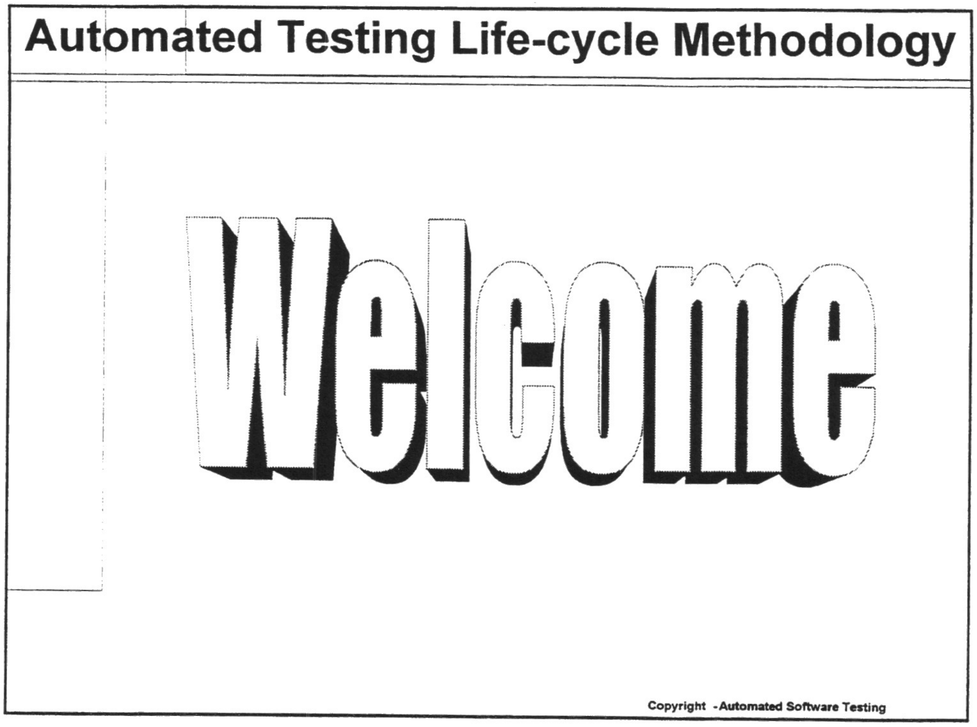
The fallout from a bad experience with a test tool on a project can have a ripple effect throughout an organization. The experience may tarnish the reputation of the test group. Confidence in the tool by product and project managers may have been shaken to the point where the test team may have difficulty obtaining approval for use of a test tool on future efforts. Likewise, when budget pressures materialize, planned expenditures for test tool licenses and related tool support may be scratched.

By developing and following a strategy for rolling out and implement an automated test tool as part of the Automated Testing Life-cycle methodology, the test team can avoid having to make major unplanned adjustments throughout the test process. The tutorial "Automated Software Testing" addresses these various issues and their solutions. The ATLM describes how and where "Automated Software Testing" fits into the system development life-cycle.

About the Author

Founder and President of Minjoh Technology Solutions,Inc (www.minjohtech.com), a company dedicated to providing quality Information Technology Solutions to government and commercial agencies. He has a BS degree in Computer Science and 12 years of professional software development experience. His experience extends through all phases of software development life cycle. He has been a speaker/presenter at various professional seminars and conferences.

He is also a co-author of the best-selling book "Automated Software Testing", published by Addison-Wesley Pub Co; ISBN: 0201432870, July 1999. The book has been getting excellent reviews throughout the testing community (see also www.amazon.com). The book has now been translated into German and is available in German. The book is currently being translated into Japanese and should be available in Japanese middle of 2001. For other details about the book see website at www.autotestco.com.



Automated Testing Life-cycle Methodology

Welcome

Copyright -Automated Software Testing

by

John Paul

President

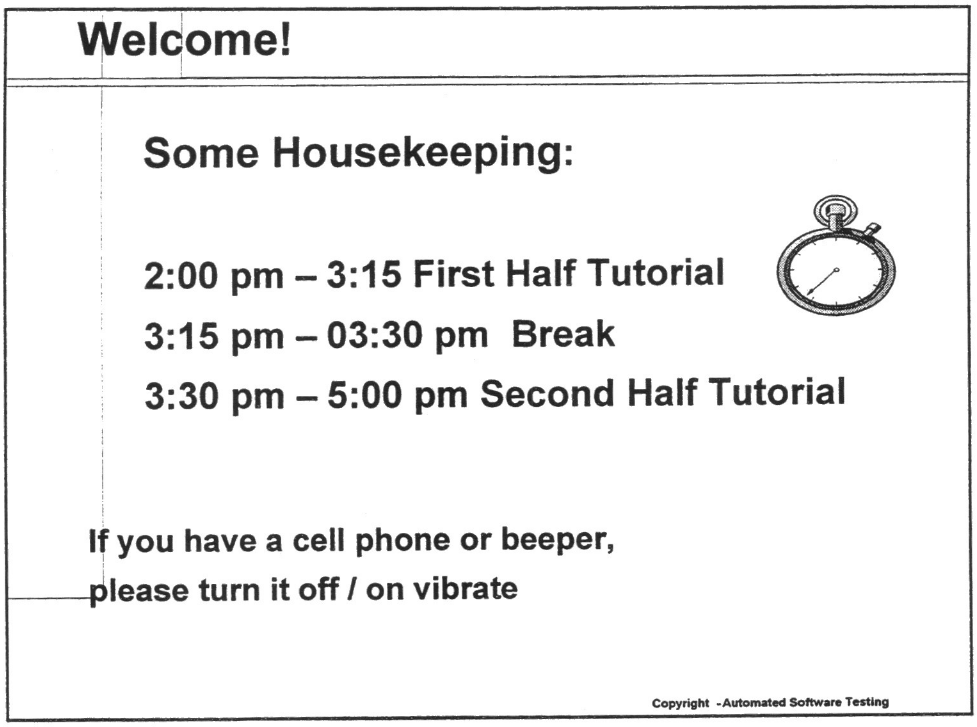
Minjoh Technology Solutions, Inc.

http://www.minjohtech.com

Email: johnpaul@minjohtech.com

Copyright -Automated Software Testing

Workshop 2



Welcome!

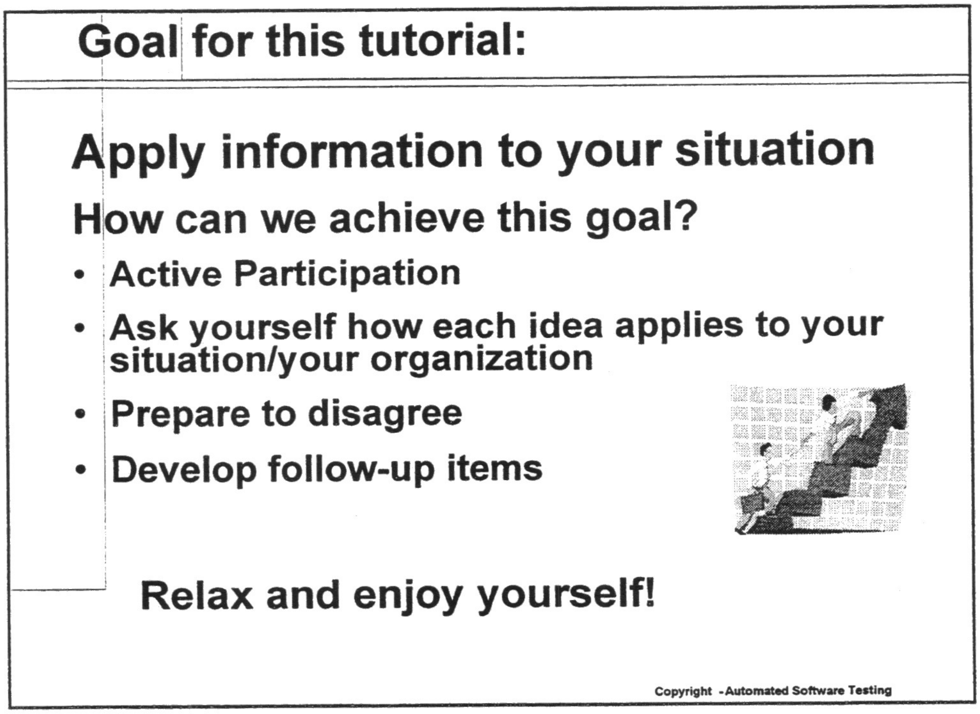
Some Housekeeping:

2:00 pm – 3:15 First Half Tutorial

3:15 pm – 03:30 pm Break

3:30 pm – 5:00 pm Second Half Tutorial

If you have a cell phone or beeper,
please turn it off / on vibrate



Goal for this tutorial:

Apply information to your situation

How can we achieve this goal?

- Active Participation
- Ask yourself how each idea applies to your situation/your organization
- Prepare to disagree
- Develop follow-up items

Relax and enjoy yourself!

Book: Automated Software Testing

Automated Software Testing

Introduction, Management, and Performance

ELFRIEDE DUSTIN
JEFF RASHKA
JOHN PAUL

CD-ROM includes Automated Test Life Cycle Management (ARM)

Book give away at the end of class....

http://www.autotestco.com

Quick Audience Survey:

What are you trying to get out of this tutorial?

What is your goal?

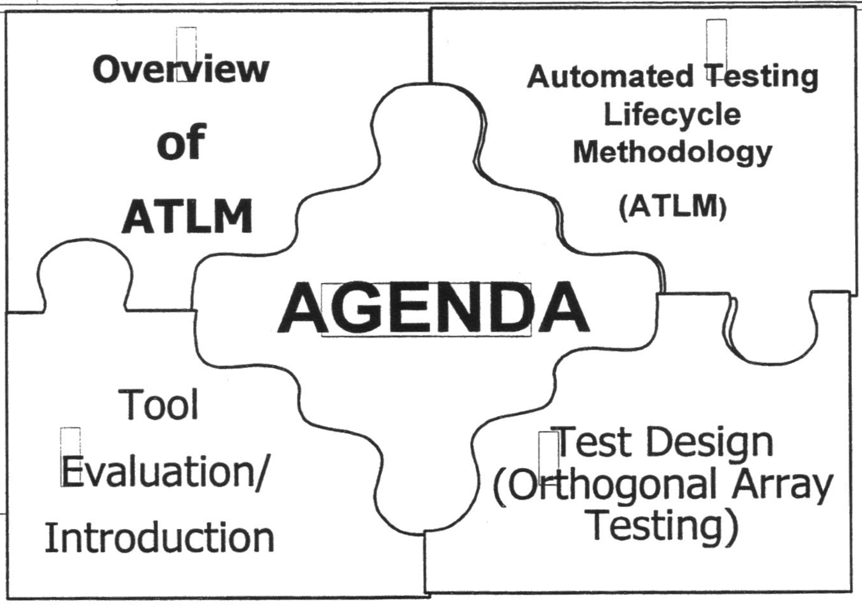
Explain in 30 seconds or less..

Quick Audience Survey:

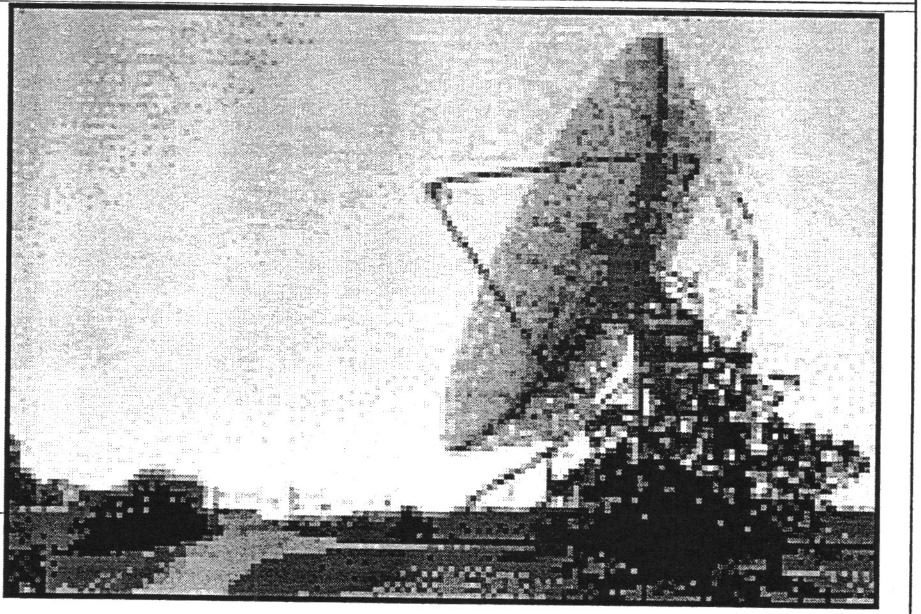
Show of Hands:

- Use of Automated Tools? Y/N
- Mercury, Segue, Rational, RSW, Compuware, other?
- Environment – Windows, Unix, Mainframe, other?
- Test Experience – in years?

AGENDA


Overview of ATLM
Automated Testing Lifecycle Methodology (ATLM)
AGENDA
Tool Evaluation/ Introduction
Test Design (Orthogonal Array Testing)

NASA Gives Up Search for Lost Mars Lander



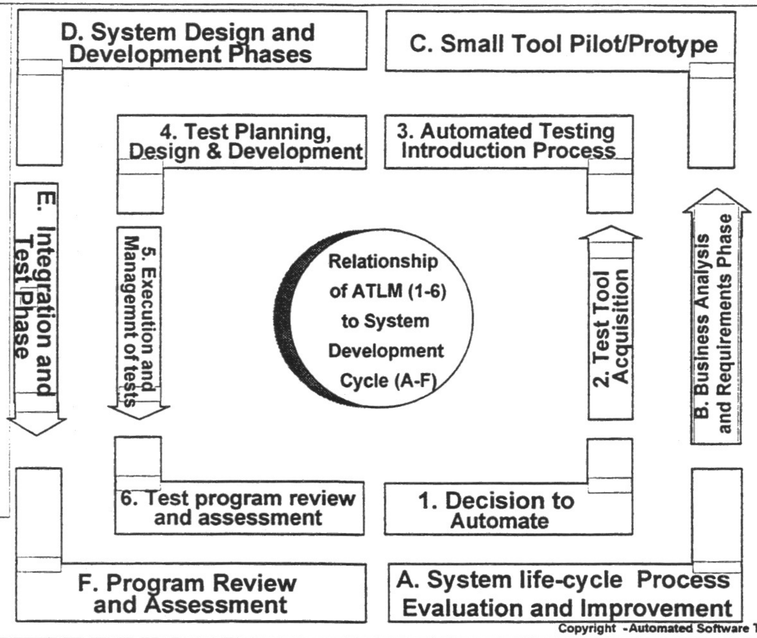
Copyright - Automated Software Testing

Message:

Defects are serious!

Copyright - Automated Software Testing

Automated Testing Life-cycle Methodology (ATLM)



A. System life-cycle Process Evaluation and Improvement

B. Business Analysis and Requirements Phase

C. Small Tool Pilot/Protype

D. System Design and Development Phases

E. Integration and Test Phase

F. Program Review and Assessment

1. Decision to Automate

2. Test Tool Acquisition

3. Automated Testing Introduction Process

4. Test Planning, Design & Development

5. Execution and Managemnt of tests

6. Test program review and assessment

Relationship of ATLM (1-6) to System Development Cycle (A-F)

ATLM and Software Development

1. Decision to Automate

2. Test Tool Acquisition

3. Automated Testing Introduction Process

4. Test Planning Design & Development

5. Execution and Management of Test

6. Process Evolution and Improvement

A. System Life-cycle Process Evaluation and Improvement

B. Business Analysis and Requirements Phase

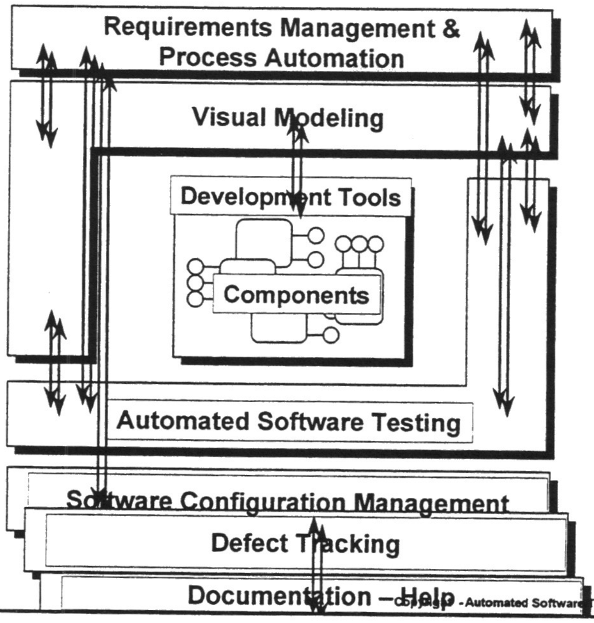
C. Small Tool Pilot/Prototype

D. System Design and Development Phases

E. Integration and Test Phase

F. Program Review and Assessment

Automated Testing spans the entire Lifecycle:



Requirements Management & Process Automation

Visual Modeling

Development Tools

Components

Automated Software Testing

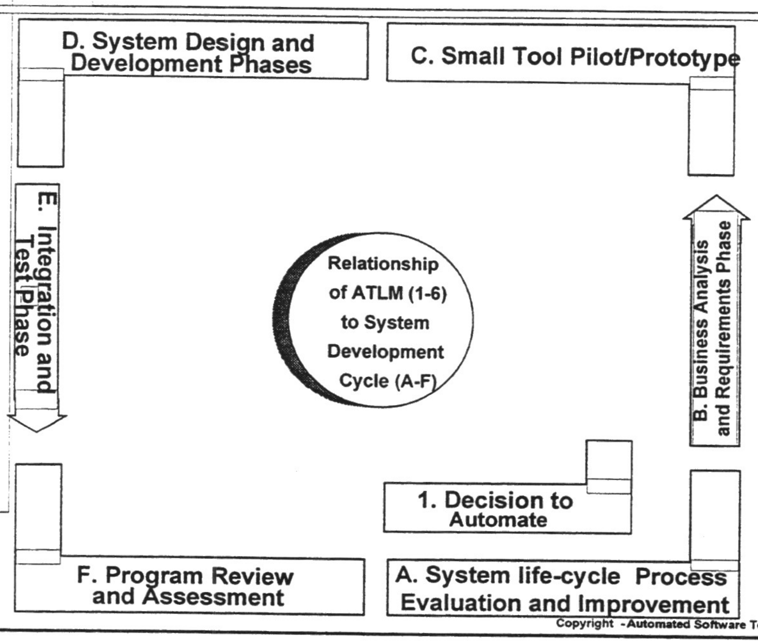
Software Configuration Management

Defect Tracking

Documentation – Help

Copyright -Automated Software Testing

Automated Testing Life-cycle Methodology (ATLM)



D. System Design and Development Phases

C. Small Tool Pilot/Prototype

E. Integration and Test Phase

Relationship of ATLM (1-6) to System Development Cycle (A-F)

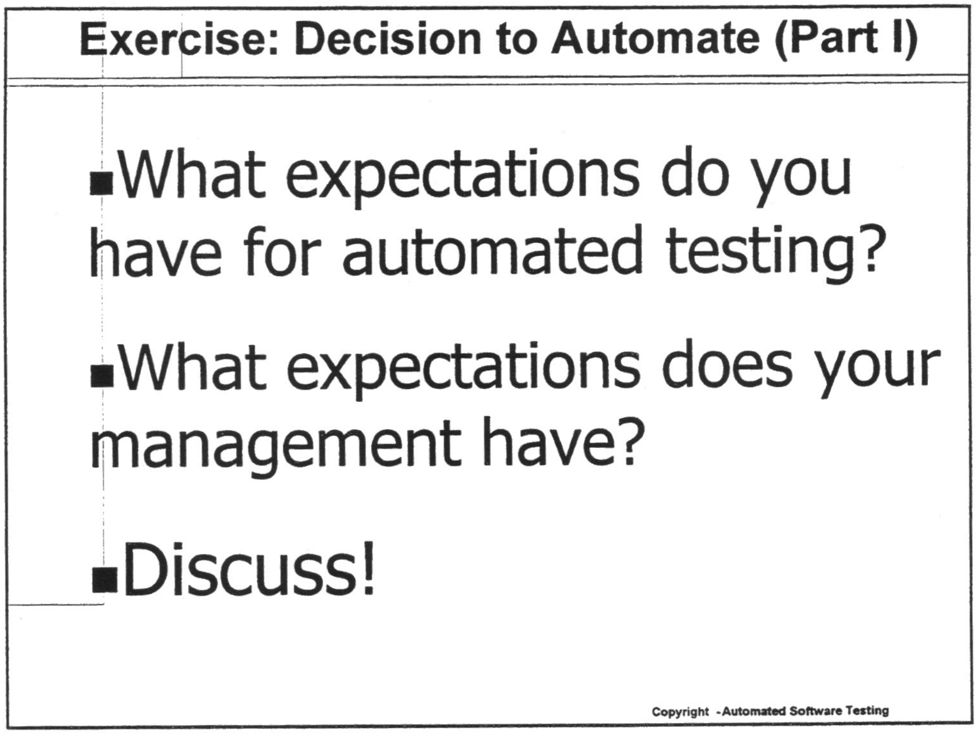
B. Business Analysis and Requirements Phase

1. Decision to Automate

F. Program Review and Assessment

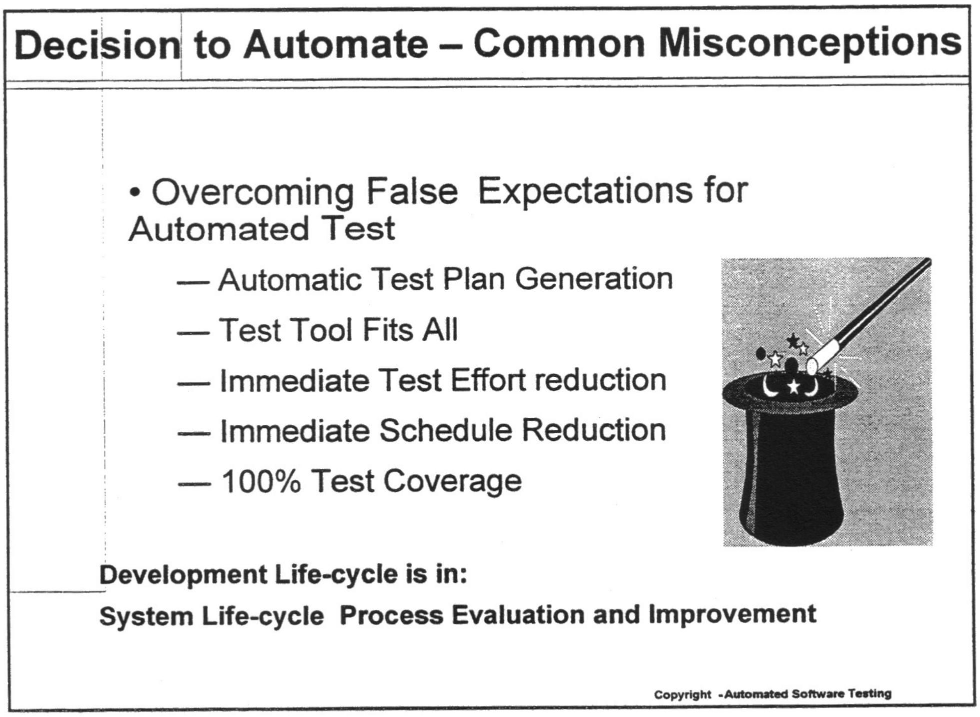
A. System life-cycle Process Evaluation and Improvement

Copyright -Automated Software Testing



Exercise: Decision to Automate (Part I)

- What expectations do you have for automated testing?
- What expectations does your management have?
- Discuss!

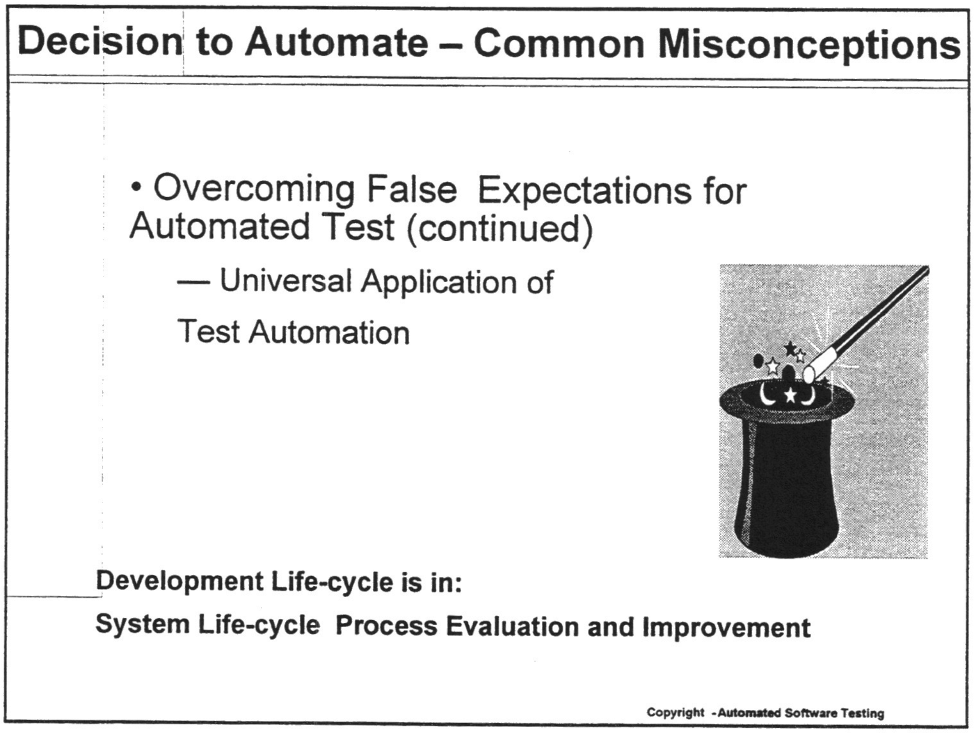


Decision to Automate – Common Misconceptions

- Overcoming False Expectations for Automated Test
  - — Automatic Test Plan Generation
  - — Test Tool Fits All
  - — Immediate Test Effort reduction
  - — Immediate Schedule Reduction
  - — 100% Test Coverage

Development Life-cycle is in:

System Life-cycle Process Evaluation and Improvement



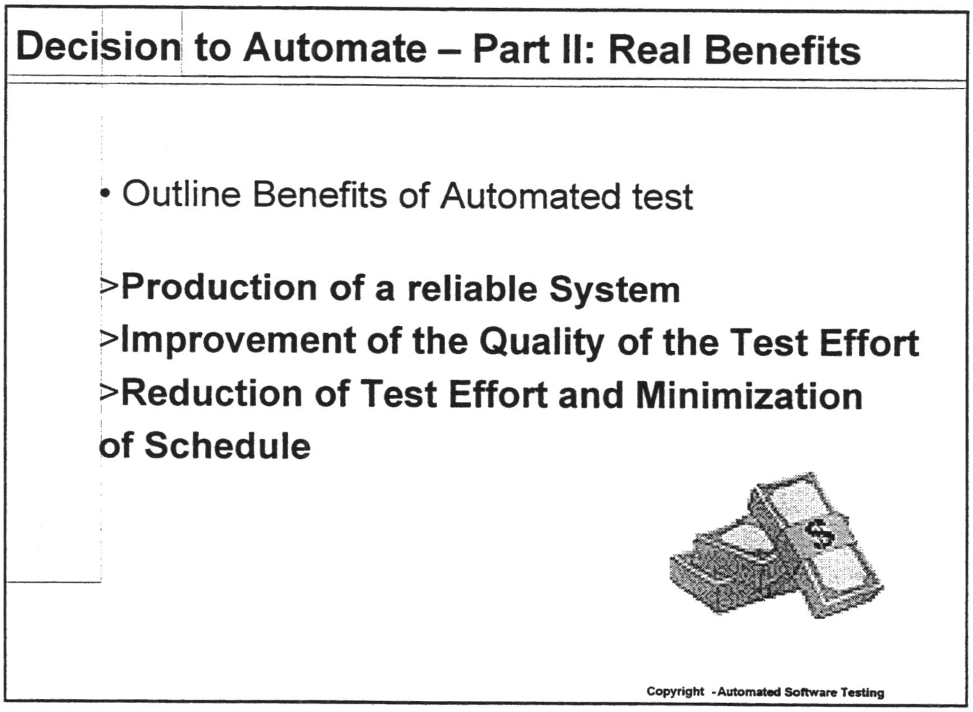
Decision to Automate – Common Misconceptions

- Overcoming False Expectations for Automated Test (continued)
  - — Universal Application of Test Automation

Development Life-cycle is in:

System Life-cycle Process Evaluation and Improvement

Copyright -Automated Software Testing

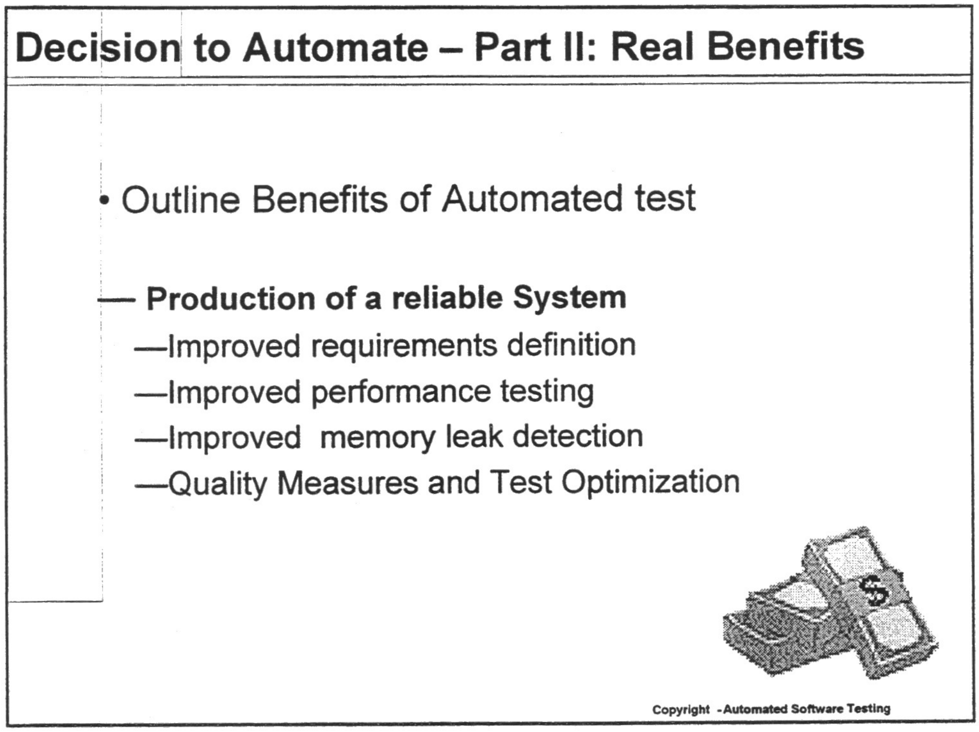


Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

>Production of a reliable System
>Improvement of the Quality of the Test Effort
>Reduction of Test Effort and Minimization of Schedule

Copyright -Automated Software Testing



Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

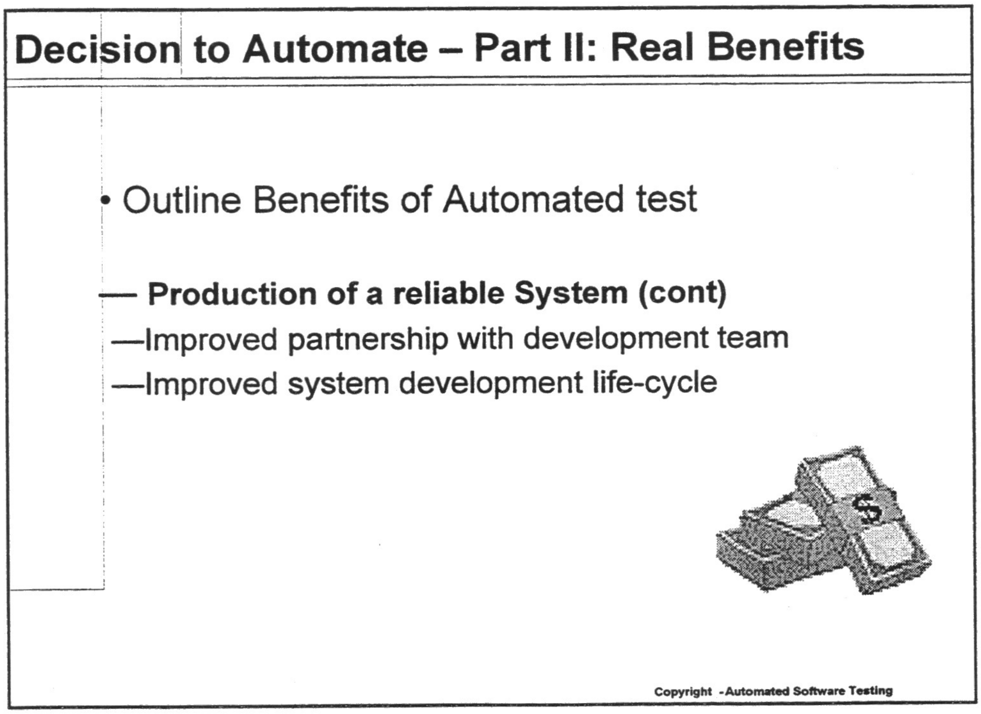
— **Production of a reliable System**

—Improved requirements definition

—Improved performance testing

—Improved memory leak detection

—Quality Measures and Test Optimization



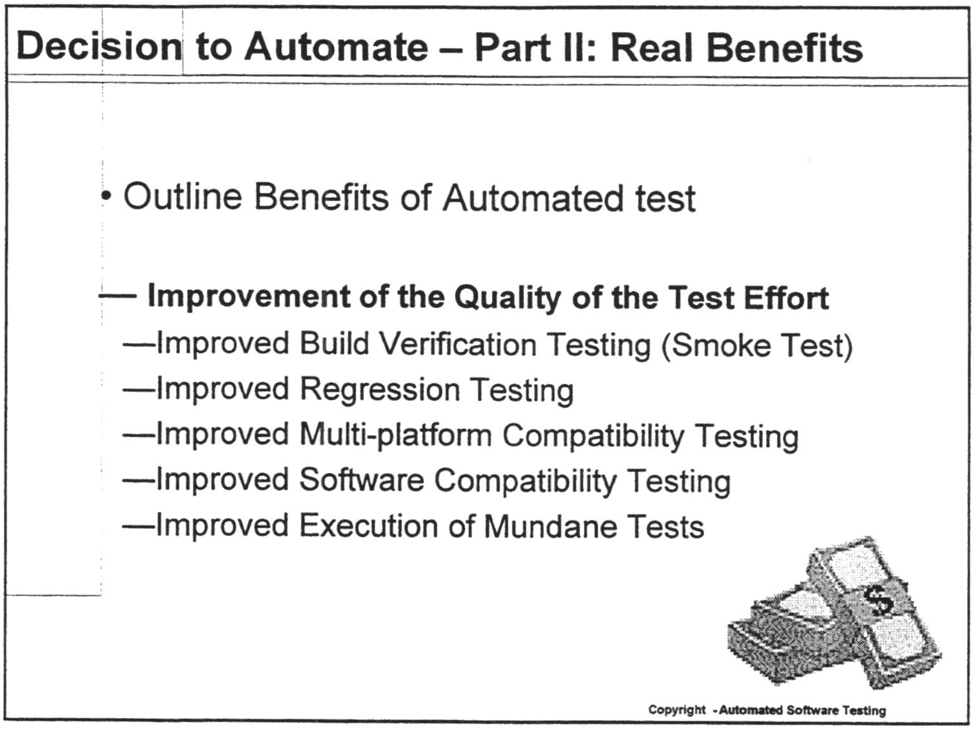
Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

— **Production of a reliable System (cont)**

—Improved partnership with development team

—Improved system development life-cycle



Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

— Improvement of the Quality of the Test Effort

—Improved Build Verification Testing (Smoke Test)

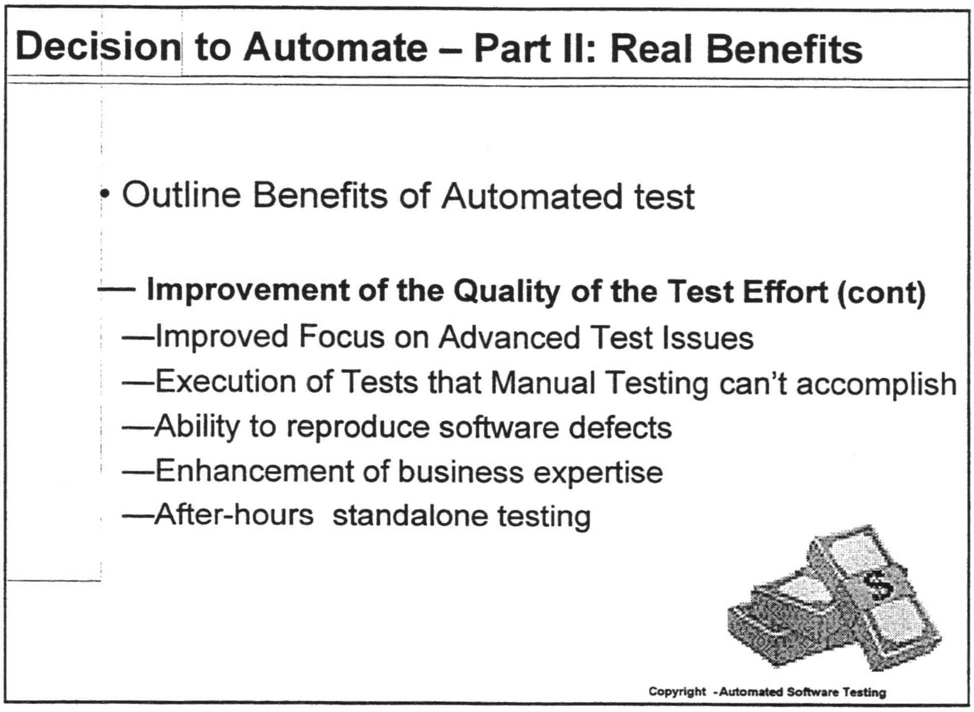
—Improved Regression Testing

—Improved Multi-platform Compatibility Testing

—Improved Software Compatibility Testing

—Improved Execution of Mundane Tests

Copyright -Automated Software Testing



Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

— Improvement of the Quality of the Test Effort (cont)

—Improved Focus on Advanced Test Issues

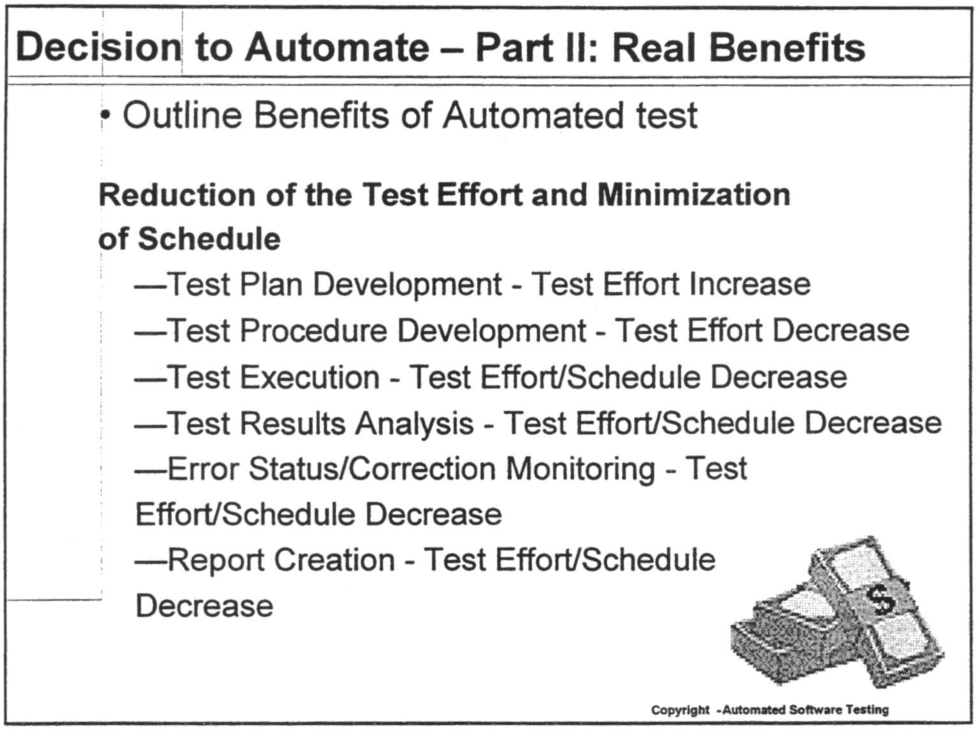
—Execution of Tests that Manual Testing can't accomplish

—Ability to reproduce software defects

—Enhancement of business expertise

—After-hours standalone testing

Copyright -Automated Software Testing



Decision to Automate – Part II: Real Benefits

- Outline Benefits of Automated test

Reduction of the Test Effort and Minimization of Schedule

—Test Plan Development - Test Effort Increase

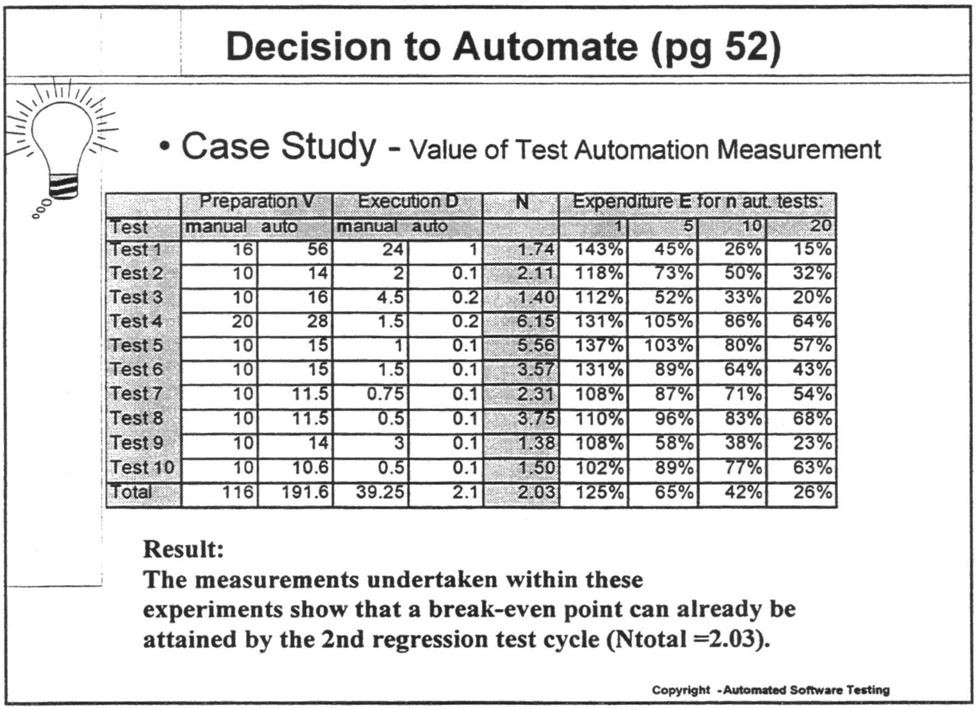
—Test Procedure Development - Test Effort Decrease

—Test Execution - Test Effort/Schedule Decrease

—Test Results Analysis - Test Effort/Schedule Decrease

—Error Status/Correction Monitoring - Test Effort/Schedule Decrease

—Report Creation - Test Effort/Schedule Decrease



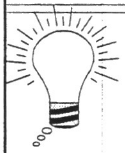
Decision to Automate (pg 52)

- Case Study - Value of Test Automation Measurement

	Preparation V		Execution D		N	Expenditure E for n aut. tests:			
Test	manual	auto	manual	auto		1	5	10	20
Test 1	16	56	24	1	1.74	143%	45%	26%	15%
Test 2	10	14	2	0.1	2.11	118%	73%	50%	32%
Test 3	10	16	4.5	0.2	1.40	112%	52%	33%	20%
Test 4	20	28	1.5	0.2	6.15	131%	105%	86%	64%
Test 5	10	15	1	0.1	5.56	137%	103%	80%	57%
Test 6	10	15	1.5	0.1	3.57	131%	89%	64%	43%
Test 7	10	11.5	0.75	0.1	2.31	108%	87%	71%	54%
Test 8	10	11.5	0.5	0.1	3.75	110%	96%	83%	68%
Test 9	10	14	3	0.1	1.38	108%	58%	38%	23%
Test 10	10	10.6	0.5	0.1	1.50	102%	89%	77%	63%
Total	116	191.6	39.25	2.1	2.03	125%	65%	42%	26%

Result:
The measurements undertaken within these experiments show that a break-even point can already be attained by the 2nd regression test cycle (Ntotal =2.03).

Decision to Automate (pg 52)



•Value of Test Automation (after the fact)

Value of automated testing also has to be measured based on
- Bugs found
- Bugs not found – because the testers were too busy automating

Decision to Automate (cont'd)

- Outline Benefits of Automated test

— Production of a reliable System
— Improvement of the quality of effort
— Reduction of Test Effort and Minimization of Schedule

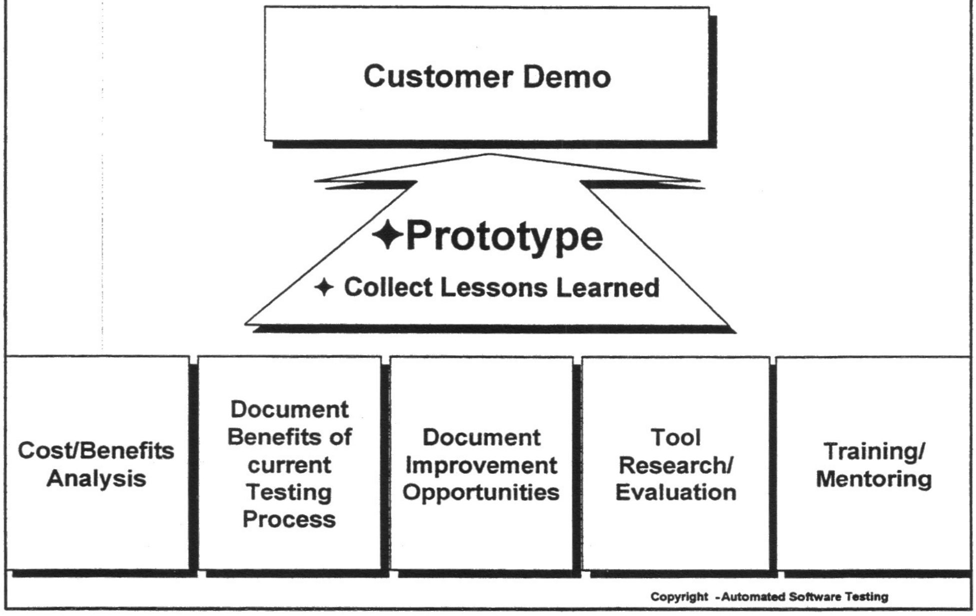
- Acquiring Management Support

Decision to Automate (cont'd)

Acquiring Management Support

- Test Tool Proposal
  - Estimate the improvement opportunities
  - Approach for selecting the correct tool
  - Tool cost range
  - Additional time to introduce tool
  - Tool expertise
  - Tool training cost
  - Tool evaluation domain
  - Tool rollout process

Case Study: Decision to Automate/Tool Acquisition



Customer Demo

✦Prototype

✦ Collect Lessons Learned

Cost/Benefits Analysis

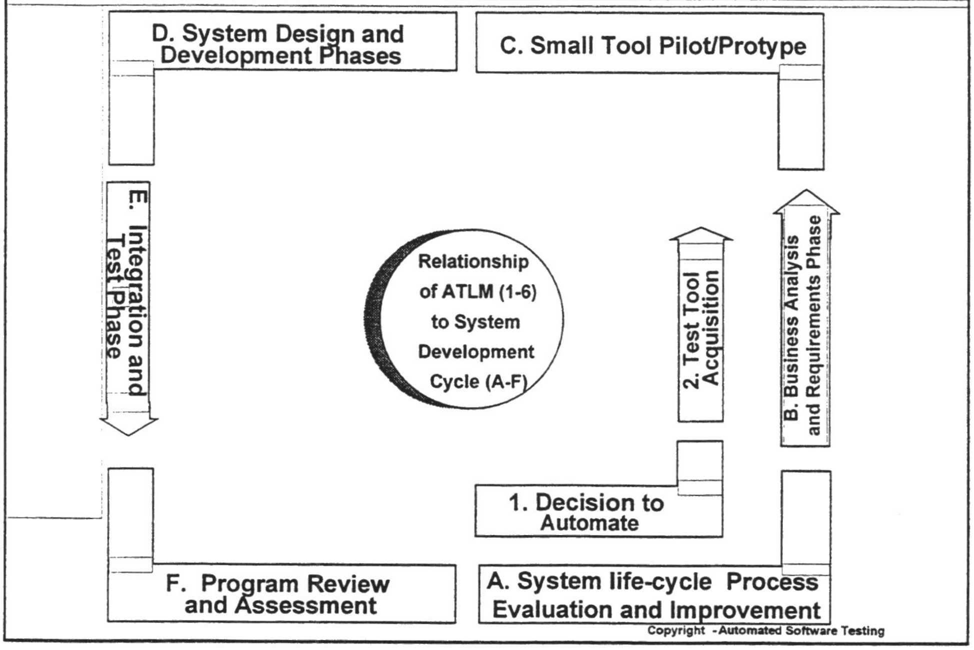
Document Benefits of current Testing Process

Document Improvement Opportunities

Tool Research/ Evaluation

Training/ Mentoring

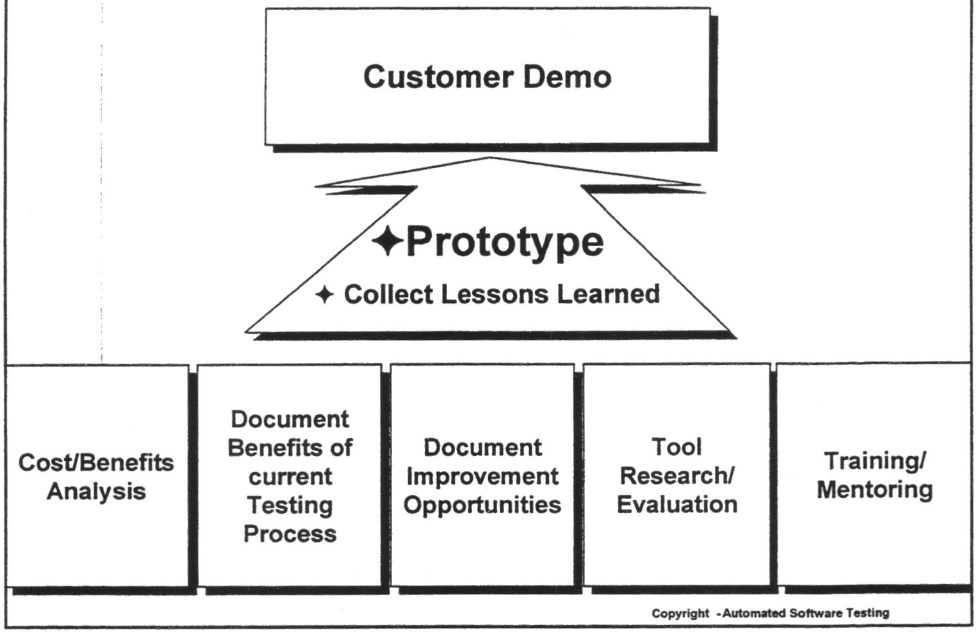
Automated Testing Life-cycle Methodology (ATLM)


D. System Design and Development Phases
C. Small Tool Pilot/Protype
E. Integration and Test Phase
Relationship of ATLM (1-6) to System Development Cycle (A-F)
2. Test Tool Acquisition
B. Business Analysis and Requirements Phase
1. Decision to Automate
F. Program Review and Assessment
A. System life-cycle Process Evaluation and Improvement

Exercise – Brainstorming (2 – 3 min)

- Select a project on which you are currently working, or with which you might be working on in the future.
- List the most important factors to be considered when choosing a tool

Case Study: Decision to Automate/Tool Acquisition



Customer Demo

✦Prototype

✦ Collect Lessons Learned

Cost/Benefits Analysis

Document Benefits of current Testing Process

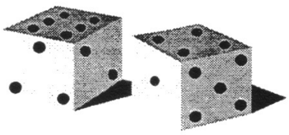
Document Improvement Opportunities

Tool Research/ Evaluation

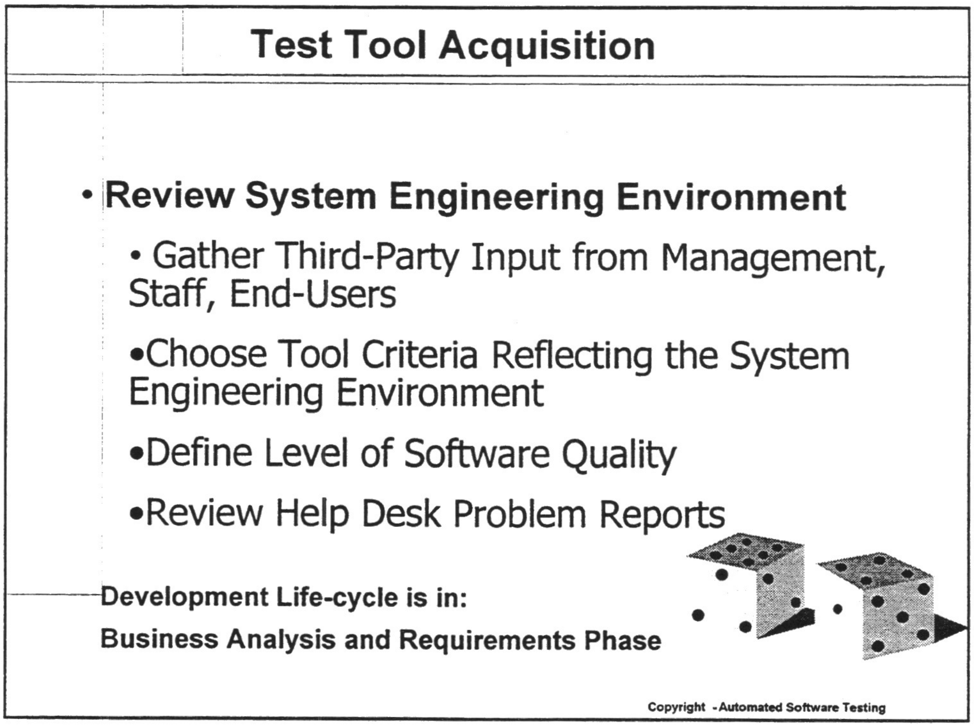
Training/ Mentoring

Test Tool Acquisition

- Review System Engineering Environment
- Review Tools Available on the market
- Tool Research and Evaluation
  - — Reviewing the Test Life-Cycle Tools
- Tool Purchase



Development Life-cycle is in:
Business Analysis and Requirements Phase

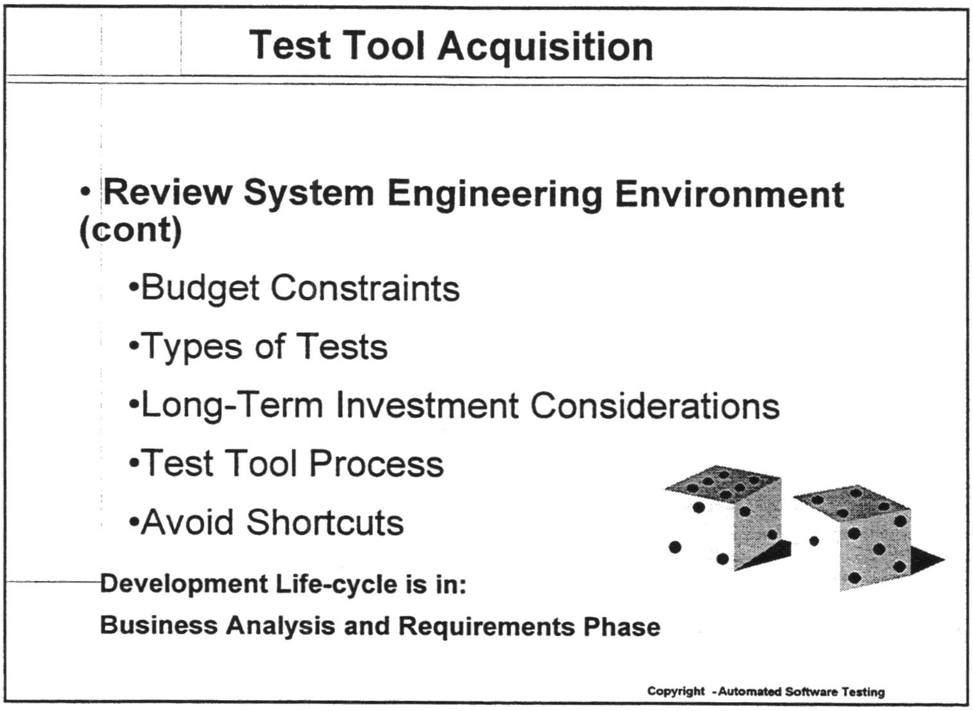


Test Tool Acquisition

- **Review System Engineering Environment**
  - Gather Third-Party Input from Management, Staff, End-Users
  - Choose Tool Criteria Reflecting the System Engineering Environment
  - Define Level of Software Quality
  - Review Help Desk Problem Reports

Development Life-cycle is in:

Business Analysis and Requirements Phase



Test Tool Acquisition

- **Review System Engineering Environment (cont)**
  - Budget Constraints
  - Types of Tests
  - Long-Term Investment Considerations
  - Test Tool Process
  - Avoid Shortcuts

Development Life-cycle is in:

Business Analysis and Requirements Phase

Test Tool Evaluation - Case Study

High-level Test Tool Selection Objectives:

- Vendor has sufficient market share to ensure vendors potential to stay in business
- Supports Web (and possibly VC++, VB) applications
- Provides automated test execution and results logging as well as reporting.
- Is compatible with Windows 2000, DHTML, HTML, Javascript, XML, and Java Applets
- Supports all testing phases and testing in the web environment (and desktop app written in VC++ ,VB):

Test Tool Evaluation (cont):

Detailed test feature and capability assessment

- Test Development Capability (Capture/Playback)
- Test Execution Capability
- Test Tool Integration Capability
- Test Reporting Capability
- Load/Stress Test Capability

- Cost Model

Test Tool Evaluation – A Case Study

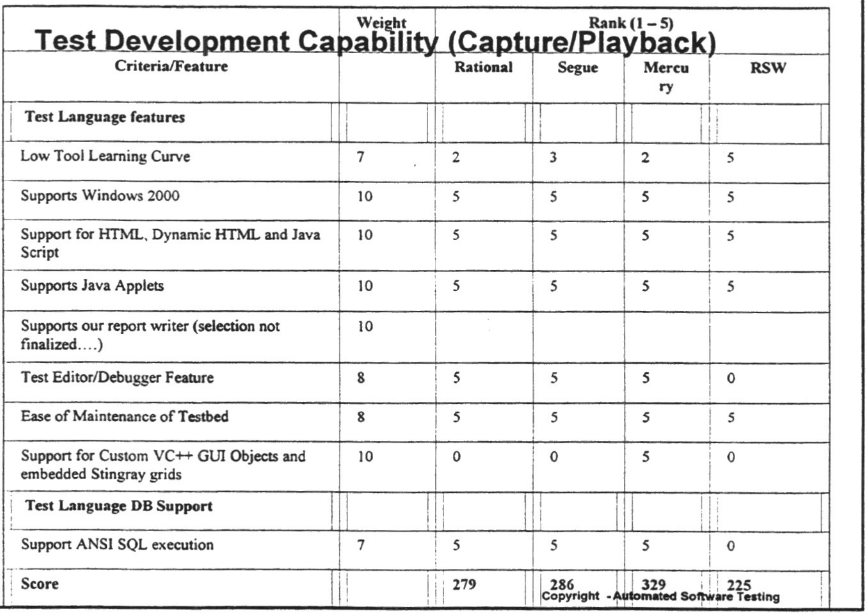
- Tool xyz worked fine on one project with html, java, etc.
- New company, new project: dhtml – we actually discovered a bug with the tool –still waiting on the patch…. – had asked for prototype over and over……
- If buying multiple tools, make sure they install correctly and work together correctly
- Example: Micron computer needed BIOS upgrade – ReqPro didn't work

Very important Evaluate tool

in your environment

Copyright -Automated Software Testing

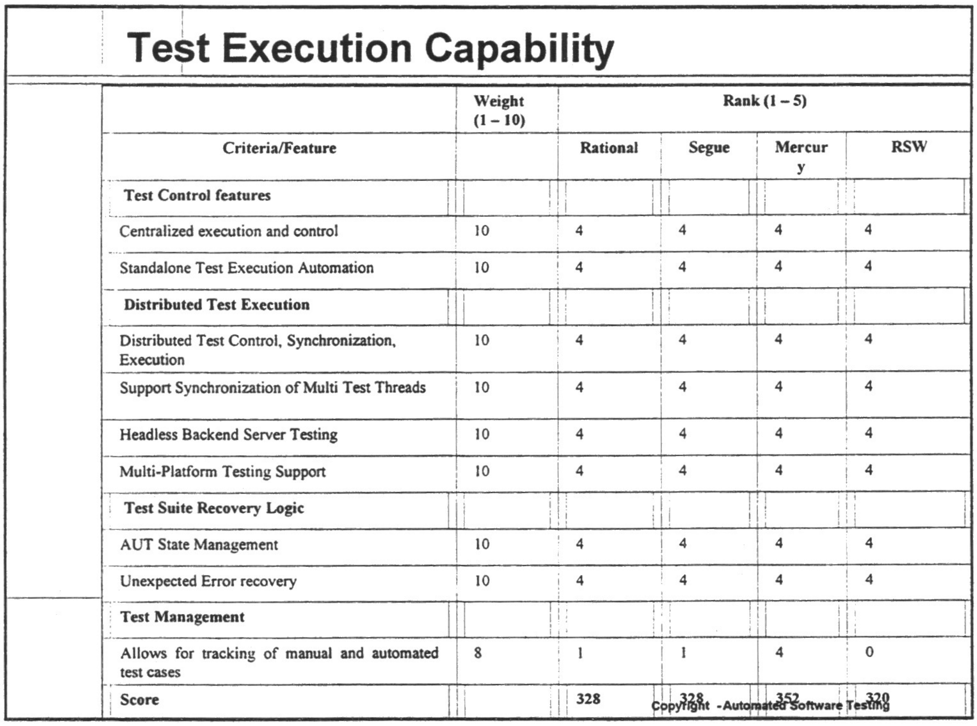
Test Development Capability (Capture/Playback)



Test Development Capability (Capture/Playback)

Criteria/Feature	Weight	Rank (1 – 5)			
		Rational	Segue	Mercu ry	RSW
Test Language features					
Low Tool Learning Curve	7	2	3	2	5
Supports Windows 2000	10	5	5	5	5
Support for HTML, Dynamic HTML and Java Script	10	5	5	5	5
Supports Java Applets	10	5	5	5	5
Supports our report writer (selection not finalized….)	10				
Test Editor/Debugger Feature	8	5	5	5	0
Ease of Maintenance of Testbed	8	5	5	5	5
Support for Custom VC++ GUI Objects and embedded Stingray grids	10	0	0	5	0
Test Language DB Support					
Support ANSI SQL execution	7	5	5	5	0
Score		279	286	329	225

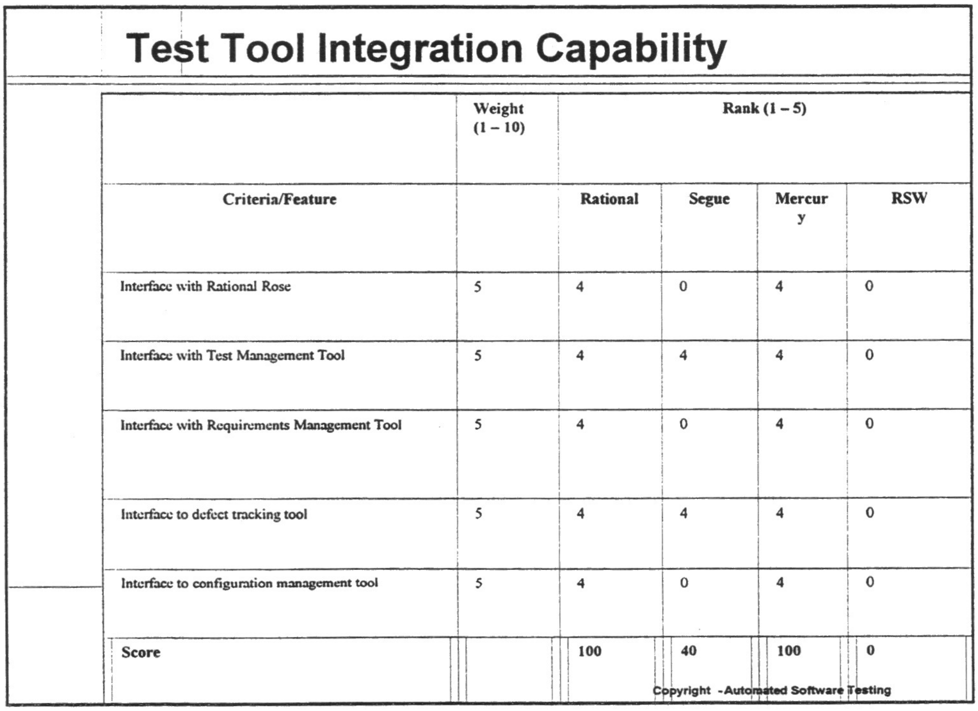
Copyright -Automated Software Testing



Test Execution Capability

Criteria/Feature	Weight (1 – 10)	Rank (1 – 5)			
		Rational	Segue	Mercury	RSW
Test Control features					
Centralized execution and control	10	4	4	4	4
Standalone Test Execution Automation	10	4	4	4	4
Distributed Test Execution					
Distributed Test Control, Synchronization, Execution	10	4	4	4	4
Support Synchronization of Multi Test Threads	10	4	4	4	4
Headless Backend Server Testing	10	4	4	4	4
Multi-Platform Testing Support	10	4	4	4	4
Test Suite Recovery Logic					
AUT State Management	10	4	4	4	4
Unexpected Error recovery	10	4	4	4	4
Test Management					
Allows for tracking of manual and automated test cases	8	1	1	4	0
Score		328	328	352	320

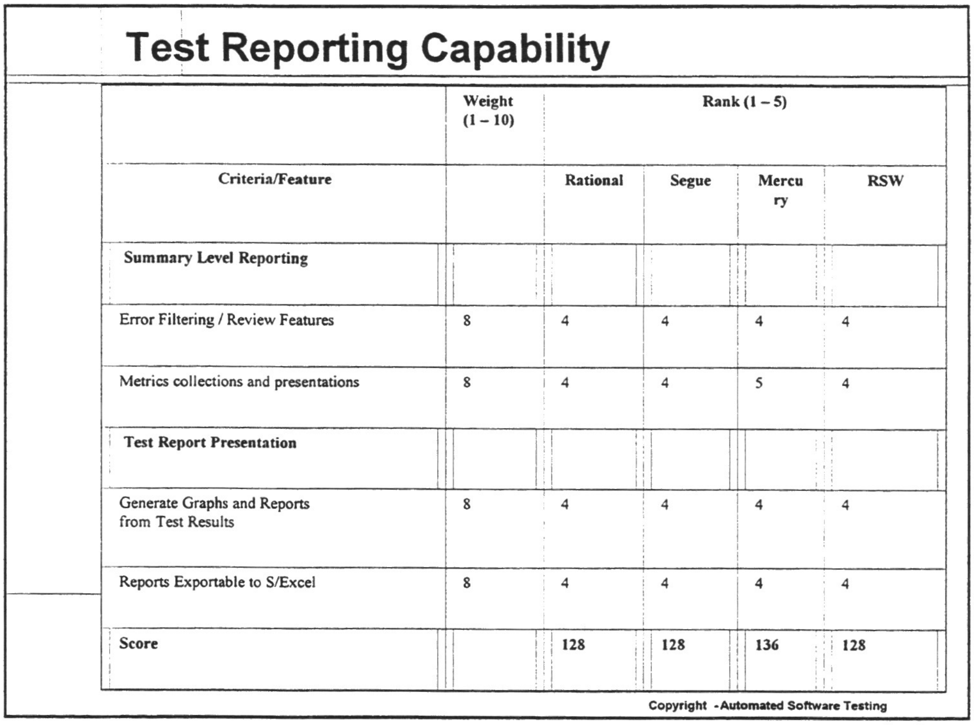
Copyright -Automated Software Testing



Test Tool Integration Capability

Criteria/Feature	Weight (1 – 10)	Rank (1 – 5)			
		Rational	Segue	Mercury	RSW
Interface with Rational Rose	5	4	0	4	0
Interface with Test Management Tool	5	4	4	4	0
Interface with Requirements Management Tool	5	4	0	4	0
Interface to defect tracking tool	5	4	4	4	0
Interface to configuration management tool	5	4	0	4	0
Score		100	40	100	0

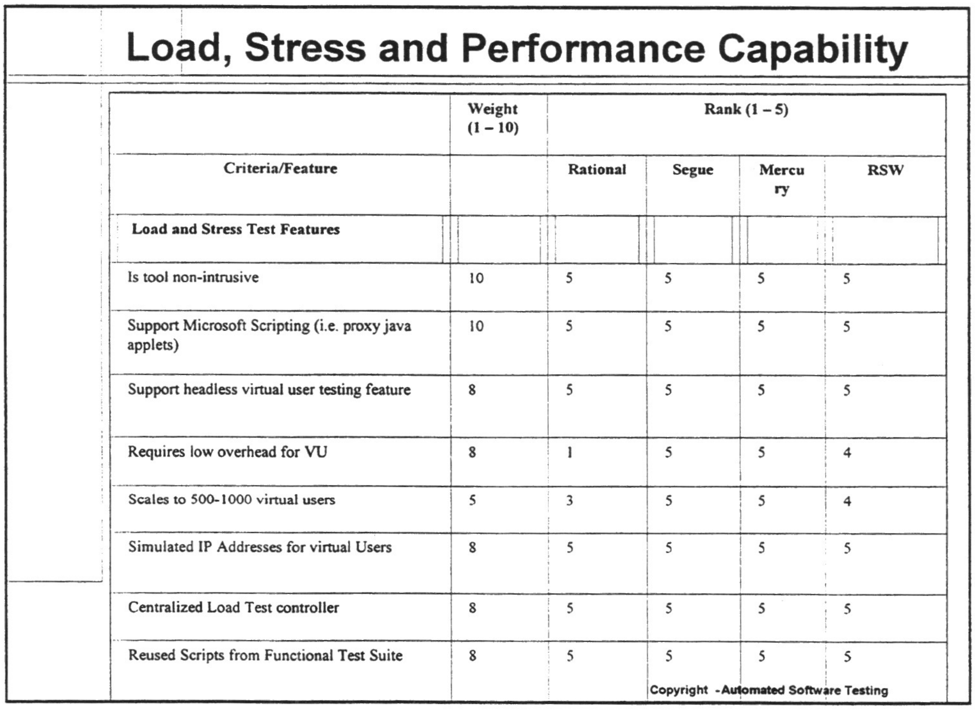
Copyright -Automated Software Testing



Test Reporting Capability

	Weight (1 – 10)	Rank (1 – 5)			
Criteria/Feature		Rational	Segue	Mercury	RSW
Summary Level Reporting					
Error Filtering / Review Features	8	4	4	4	4
Metrics collections and presentations	8	4	4	5	4
Test Report Presentation					
Generate Graphs and Reports from Test Results	8	4	4	4	4
Reports Exportable to S/Excel	8	4	4	4	4
Score		**128**	**128**	**136**	**128**

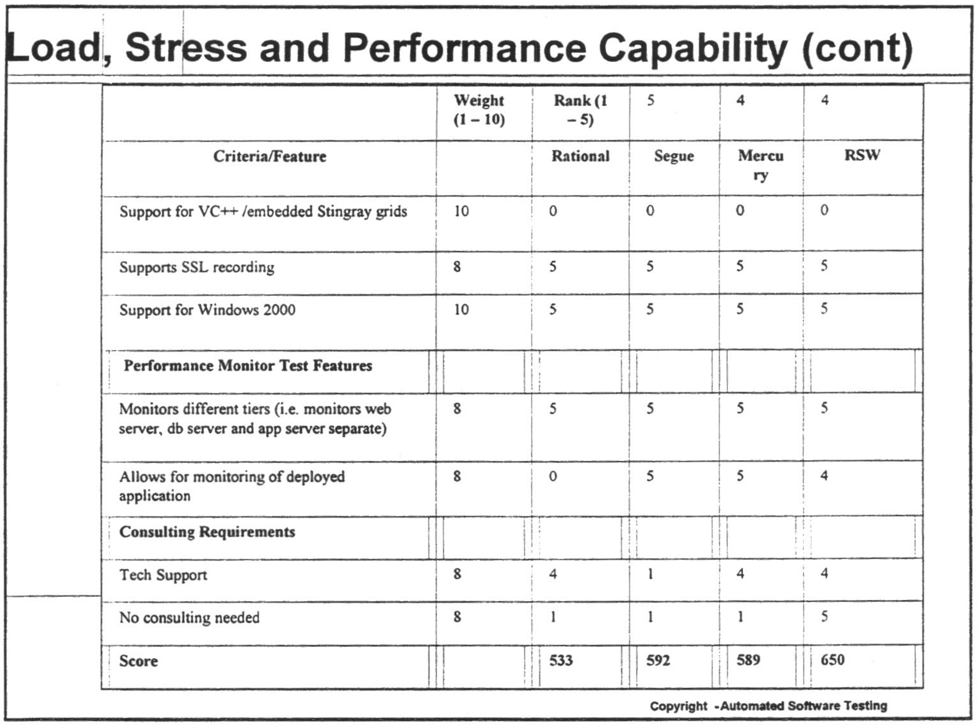
Copyright - Automated Software Testing



Load, Stress and Performance Capability

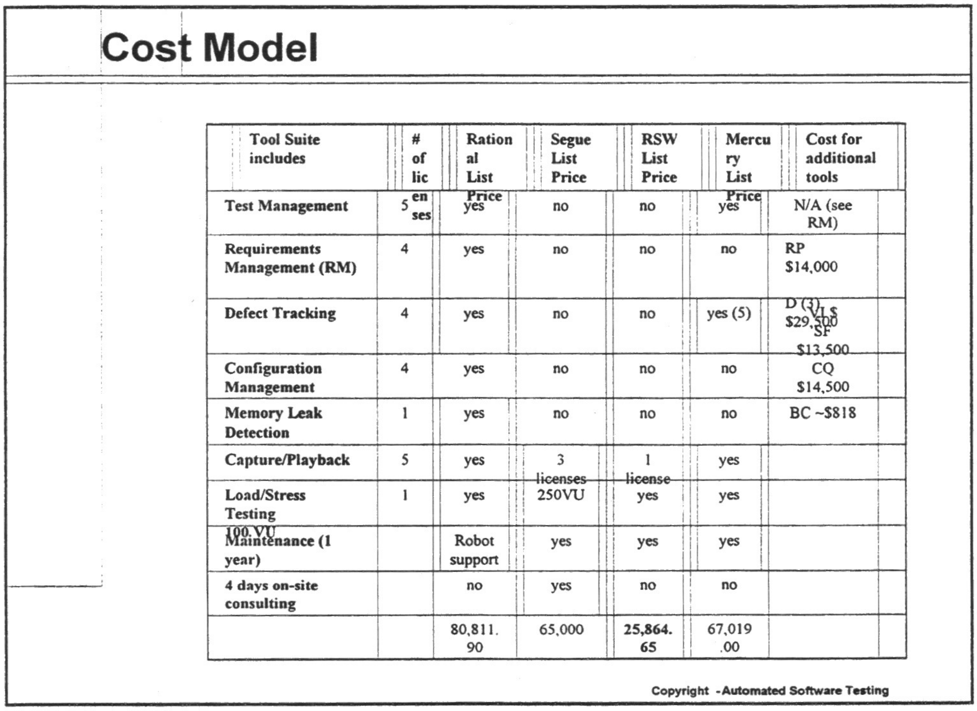
	Weight (1 – 10)	Rank (1 – 5)			
Criteria/Feature		Rational	Segue	Mercury	RSW
Load and Stress Test Features					
Is tool non-intrusive	10	5	5	5	5
Support Microsoft Scripting (i.e. proxy java applets)	10	5	5	5	5
Support headless virtual user testing feature	8	5	5	5	5
Requires low overhead for VU	8	1	5	5	4
Scales to 500-1000 virtual users	5	3	5	5	4
Simulated IP Addresses for virtual Users	8	5	5	5	5
Centralized Load Test controller	8	5	5	5	5
Reused Scripts from Functional Test Suite	8	5	5	5	5

Copyright - Automated Software Testing



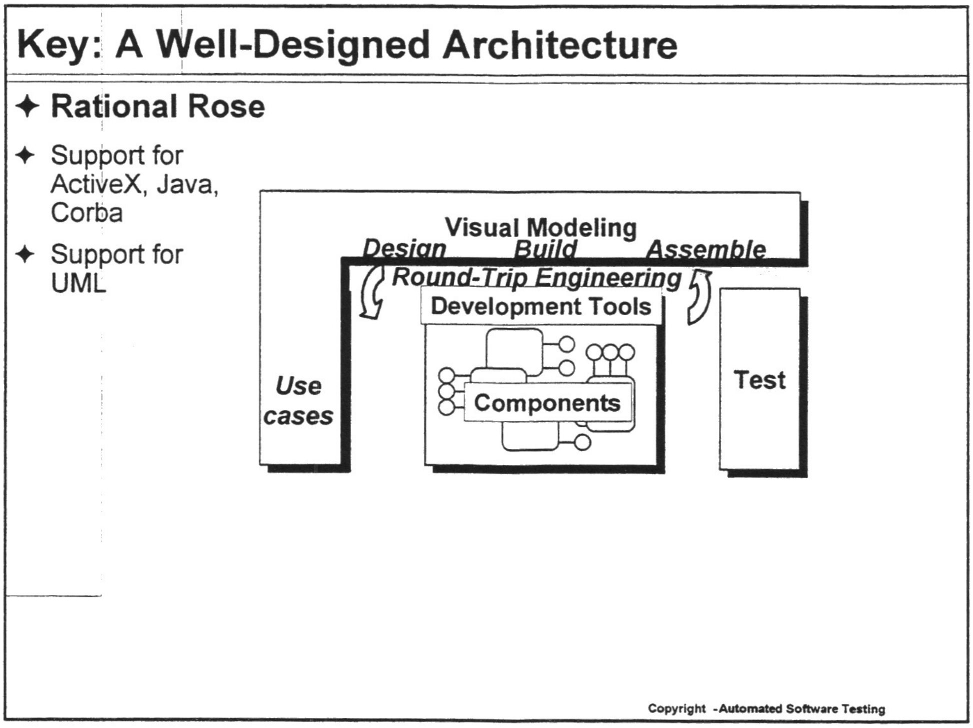
Load, Stress and Performance Capability (cont)

	Weight (1 – 10)	Rank (1 – 5)	5	4	4
Criteria/Feature		Rational	Segue	Mercury	RSW
Support for VC++ /embedded Stingray grids	10	0	0	0	0
Supports SSL recording	8	5	5	5	5
Support for Windows 2000	10	5	5	5	5
Performance Monitor Test Features					
Monitors different tiers (i.e. monitors web server, db server and app server separate)	8	5	5	5	5
Allows for monitoring of deployed application	8	0	5	5	4
Consulting Requirements					
Tech Support	8	4	1	4	4
No consulting needed	8	1	1	1	5
Score		533	592	589	650



Cost Model

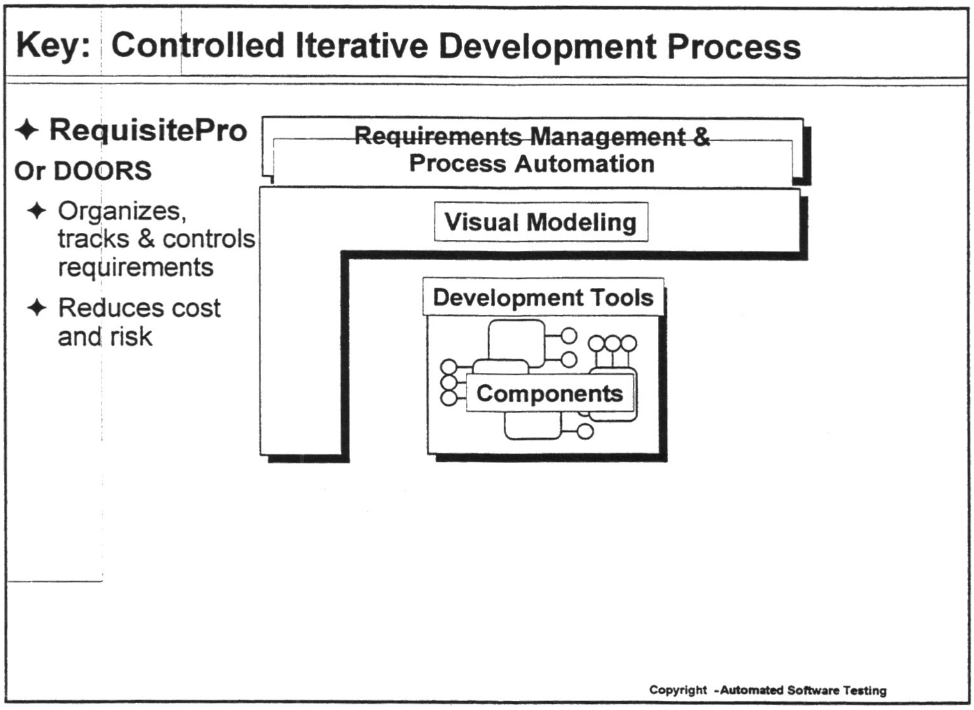
Tool Suite includes	# of licenses	Rational List Price	Segue List Price	RSW List Price	Mercury List Price	Cost for additional tools
Test Management	5	yes	no	no	yes	N/A (see RM)
Requirements Management (RM)	4	yes	no	no	no	RP $14,000
Defect Tracking	4	yes	no	no	yes (5)	D (3) $29,500 VI $ SF $13,500
Configuration Management	4	yes	no	no	no	CQ $14,500
Memory Leak Detection	1	yes	no	no	no	BC ~$818
Capture/Playback	5	yes	3 licenses	1 license	yes	
Load/Stress Testing 100 VU	1	yes	250VU	yes	yes	
Maintenance (1 year)		Robot support	yes	yes	yes	
4 days on-site consulting		no	yes	no	no	
		80,811.90	65,000	25,864.65	67,019.00	



Key: A Well-Designed Architecture

- ✦ **Rational Rose**
- ✦ Support for ActiveX, Java, Corba
- ✦ Support for UML

Copyright -Automated Software Testing



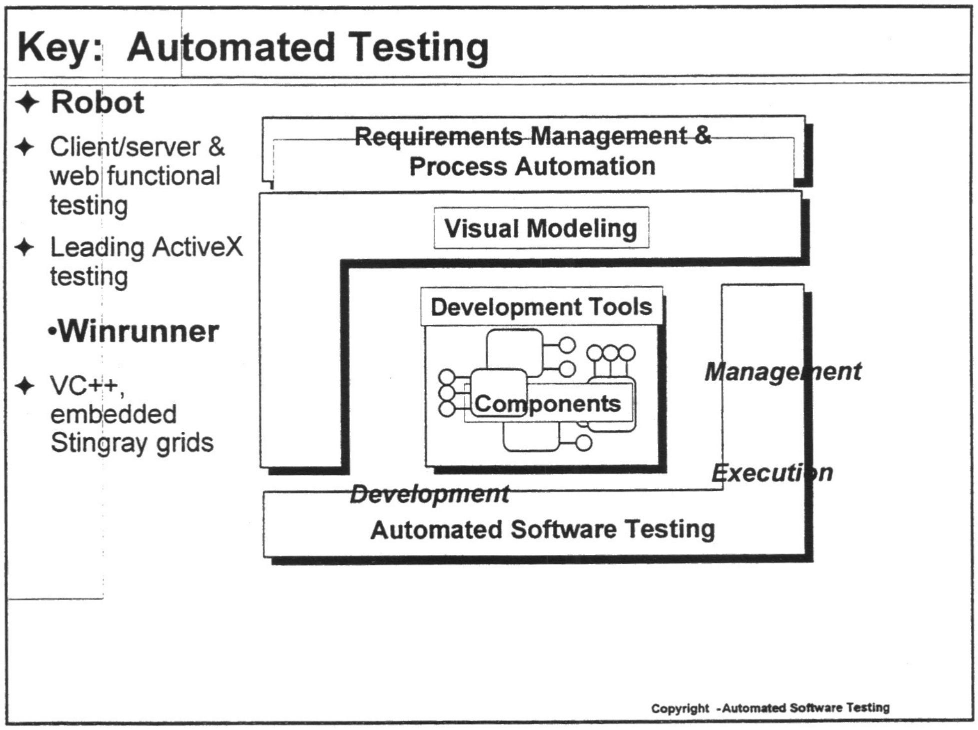
Key: Controlled Iterative Development Process

- ✦ **RequisitePro**

Or DOORS

- ✦ Organizes, tracks & controls requirements
- ✦ Reduces cost and risk

Copyright -Automated Software Testing

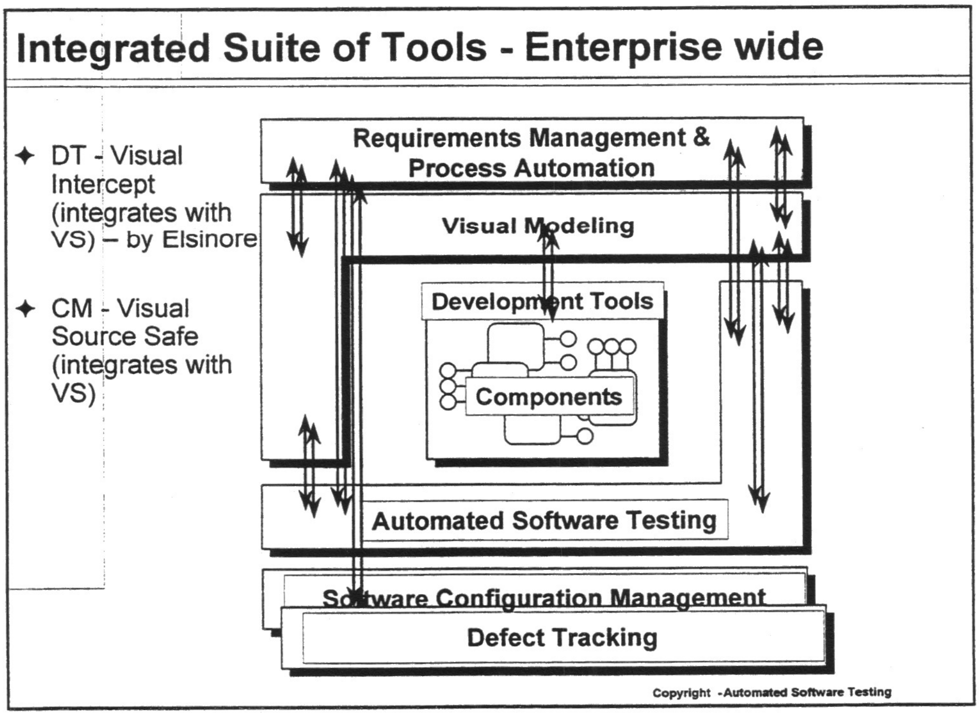


Key: Automated Testing

- ✦ **Robot**
- ✦ Client/server & web functional testing
- ✦ Leading ActiveX testing

•Winrunner

- ✦ VC++, embedded Stingray grids



Integrated Suite of Tools - Enterprise wide

- ✦ DT - Visual Intercept (integrates with VS) – by Elsinore
- ✦ CM - Visual Source Safe (integrates with VS)

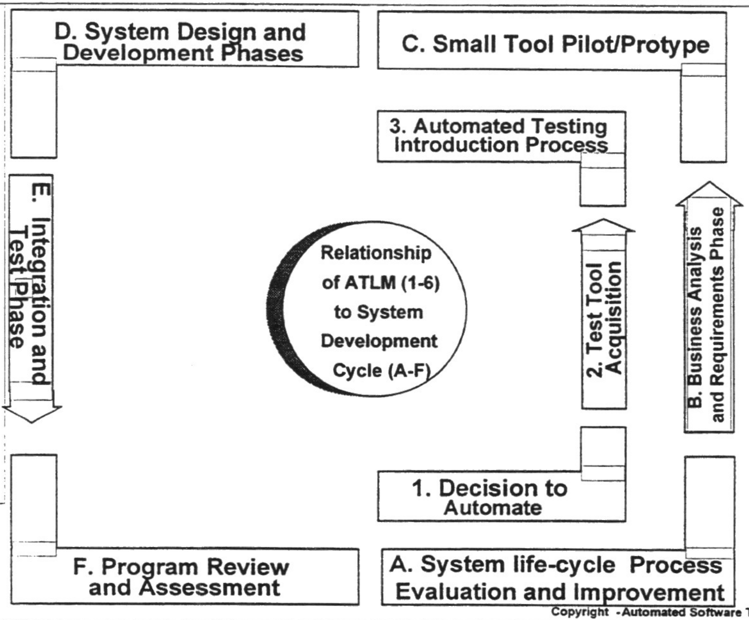
Requirements Management &
Process Automation
Visual Modeling
Development Tools
Components
Automated Software Testing
~~Software Configuration Management~~
Defect Tracking

Load Testing Tools

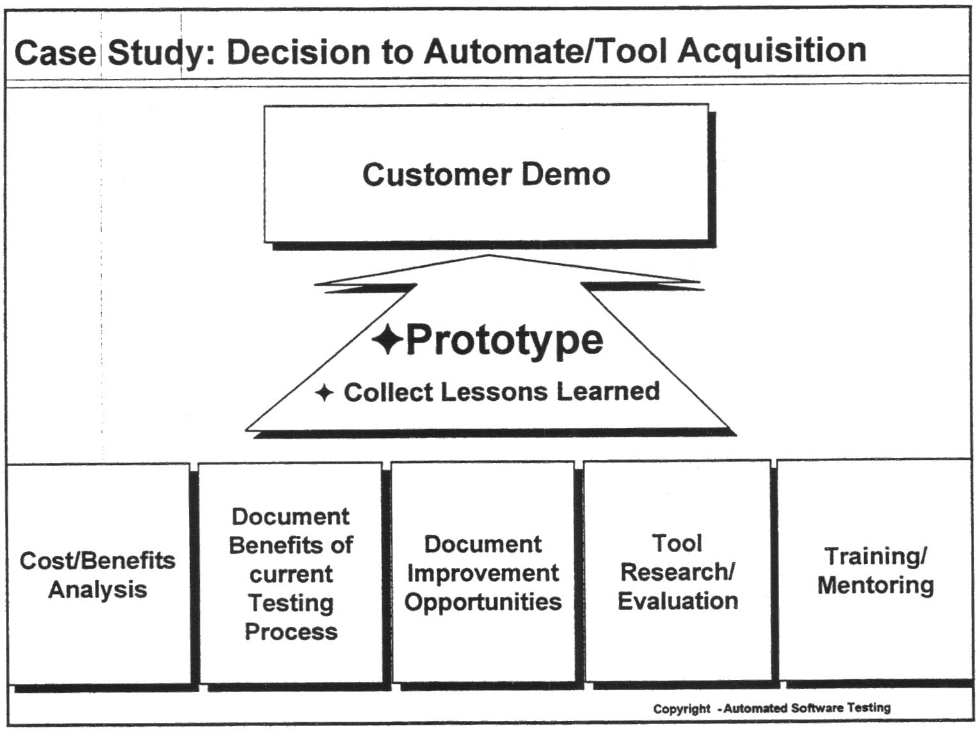
- •Conduct Load Testing using Load Testing Tool
- •Example: Performance Studio/Loadrunner
- •Reuse Scripts developed using Capture Playback tool - not VU users
- •Other example: MS WAS free load testing tool

Copyright -Automated Software Testing

Automated Testing Life-cycle Methodology (ATLM)


D. System Design and Development Phases
C. Small Tool Pilot/Protype
3. Automated Testing Introduction Process
E. Integration and Test Phase
Relationship of ATLM (1-6) to System Development Cycle (A-F)
2. Test Tool Acquisition
B. Business Analysis and Requirements Phase
1. Decision to Automate
F. Program Review and Assessment
A. System life-cycle Process Evaluation and Improvement

Copyright -Automated Software Testing



Case Study: Decision to Automate/Tool Acquisition

Customer Demo

✦Prototype

✦ Collect Lessons Learned

Cost/Benefits Analysis

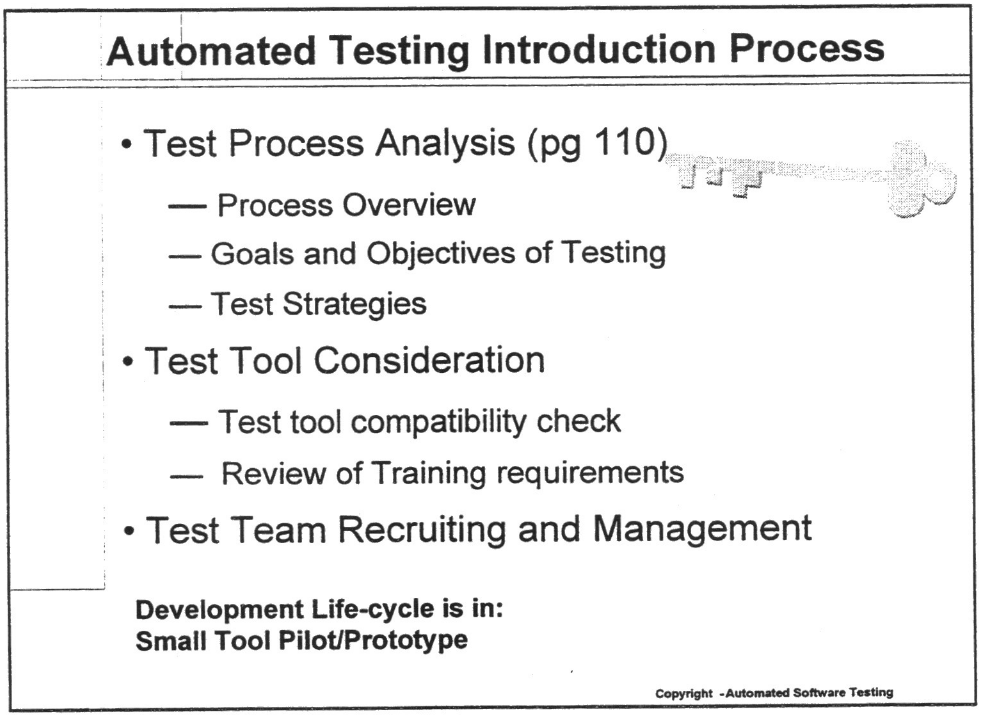
Document Benefits of current Testing Process

Document Improvement Opportunities

Tool Research/ Evaluation

Training/ Mentoring

Copyright -Automated Software Testing

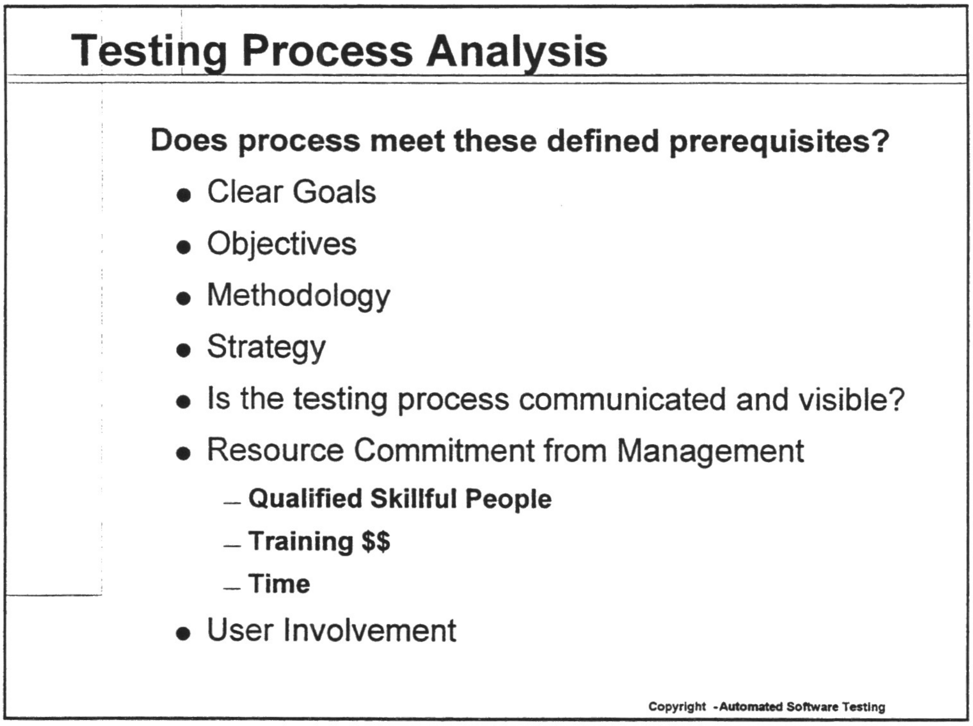


Automated Testing Introduction Process

- Test Process Analysis (pg 110)
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management

Development Life-cycle is in:
Small Tool Pilot/Prototype

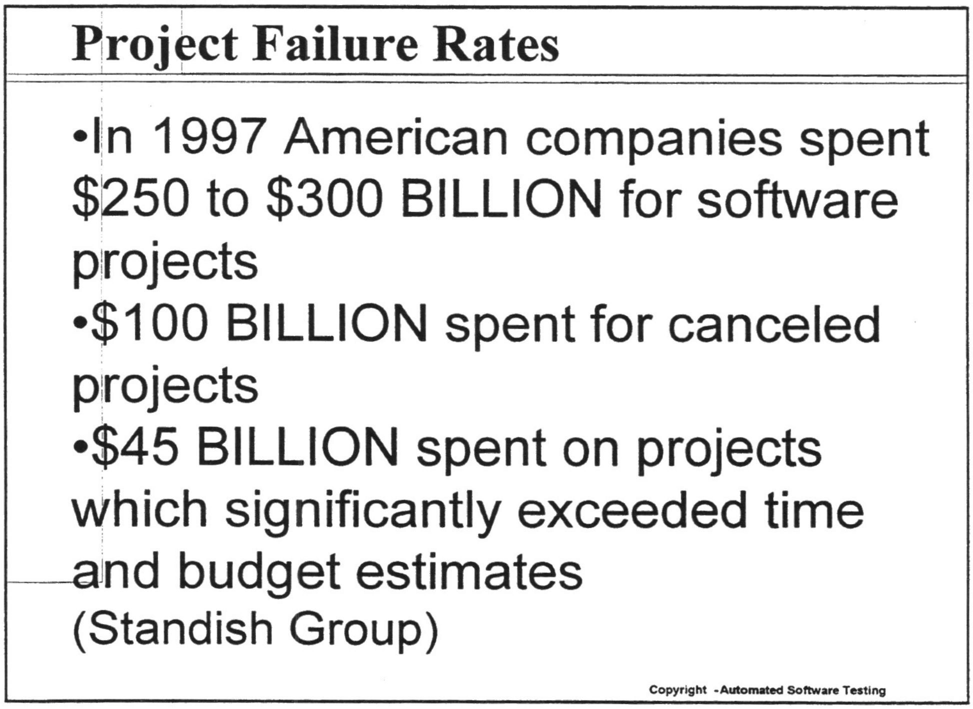
Copyright -Automated Software Testing



Testing Process Analysis

Does process meet these defined prerequisites?

- Clear Goals
- Objectives
- Methodology
- Strategy
- Is the testing process communicated and visible?
- Resource Commitment from Management
  - **Qualified Skillful People**
  - **Training $$**
  - **Time**
- User Involvement



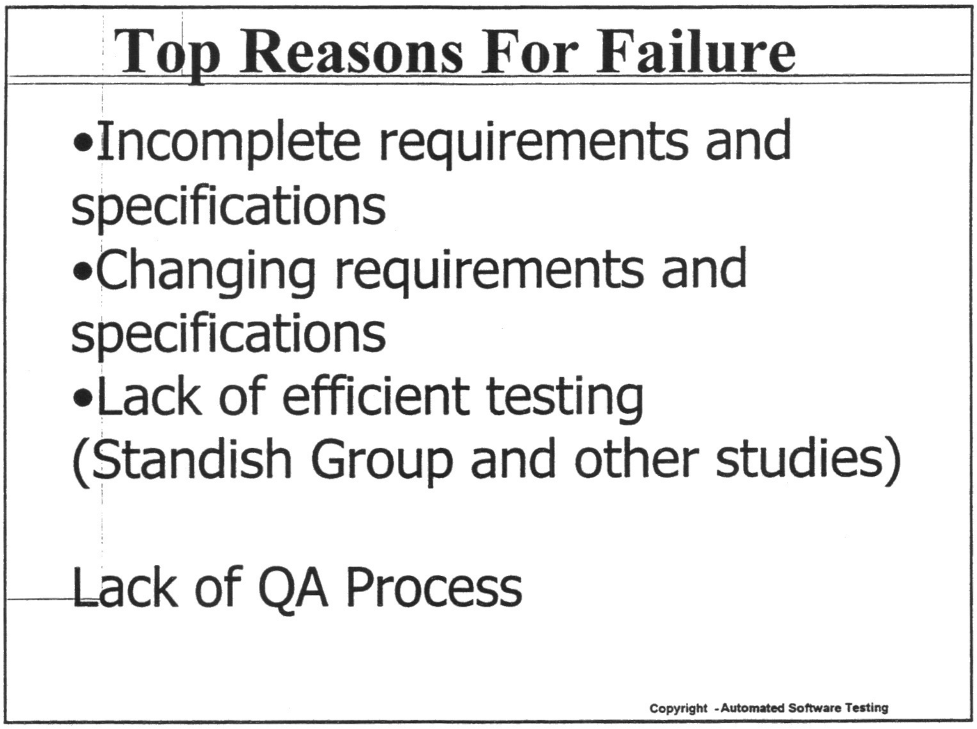
Project Failure Rates

•In 1997 American companies spent $250 to $300 BILLION for software projects

•$100 BILLION spent for canceled projects

•$45 BILLION spent on projects which significantly exceeded time and budget estimates

(Standish Group)

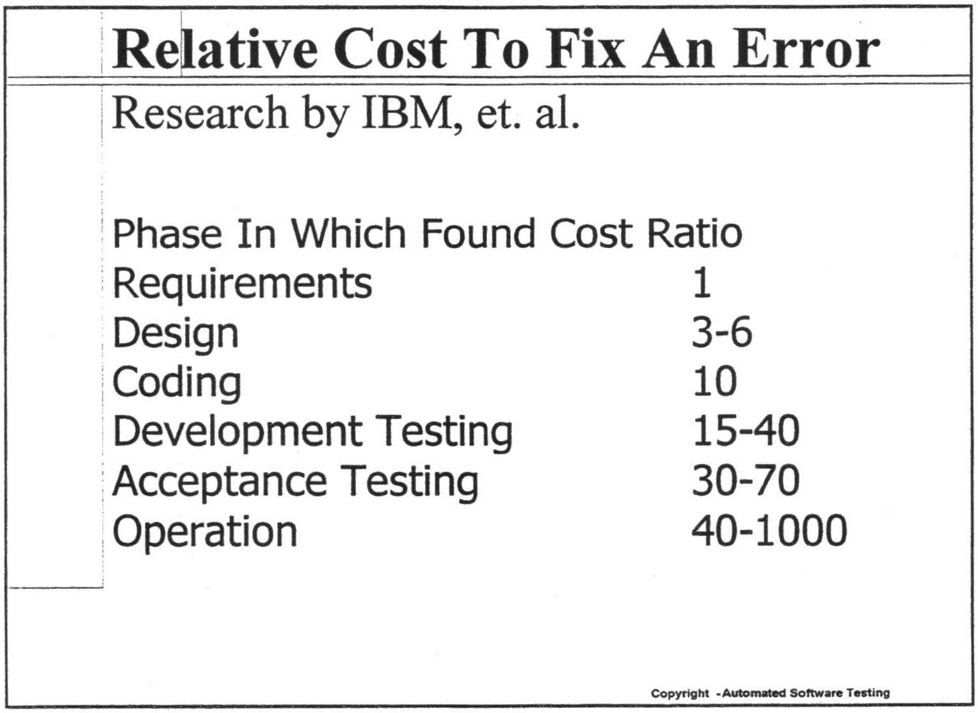


Top Reasons For Failure

- Incomplete requirements and specifications
- Changing requirements and specifications
- Lack of efficient testing

(Standish Group and other studies)

Lack of QA Process

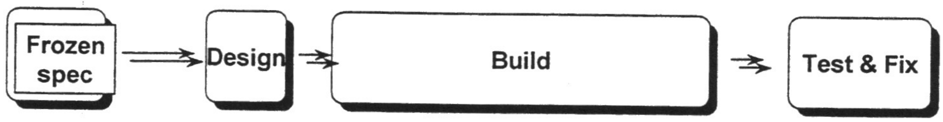


Relative Cost To Fix An Error

Research by IBM, et. al.

Phase In Which Found	Cost Ratio
Requirements	1
Design	3-6
Coding	10
Development Testing	15-40
Acceptance Testing	30-70
Operation	40-1000

Conventional Testing Process:



Frozen spec

Design

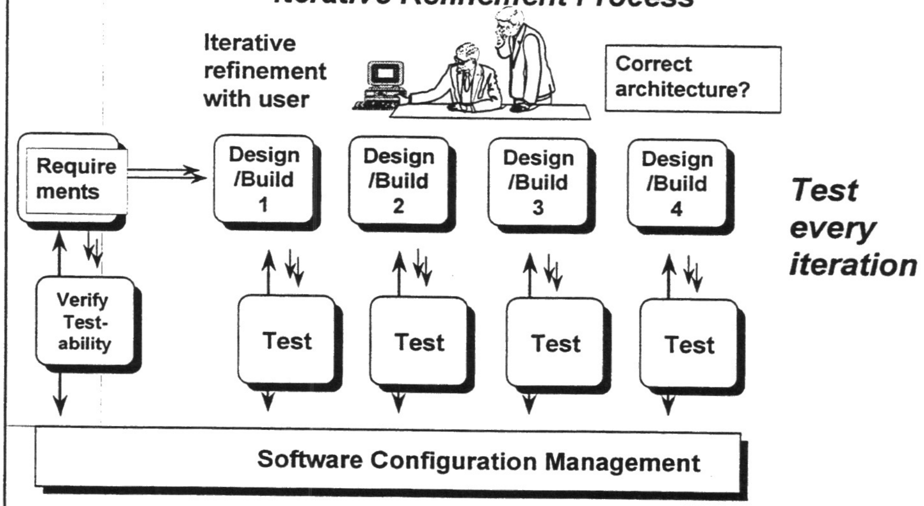
Build

Test & Fix

ATLM - Testing Throughout the Lifecycle

Iterative Development

Iterative Refinement Process



Iterative refinement with user

Correct architecture?

Require ments

Verify Test- ability

Design /Build 1

Design /Build 2

Design /Build 3

Design /Build 4

Test

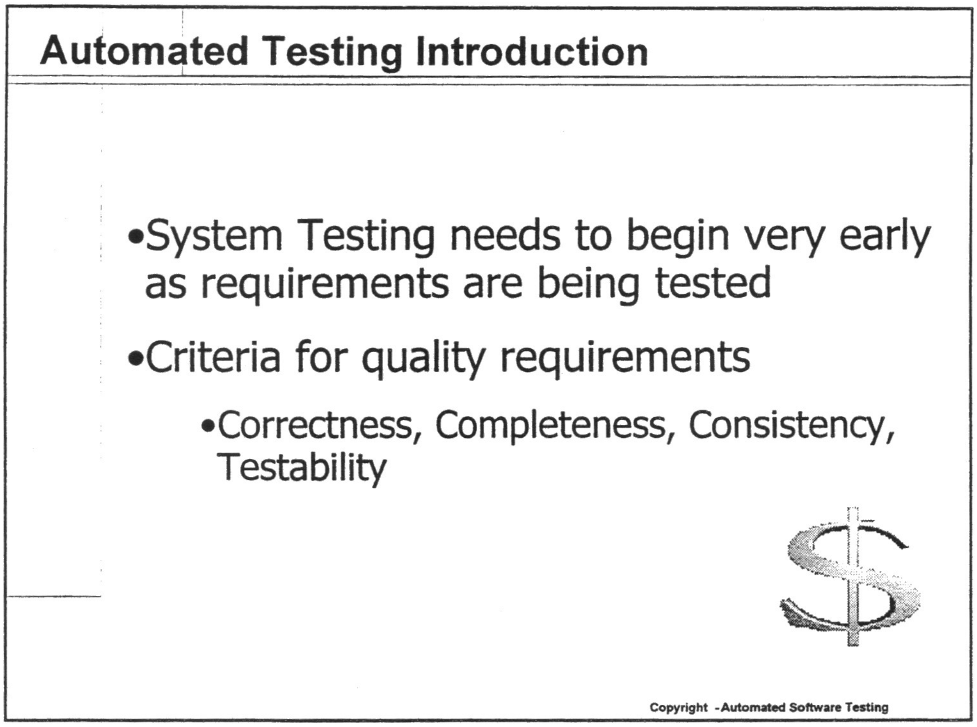
Test

Test

Test

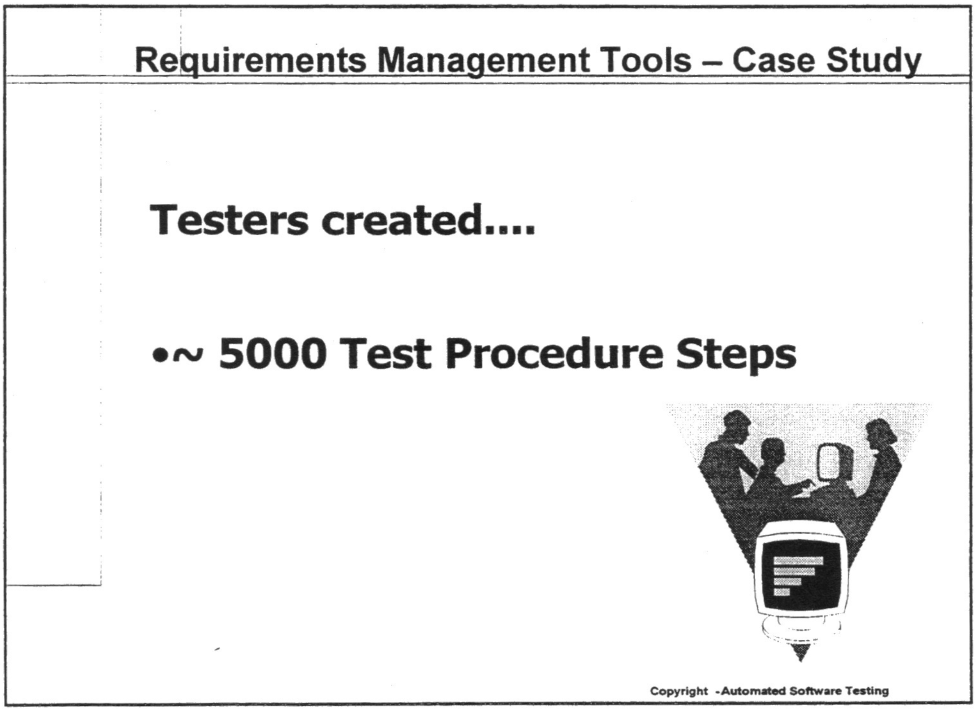
Software Configuration Management

Test every iteration



Automated Testing Introduction

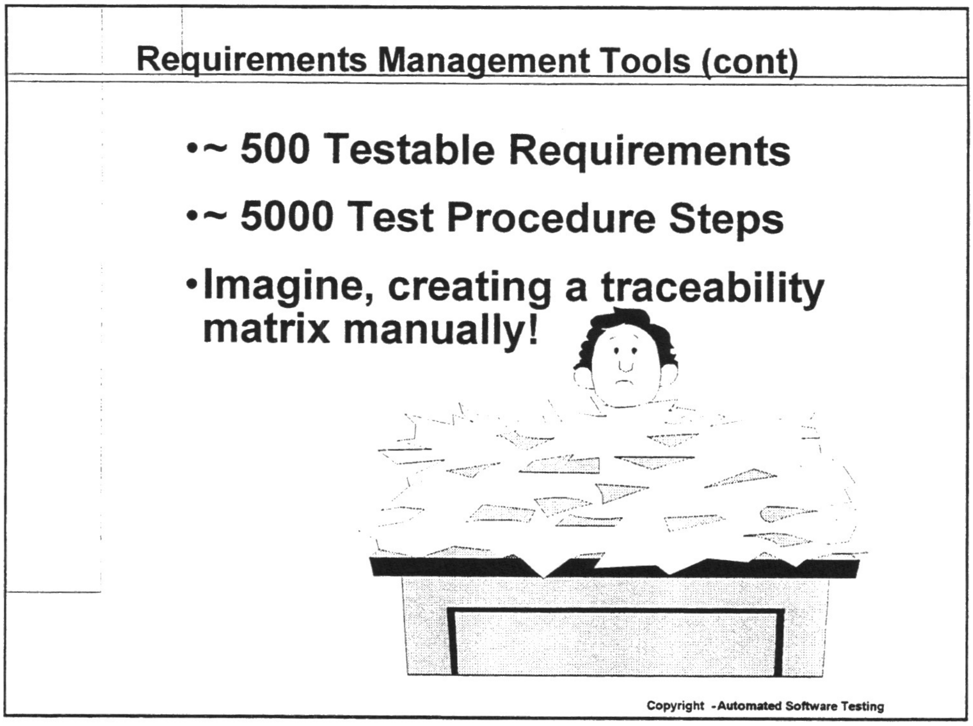
- System Testing needs to begin very early as requirements are being tested
- Criteria for quality requirements
  - Correctness, Completeness, Consistency, Testability



Requirements Management Tools – Case Study

Testers created....

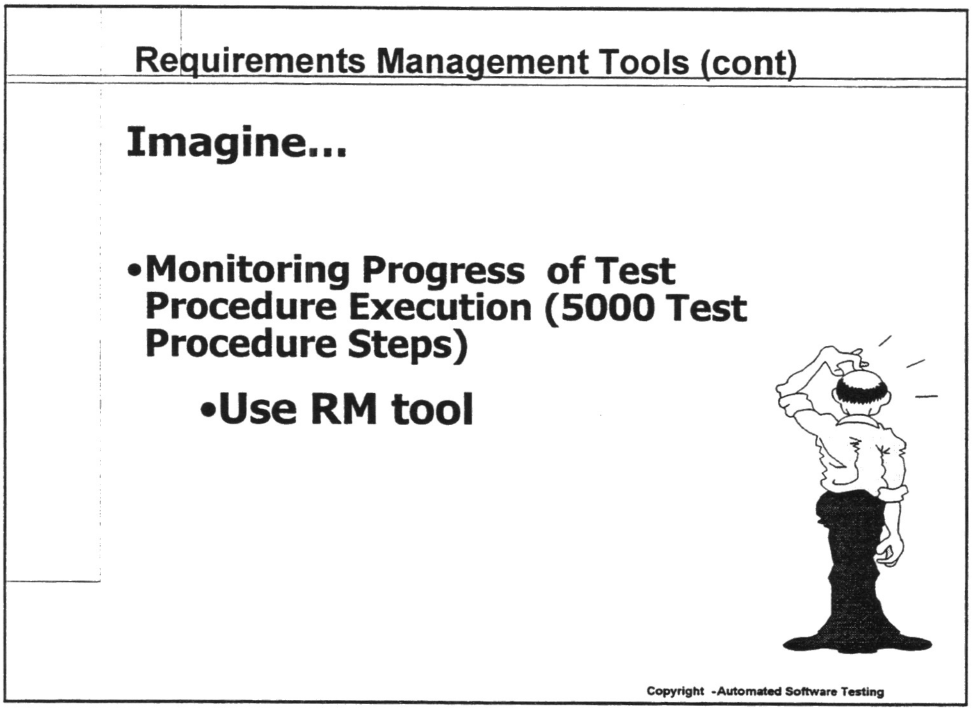
- **~ 5000 Test Procedure Steps**



Requirements Management Tools (cont)

- ~ 500 Testable Requirements
- ~ 5000 Test Procedure Steps
- Imagine, creating a traceability matrix manually!

Copyright -Automated Software Testing

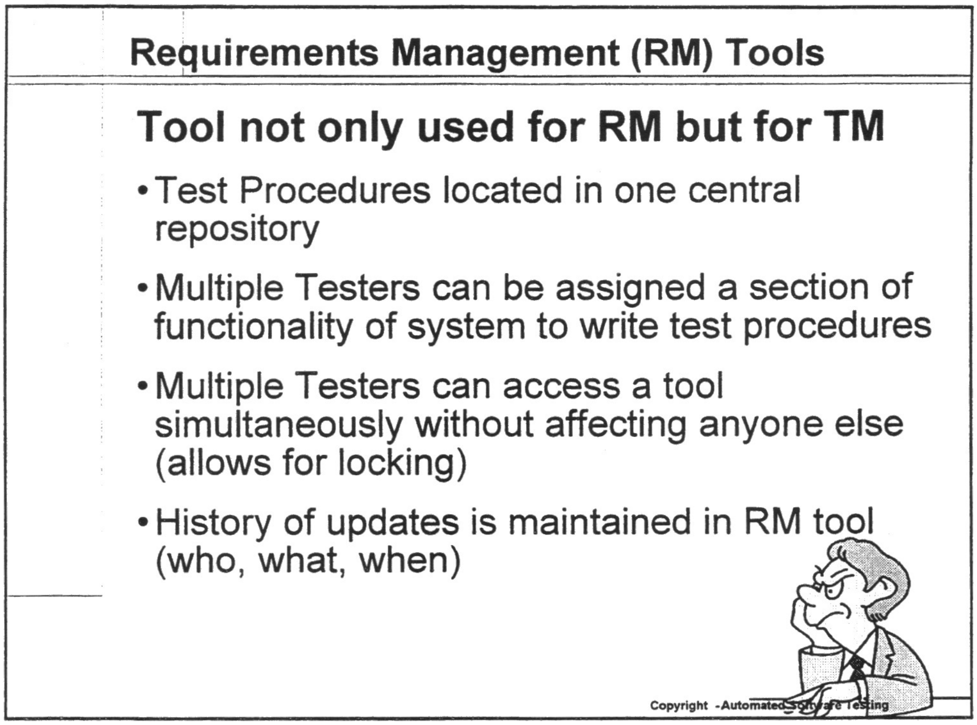


Requirements Management Tools (cont)

Imagine...

- **Monitoring Progress of Test Procedure Execution (5000 Test Procedure Steps)**
  - **Use RM tool**

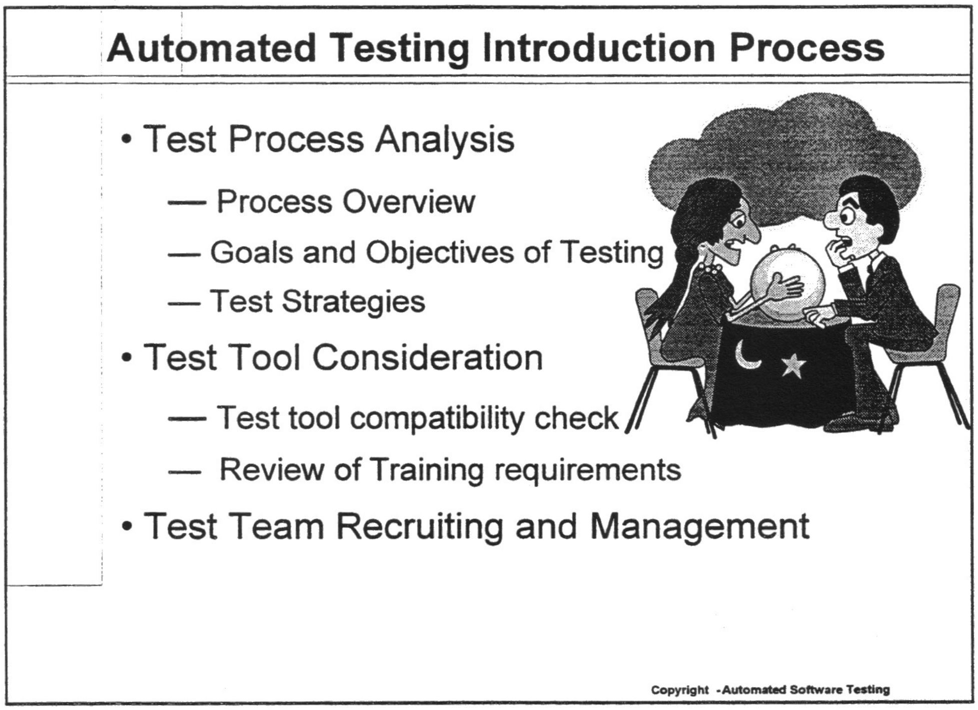
Copyright -Automated Software Testing



Requirements Management (RM) Tools

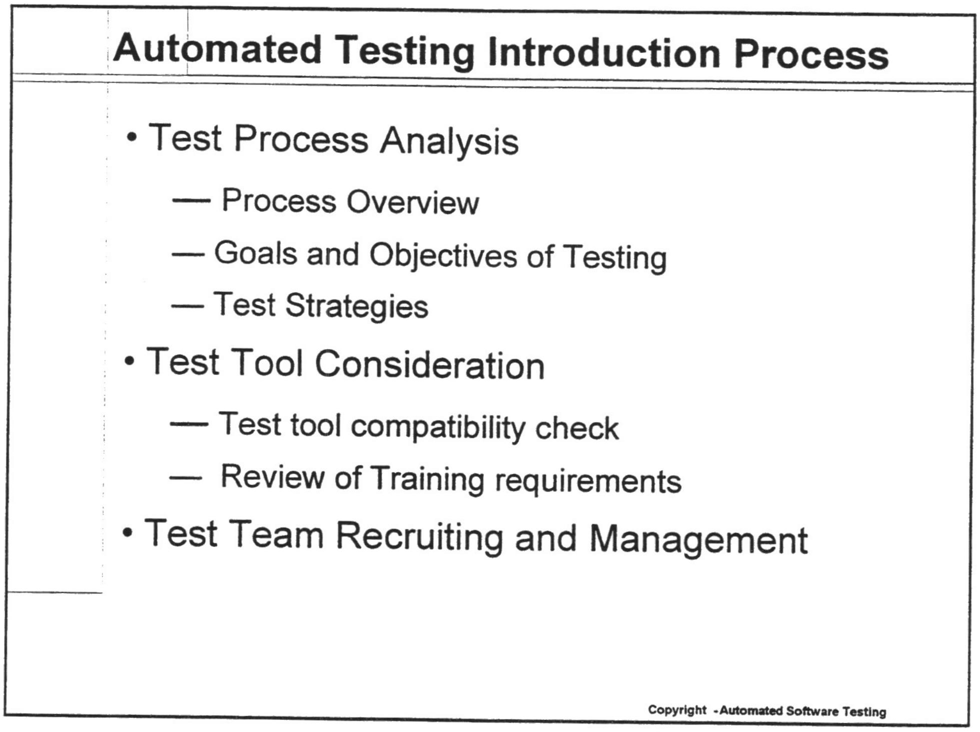
Tool not only used for RM but for TM

- Test Procedures located in one central repository
- Multiple Testers can be assigned a section of functionality of system to write test procedures
- Multiple Testers can access a tool simultaneously without affecting anyone else (allows for locking)
- History of updates is maintained in RM tool (who, what, when)



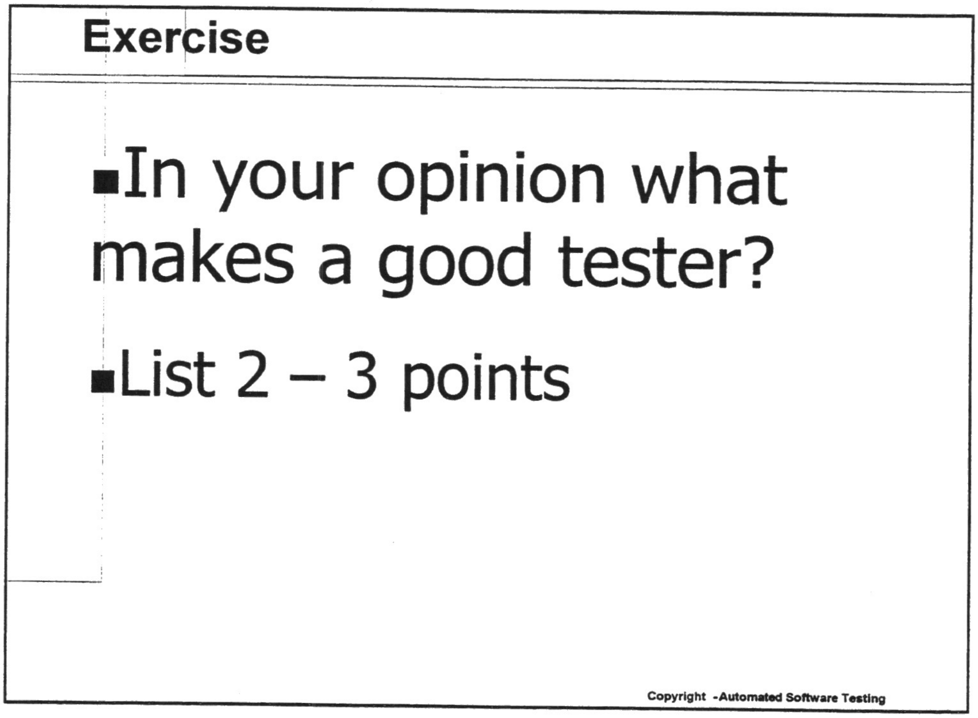
Automated Testing Introduction Process

- Test Process Analysis
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management



Automated Testing Introduction Process

- Test Process Analysis
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management



Exercise

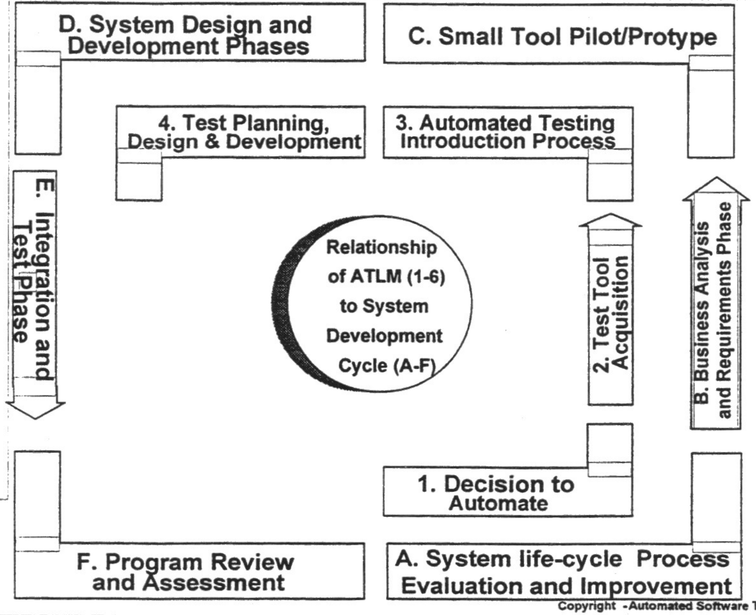
▪In your opinion what makes a good tester?

▪List 2 – 3 points

Automated Testing Introduction Process

- Test Process Analysis
  - — Process Overview
  - — Goals and Objectives of Testing
  - — Test Strategies
- Test Tool Consideration
  - — Test tool compatibility check
  - — Review of Training requirements
- Test Team Recruiting and Management

Automated Testing Life-cycle Methodology (ATLM)



D. System Design and Development Phases

C. Small Tool Pilot/Protype

4. Test Planning, Design & Development

3. Automated Testing Introduction Process

E. Integration and Test Phase

Relationship of ATLM (1-6) to System Development Cycle (A-F)

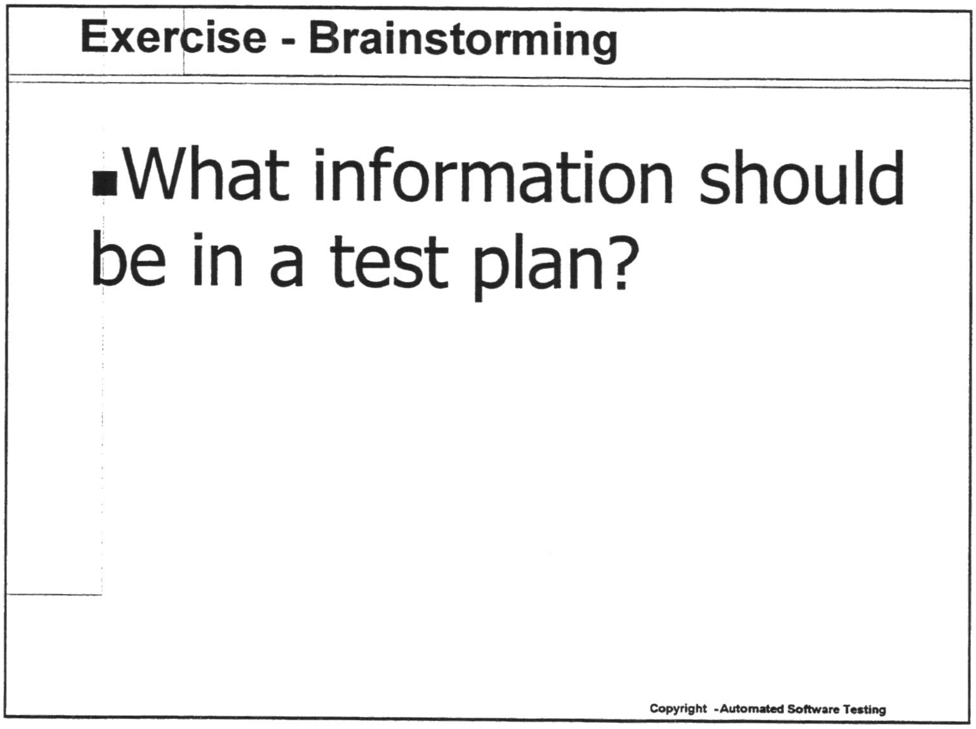
2. Test Tool Acquisition

B. Business Analysis and Requirements Phase

1. Decision to Automate

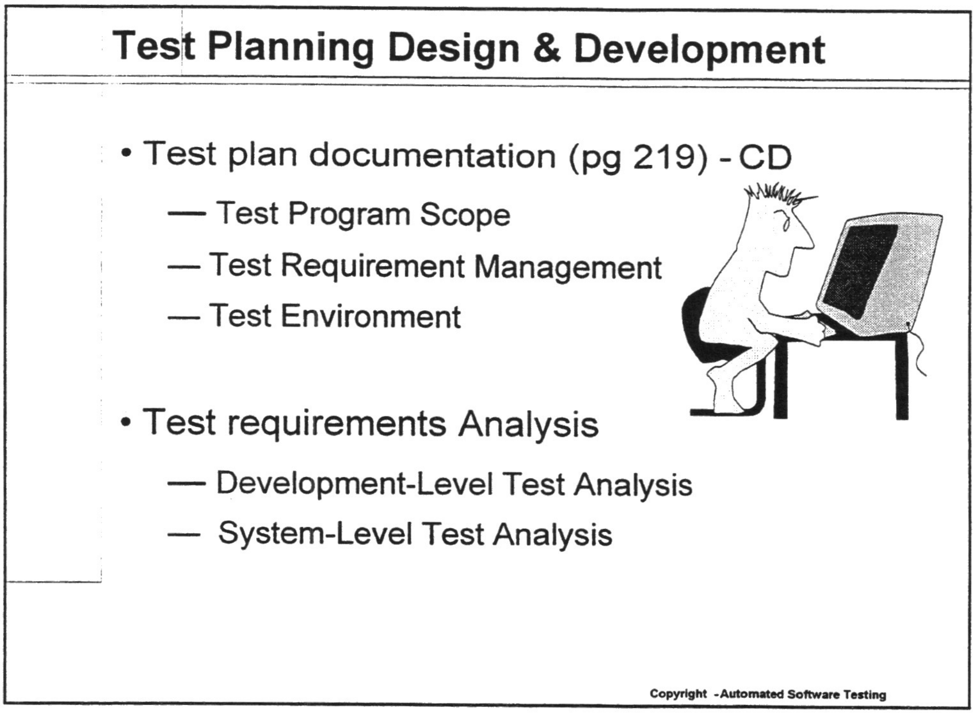
F. Program Review and Assessment

A. System life-cycle Process Evaluation and Improvement



Exercise - Brainstorming

- What information should be in a test plan?



Test Planning Design & Development

- Test plan documentation (pg 219) - CD
  - — Test Program Scope
  - — Test Requirement Management
  - — Test Environment
- Test requirements Analysis
  - — Development-Level Test Analysis
  - — System-Level Test Analysis

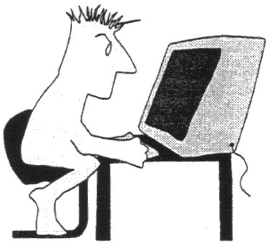
Test Planning Design & Development

- Test plan documentation
- Everything discussed so far needs to be in the test plan
- Goals, Objectives, Strategies, Roles and Responsibilities, Test Environment
- Take a look at test plan sample

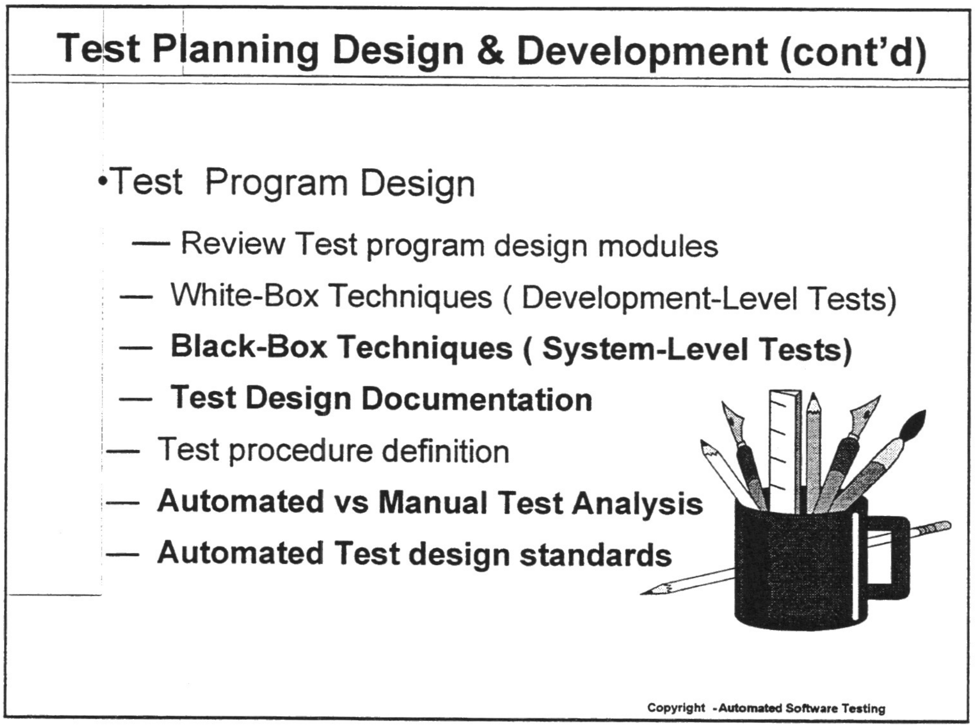
Caveat: It's not always feasible to come up with an extensive test plan

Test Planning Design & Development

- Test plan documentation
  - — Test Program Scope
  - — Test Requirement Management
  - — Test Environment

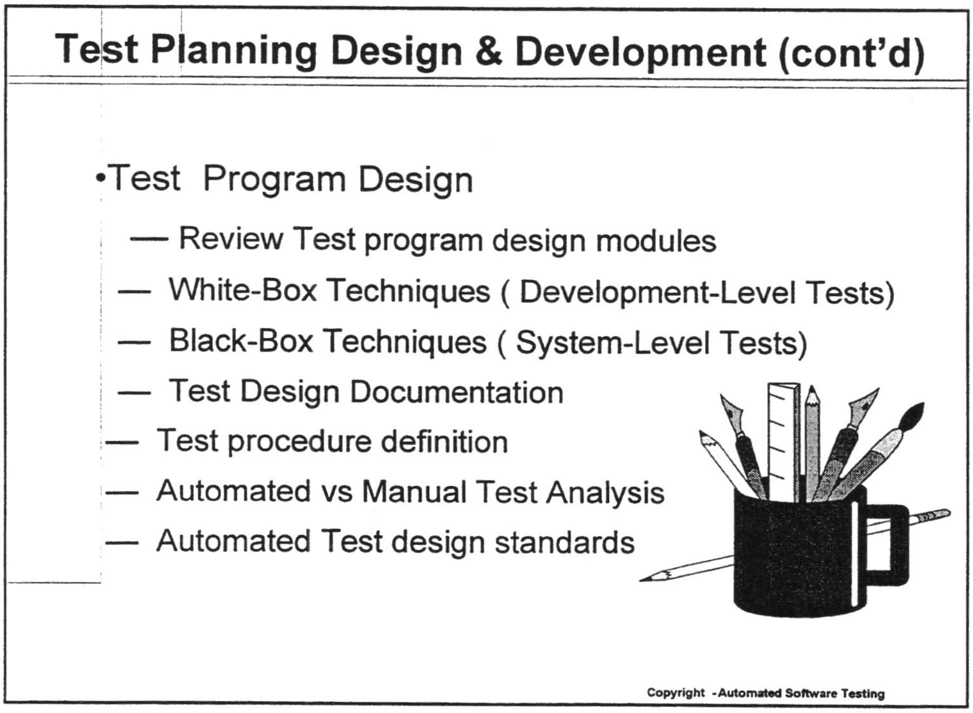


- Test requirements Analysis
  - — Development-Level Test Analysis
  - — System-Level Test Analysis



Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - **Black-Box Techniques (System-Level Tests)**
  - **Test Design Documentation**
  - Test procedure definition
  - **Automated vs Manual Test Analysis**
  - **Automated Test design standards**



Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - Black-Box Techniques (System-Level Tests)
  - Test Design Documentation
  - Test procedure definition
  - Automated vs Manual Test Analysis
  - Automated Test design standards

•Black-Box Techniques (System-Level Tests)

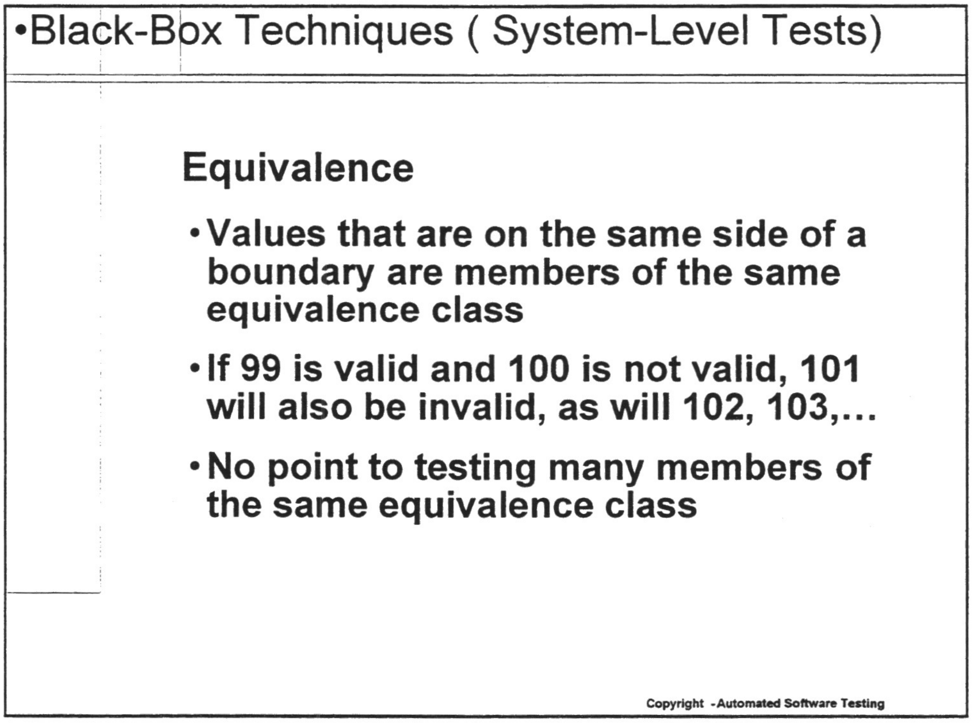
Functional Testing

- Main concepts
  - Based on requirements
  - Based on use cases
  - Discussed later.....

•Black-Box Techniques (System-Level Tests)

Boundary Value Testing

- Main concepts
  - Errors congregate at the boundaries between valid and invalid input
  - Tests using boundary values are highly effective
  - Tests of both valid and invalid input are needed
  - Closely related to equivalence partitioning

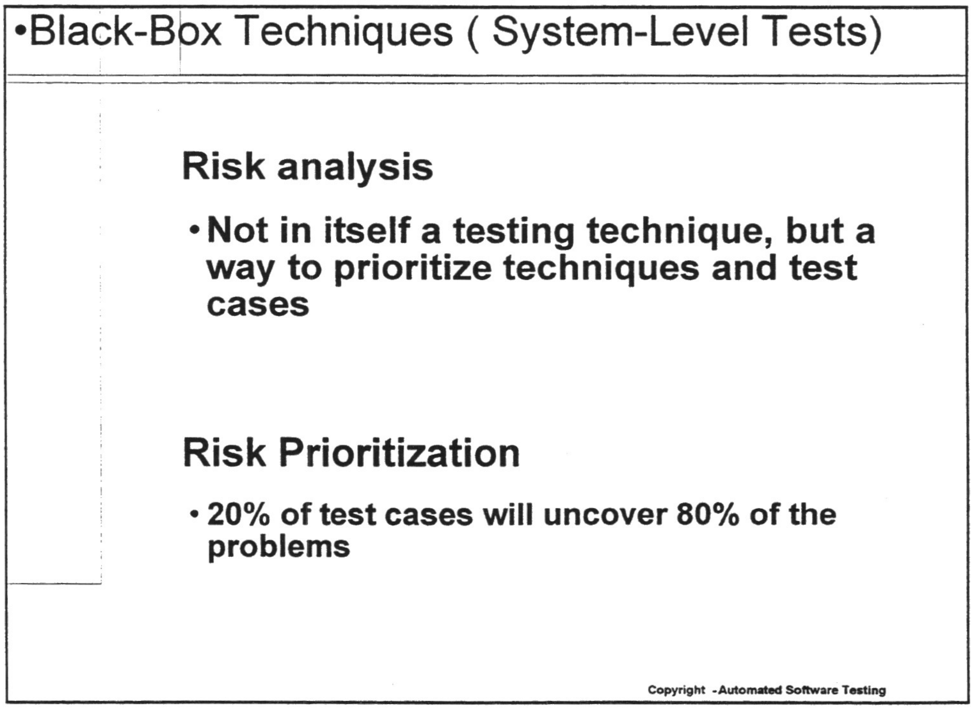


•Black-Box Techniques (System-Level Tests)

Equivalence

- **Values that are on the same side of a boundary are members of the same equivalence class**
- **If 99 is valid and 100 is not valid, 101 will also be invalid, as will 102, 103,...**
- **No point to testing many members of the same equivalence class**

Copyright -Automated Software Testing



•Black-Box Techniques (System-Level Tests)

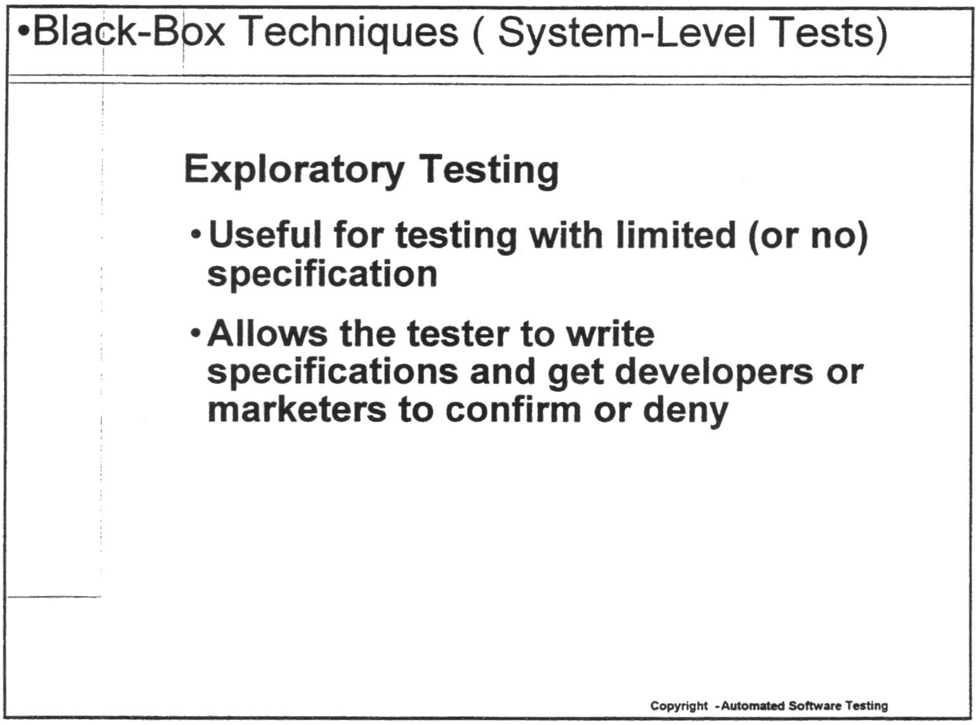
Risk analysis

- **Not in itself a testing technique, but a way to prioritize techniques and test cases**

Risk Prioritization

- **20% of test cases will uncover 80% of the problems**

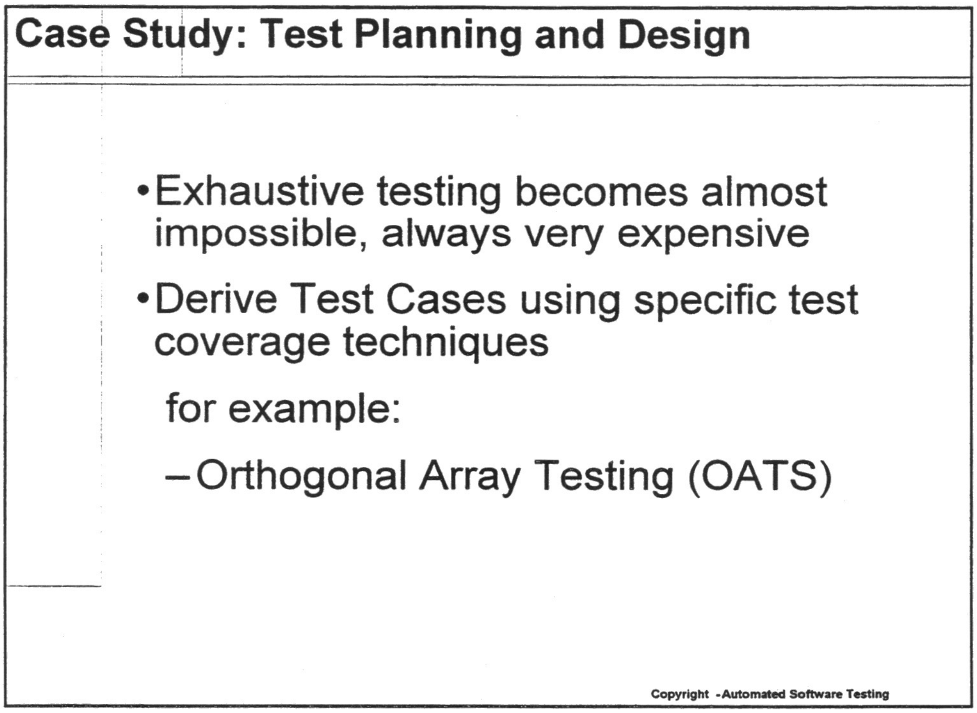
Copyright -Automated Software Testing



•Black-Box Techniques (System-Level Tests)

Exploratory Testing

- **Useful for testing with limited (or no) specification**
- **Allows the tester to write specifications and get developers or marketers to confirm or deny**

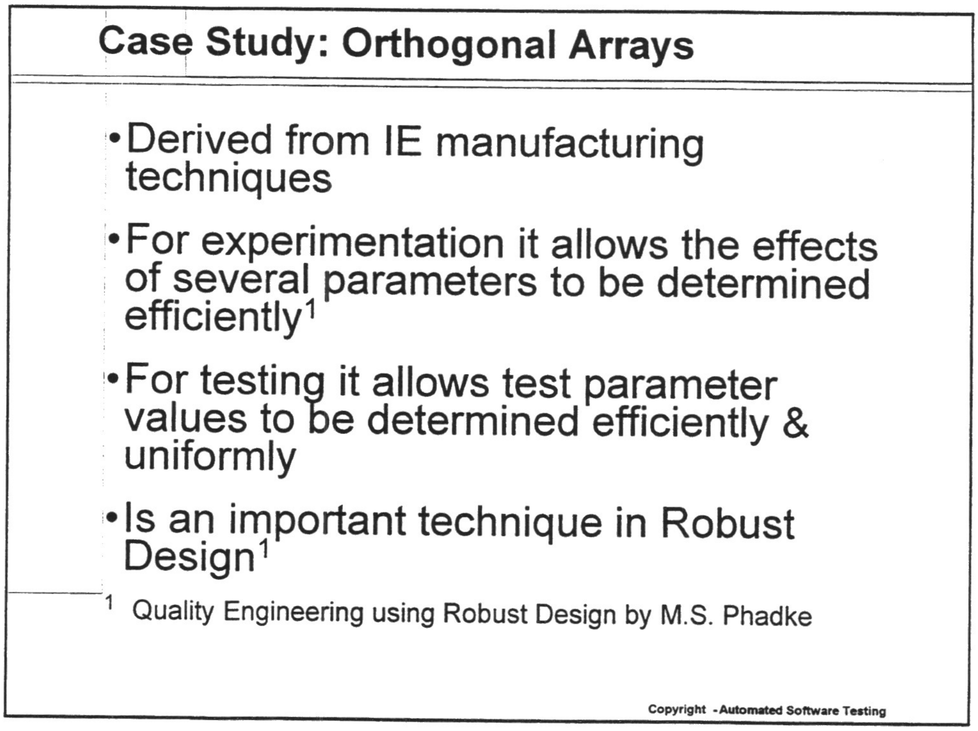


Case Study: Test Planning and Design

- Exhaustive testing becomes almost impossible, always very expensive
- Derive Test Cases using specific test coverage techniques

  for example:

  – Orthogonal Array Testing (OATS)

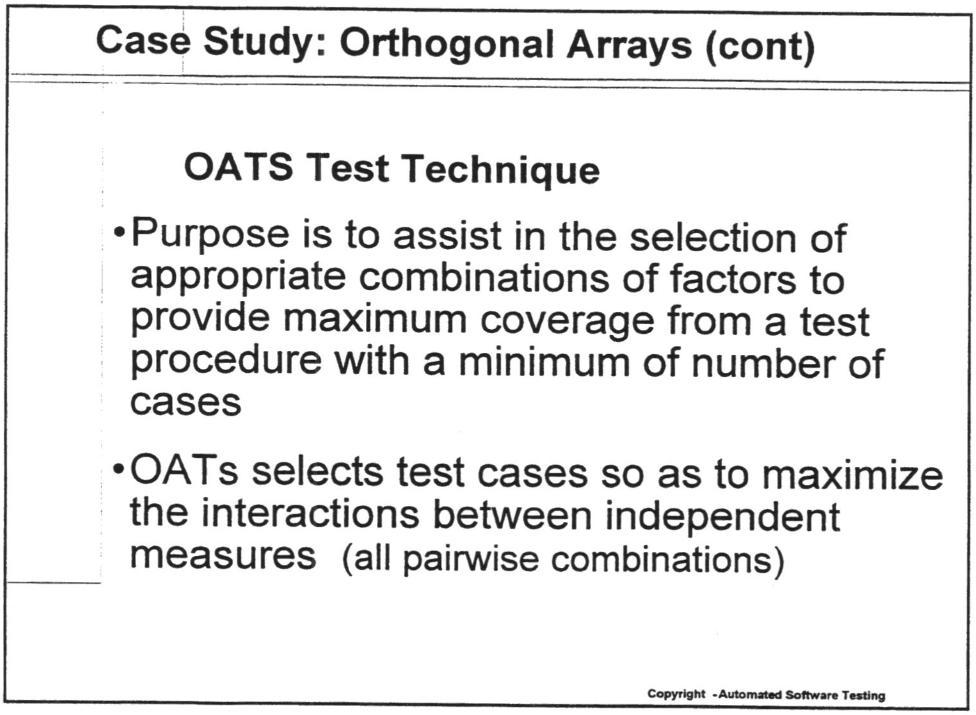


Case Study: Orthogonal Arrays

- Derived from IE manufacturing techniques
- For experimentation it allows the effects of several parameters to be determined efficiently[1]
- For testing it allows test parameter values to be determined efficiently & uniformly
- Is an important technique in Robust Design[1]

[1] Quality Engineering using Robust Design by M.S. Phadke

Copyright - Automated Software Testing



Case Study: Orthogonal Arrays (cont)

OATS Test Technique

- Purpose is to assist in the selection of appropriate combinations of factors to provide maximum coverage from a test procedure with a minimum of number of cases
- OATs selects test cases so as to maximize the interactions between independent measures (all pairwise combinations)

Copyright - Automated Software Testing

Case Study: Orthogonal Arrays (cont)

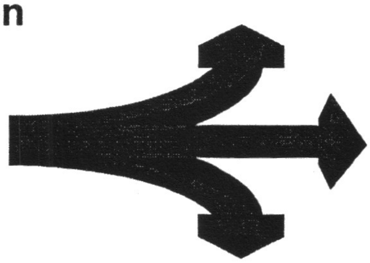
OATS Array Interpretation

- System parameters are array columns
  - Parameter values must be mapped to array number values
- Tester can chose valid values for unassigned levels in effort to minimize test
  - redundant test cases are eliminated
  - add cases based on known risk areas
- Each remaining row in the orthogonal array specifies one specific test case

Case Study: Orthogonal Arrays (cont)

Array Determination

- Max # of values or states for all parameters
- Number of parameters or factors

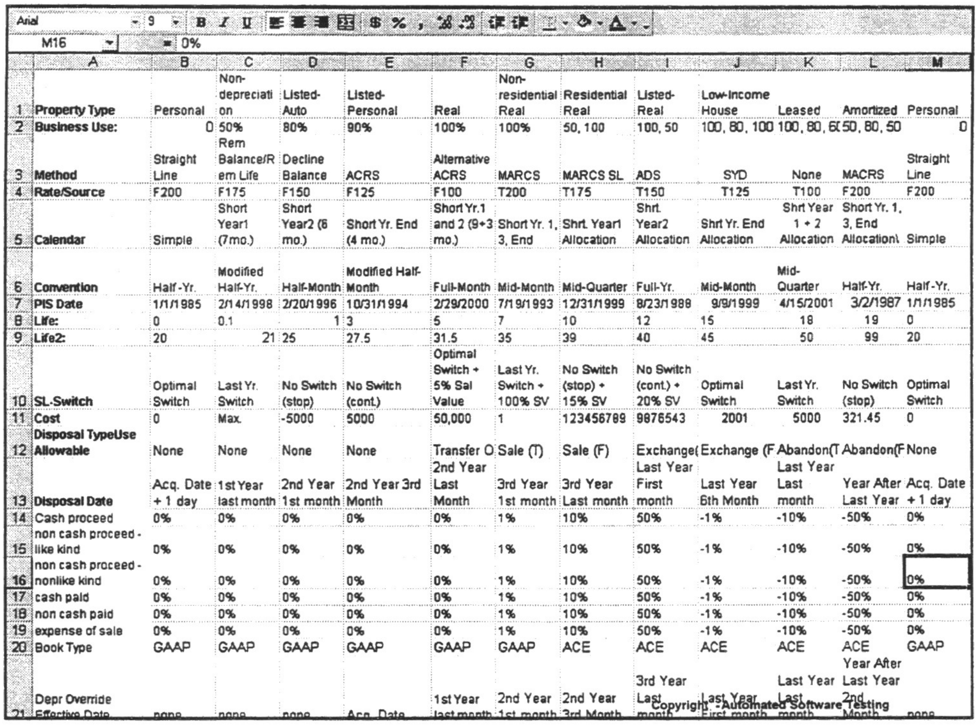


An Example Array

- Consider an example in which there are three factors and each factor has three levels.
- Testing all possible combinations of the three factors would require 27 test cases.
- But, testing pair-wise combinations will result in only 9 cases required to test the interactions of the independent factors

An Example Array

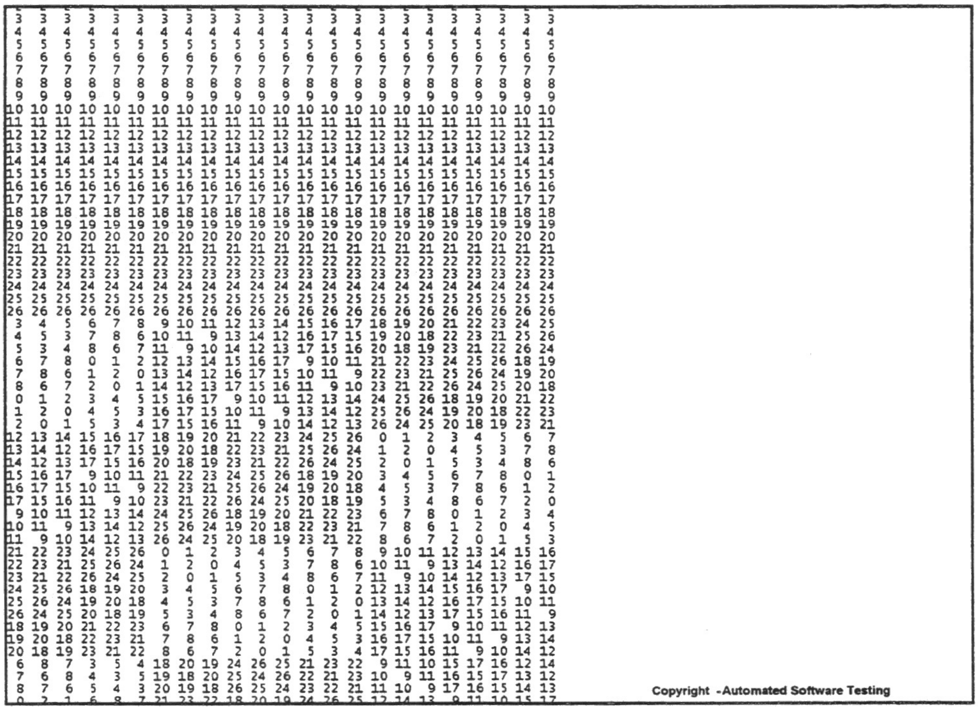
	A	B	C
1	1	1	3
2	1	2	2
3	1	3	1
4	2	1	2
5	2	2	1
6	2	3	3
7	3	1	1
8	3	2	3
9	3	3	2



M16 = 0%

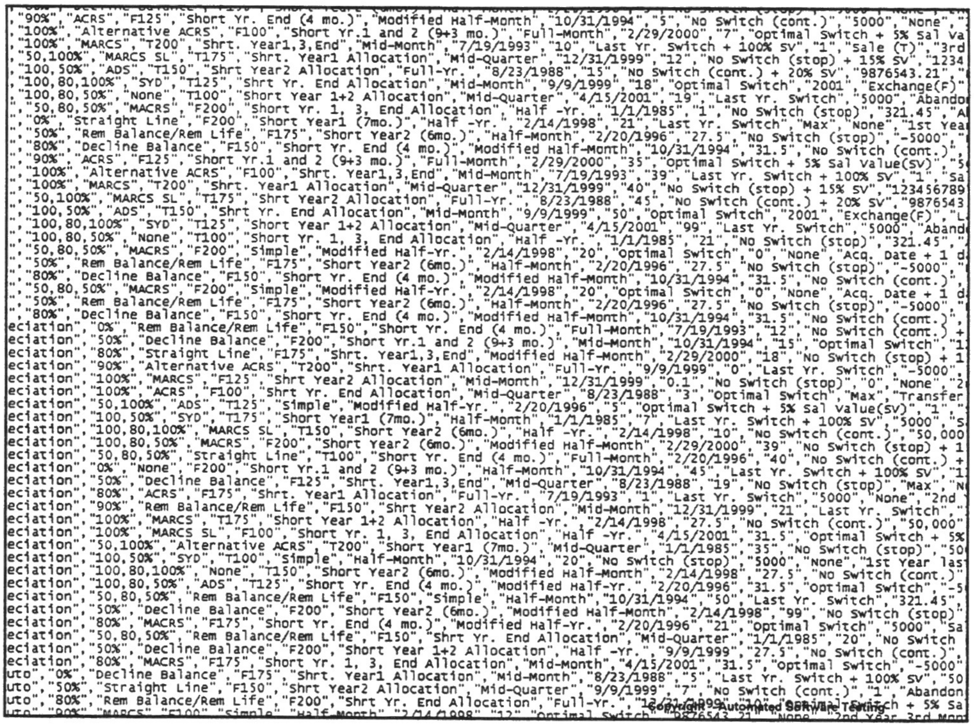
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Property Type**	Personal	Non-depreciation	Listed-Auto	Listed-Personal	Real	Non-residential Real	Residential Real	Listed-Real	Low-Income House	Leased	Amortized	Personal
2	**Business Use:**	0	50%	80%	90%	100%	100%	50, 100	100, 50	100, 80, 100	100, 80, 6(	50, 80, 50	0
3	**Method**	Straight Line	Rem Balance/Rem Life	Decline Balance	ACRS	Alternative ACRS	MARCS	MARCS SL	ADS	SYD	None	MACRS	Straight Line
4	**Rate/Source**	F200	F175	F150	F125	F100	T200	T175	T150	T125	T100	F200	F200
5	**Calendar**	Simple	Short Year1 (7mo.)	Short Year2 (6 mo.)	Short Yr. End (4 mo.)	Short Yr.1 and 2 (9+3 mo.)	Short Yr. 1, 3, End	Shrt. Year1 Allocation	Shrt. Year2 Allocation	Shrt Yr. End Allocation	Shrt Year 1 + 2 Allocation	Short Yr. 1, 3, End Allocation\	Simple
6	**Convention**	Half -Yr.	Modified Half-Yr.	Half-Month	Modified Half-Month	Full-Month	Mid-Month	Mid-Quarter	Full-Yr.	Mid-Month	Mid-Quarter	Half-Yr.	Half -Yr.
7	**PIS Date**	1/1/1985	2/14/1998	2/20/1996	10/31/1994	2/29/2000	7/19/1993	12/31/1999	8/23/1988	9/9/1999	4/15/2001	3/2/1987	1/1/1985
8	**Life:**	0	0.1	1	3	5	7	10	12	15	18	19	0
9	**Life2:**	20	21	25	27.5	31.5	35	39	40	45	50	99	20
10	**SL-Switch**	Optimal Switch	Last Yr. Switch	No Switch (stop)	No Switch (cont.)	Optimal Switch + 5% Sal Value	Last Yr. Switch + 100% SV	No Switch (stop) + 15% SV	No Switch (cont.) + 20% SV	Optimal Switch	Last Yr. Switch	No Switch (stop)	Optimal Switch
11	**Cost**	0	Max.	-5000	5000	50,000	1	123456789	9876543	2001	5000	321.45	0
12	**Disposal TypeUse Allowable**	None	None	None	None	Transfer O	Sale (T)	Sale (F)	Exchange(	Exchange (F	Abandon(T	Abandon(F	None
13	**Disposal Date**	Acq. Date + 1 day	1st Year last month	2nd Year 1st month	2nd Year 3rd Month	2nd Year Last Month	3rd Year 1st month	3rd Year Last month	Last Year First month	Last Year 6th Month	Last Year Last month	Year After Last Year	Acq. Date + 1 day
14	Cash proceed	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
15	non cash proceed - like kind	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
16	non cash proceed - nonlike kind	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
17	cash paid	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
18	non cash paid	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
19	expense of sale	0%	0%	0%	0%	0%	1%	10%	50%	-1%	-10%	-50%	0%
20	Book Type	GAAP	GAAP	GAAP	GAAP	GAAP	GAAP	ACE	ACE	ACE	ACE	ACE	GAAP
21	Depr Override Effective Date	none	none	none	Acq. Date	1st Year last month	2nd Year 1st month	2nd Year 3rd Month	3rd Year Last month	Last Year First month	Last Year Last month	Year After Last Year 2nd Month	none

Copyright - Automated Software Testing



```
  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3
  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21
 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26
  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
  4  5  3  7  8  6 10 11  9 13 14 12 16 17 15 19 20 18 22 23 21 25 26
  5  3  4  8  6  7 11  9 10 14 12 13 17 15 16 20 18 19 23 21 22 26 24
  6  7  8  0  1  2 12 13 14 15 16 17  9 10 11 21 22 23 24 25 26 18 19
  7  8  6  1  2  0 13 14 12 16 17 15 10 11  9 22 23 21 25 26 24 19 20
  8  6  7  2  0  1 14 12 13 17 15 16 11  9 10 23 21 22 26 24 25 20 18
  0  1  2  3  4  5 15 16 17  9 10 11 12 13 14 24 25 26 18 19 20 21 22
  1  2  0  4  5  3 16 17 15 10 11  9 13 14 12 25 26 24 19 20 18 22 23
  2  0  1  5  3  4 17 15 16 11  9 10 14 12 13 26 24 25 20 18 19 23 21
 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26  0  1  2  3  4  5  6  7
 13 14 12 16 17 15 19 20 18 22 23 21 25 26 24  1  2  0  4  5  3  7  8
 14 12 13 17 15 16 20 18 19 23 21 22 26 24 25  2  0  1  5  3  4  8  6
 15 16 17  9 10 11 21 22 23 24 25 26 18 19 20  3  4  5  6  7  8  0  1
 16 17 15 10 11  9 22 23 21 25 26 24 19 20 18  4  5  3  7  8  6  1  2
 17 15 16 11  9 10 23 21 22 26 24 25 20 18 19  5  3  4  8  6  7  2  0
  9 10 11 12 13 14 24 25 26 18 19 20 21 22 23  6  7  8  0  1  2  3  4
 10 11  9 13 14 12 25 26 24 19 20 18 22 23 21  7  8  6  1  2  0  4  5
 11  9 10 14 12 13 26 24 25 20 18 19 23 21 22  8  6  7  2  0  1  5  3
 21 22 23 24 25 26  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
 22 23 21 25 26 24  1  2  0  4  5  3  7  8  6 10 11  9 13 14 12 16 17
 23 21 22 26 24 25  2  0  1  5  3  4  8  6  7 11  9 10 14 12 13 17 15
 24 25 26 18 19 20  3  4  5  6  7  8  0  1  2 12 13 14 15 16 17  9 10
 25 26 24 19 20 18  4  5  3  7  8  6  1  2  0 13 14 12 16 17 15 10 11
 26 24 25 20 18 19  5  3  4  8  6  7  2  0  1 14 12 13 17 15 16 11  9
 18 19 20 21 22 23  6  7  8  0  1  2  3  4  5 15 16 17  9 10 11 12 13
 19 20 18 22 23 21  7  8  6  1  2  0  4  5  3 16 17 15 10 11  9 13 14
 20 18 19 23 21 22  8  6  7  2  0  1  5  3  4 17 15 16 11  9 10 14 12
  6  8  7  3  5  4 18 20 19 24 26 25 21 23 22  9 11 10 15 17 16 12 14
  7  6  8  4  3  5 19 18 20 25 24 26 22 21 23 10  9 11 16 15 17 13 12
  8  7  6  5  4  3 20 19 18 26 25 24 23 22 21 11 10  9 17 16 15 14 13
  0  2  1  6  8  7 21 23 22 18 20 19 24 26 25 12 14 13  9 11 10 15 17
```

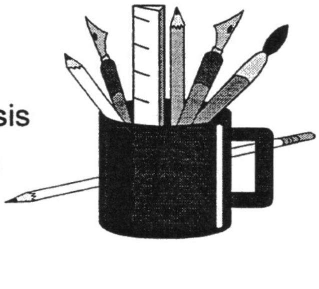
Copyright - Automated Software Testing

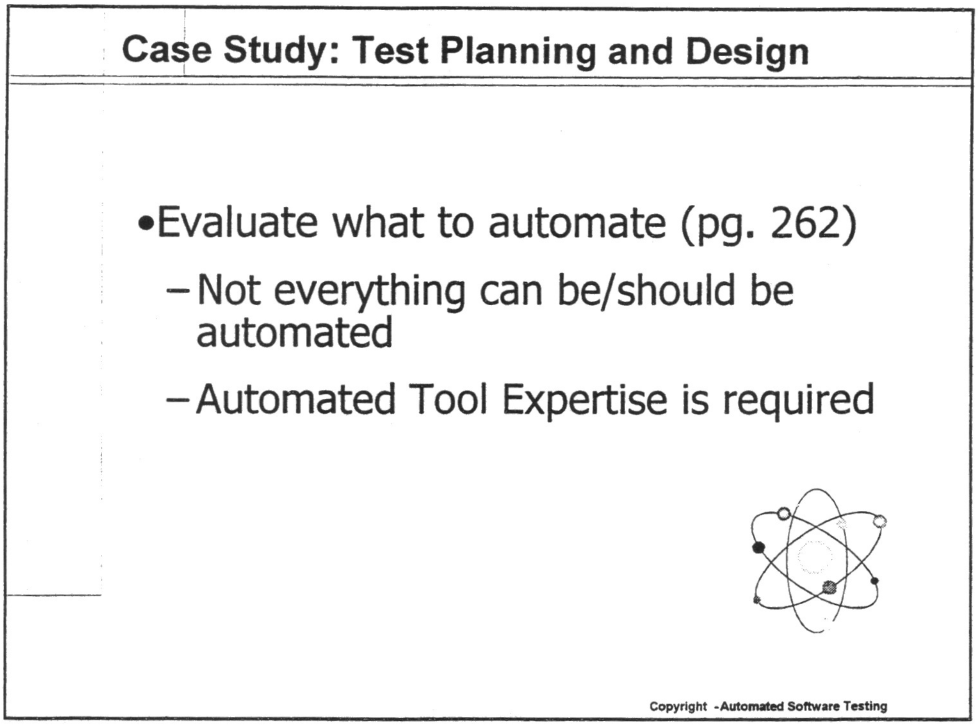


```
,"90%","ACRS","F125","Short Yr. End (4 mo.)","Modified Half-Month","10/31/1994","5","No Switch (cont.)","5000","None","
,"100%","Alternative ACRS","F100","Short Yr.1 and 2 (9+3 mo.)","Full-Month","2/29/2000","7","Optimal Switch + 5% Sal Va
,"100%","MARCS","T200","Shrt. Year1,3,End","Mid-Month","7/19/1993","10","Last Yr. Switch + 100% SV","1","Sale (T)","3rd
,"50,100%","MARCS SL","T175","Shrt. Year1 Allocation","Mid-Quarter","12/31/1999","12","No Switch (stop) + 15% SV","1234
,"100,50%","ADS","T150","Shrt Year2 Allocation","Full-Yr.","8/23/1988","15","No Switch (cont.) + 20% SV","9876543.21","
,"100,80,100%","SYD","T125","Shrt Yr. End Allocation","Mid-Month","9/9/1999","18","Optimal Switch","2001","Exchange(F)"
,"100,80,50%","None","T100","Short Year 1+2 Allocation","Mid-Quarter","4/15/2001","19","Last Yr. Switch","5000","Abando
","50,80,50%","MACRS","F200","Short Yr. 1, 3, End Allocation","Half -Yr.","1/1/1985","1","No Switch (stop)","321.45","A
","0%","Straight Line","F200","Short Year1 (7mo.)","Half -Yr.","2/14/1998","21","Last Yr. Switch","Max","None","1st Yea
","50%","Rem Balance/Rem Life","F175","Short Year2 (6mo.)","Half-Month","2/20/1996","27.5","No Switch (stop)","-5000","
","80%","Decline Balance","F150","Short Yr. End (4 mo.)","Modified Half-Month","10/31/1994","31.5","No Switch (cont.)",
","90%","ACRS","F125","Short Yr.1 and 2 (9+3 mo.)","Full-Month","2/29/2000","35","Optimal Switch + 5% Sal Value(SV)","5
","100%","Alternative ACRS","F100","Shrt. Year1,3,End","Mid-Month","7/19/1993","39","Last Yr. Switch + 100% SV","1","Sa
","100%","MARCS","T200","Shrt. Year1 Allocation","Mid-Quarter","12/31/1999","40","No Switch (stop) + 15% SV","123456789
","50,100%","MARCS SL","T175","Shrt Year2 Allocation","Full-Yr.","8/23/1988","45","No Switch (cont.) + 20% SV","9876543
","100,50%","ADS","T150","Shrt Yr. End Allocation","Mid-Month","9/9/1999","50","Optimal Switch","2001","Exchange(F)","L
","100,80,100%","SYD","T125","Short Year 1+2 Allocation","Mid-Quarter","4/15/2001","99","Last Yr. Switch","5000","Aband
","100,80,50%","None","T100","Short Yr. 1, 3, End Allocation","Half -Yr.","1/1/1985","21","No Switch (stop)","321.45","
","50,80,50%","MACRS","F200","Simple","Modified Half-Yr.","2/14/1998","20","Optimal Switch","0","None","Acq. Date + 1 d
","50%","Rem Balance/Rem Life","F175","Short Year2 (6mo.)","Half-Month","2/20/1996","27.5","No Switch (stop)","-5000","
","80%","Decline Balance","F150","Short Yr. End (4 mo.)","Modified Half-Month","10/31/1994","31.5","No Switch (cont.)",
","50,80,50%","MACRS","F200","Simple","Modified Half-Yr.","2/14/1998","20","Optimal Switch","0","None","Acq. Date + 1 d
","50%","Rem Balance/Rem Life","F175","Short Year2 (6mo.)","Half-Month","2/20/1996","27.5","No Switch (stop)","-5000","
","80%","Decline Balance","F150","Short Yr. End (4 mo.)","Modified Half-Month","10/31/1994","31.5","No Switch (cont.)",
eciation","0%","Rem Balance/Rem Life","F150","Short Yr. End (4 mo.)","Full-Month","7/19/1993","12","No Switch (cont.) +
eciation","50%","Decline Balance","F200","Short Yr.1 and 2 (9+3 mo.)","Mid-Month","10/31/1994","15","Optimal Switch","1
eciation","80%","Straight Line","F175","Shrt. Year1,3,End","Modified Half-Month","2/29/2000","18","No Switch (stop) + 1
eciation","90%","Alternative ACRS","T200","Shrt. Year1 Allocation","Full-Yr.","9/9/1999","0","Last Yr. Switch","-5000",
eciation","100%","MARCS","F125","Shrt Year2 Allocation","Mid-Month","12/31/1999","0.1","No Switch (stop)","0","None","2
eciation","100%","ACRS","F100","Shrt Yr. End Allocation","Mid-Quarter","8/23/1988","3","Optimal Switch","Max","Transfer
eciation","50,100%","ADS","T125","Simple","Modified Half-Yr.","2/20/1996","5","Optimal Switch + 5% Sal Value(SV)","1","
eciation","100,50%","SYD","T175","Short Year1 (7mo.)","Half-Month","1/1/1985","7","Last Yr. Switch + 100% SV","5000","S
eciation","100,80,100%","MARCS SL","T150","Short Year2 (6mo.)","Half -Yr.","2/14/1998","10","No Switch (cont.)","50,000
eciation","100,80,50%","MACRS","F200","Short Year2 (6mo.)","Modified Half-Month","2/29/2000","39","No Switch (stop) + 1
eciation","50,80,50%","Straight Line","T100","Short Yr. End (4 mo.)","Full-Month","2/20/1996","40","No Switch (cont.) +
eciation","0%","None","F200","Short Yr.1 and 2 (9+3 mo.)","Half-Month","10/31/1994","45","Last Yr. Switch + 100% SV","1
eciation","50%","Decline Balance","F125","Shrt. Year1,3,End","Mid-Quarter","8/23/1988","19","No Switch (stop)","Max","N
eciation","80%","ACRS","F175","Shrt. Year1 Allocation","Full-Yr.","7/19/1993","1","Last Yr. Switch","5000","None","2nd
eciation","90%","Rem Balance/Rem Life","F150","Shrt Year2 Allocation","Mid-Month","12/31/1999","21","Last Yr. Switch","
eciation","100%","MARCS","T175","Short Year 1+2 Allocation","Half -Yr.","2/14/1998","27.5","No Switch (cont.)","50,000"
eciation","100%","MARCS SL","F100","Short Yr. 1, 3, End Allocation","Half -Yr.","4/15/2001","31.5","Optimal Switch + 5%
eciation","50,100%","Alternative ACRS","T200","Short Year1 (7mo.)","Mid-Quarter","1/1/1985","35","No Switch (stop)","50
eciation","100,50%","SYD","T100","Simple","Half-Month","10/31/1994","20","No Switch (stop)","5000","None","1st Year las
eciation","100,80,100%","None","T150","Short Year2 (6mo.)","Modified Half-Month","2/14/1998","27.5","No Switch (cont.)"
eciation","100,80,50%","ADS","T125","Short Yr. End (4 mo.)","Modified Half-Yr.","2/20/1996","31.5","Optimal Switch","-5
eciation","50,80,50%","Rem Balance/Rem Life","F150","Simple","Half-Month","10/31/1994","50","Last Yr. Switch","321.45",
eciation","50%","Decline Balance","F200","Short Year2 (6mo.)","Modified Half-Month","2/14/1998","99","No Switch (stop)"
eciation","80%","MACRS","F175","Short Yr. End (4 mo.)","Modified Half-Yr.","2/20/1996","21","Optimal Switch","5000","Sa
eciation","50,80,50%","Rem Balance/Rem Life","F150","Shrt Yr. End Allocation","Mid-Quarter","1/1/1985","20","No Switch
eciation","50%","Decline Balance","F200","Short Year 1+2 Allocation","Half -Yr.","9/9/1999","27.5","No Switch (cont.)",
eciation","80%","MACRS","F175","Short Yr. 1, 3, End Allocation","Mid-Month","4/15/2001","31.5","Optimal Switch","-5000"
uto","0%","Decline Balance","F175","Shrt. Year1 Allocation","Mid-Month","8/23/1988","5","Last Yr. Switch + 100% SV","50
uto","50%","Straight Line","F150","Shrt Year2 Allocation","Mid-Quarter","9/9/1999","7","No Switch (cont.)","1","Abandon
uto","80%","Rem Balance/Rem Life","F200","Shrt Yr. End Allocation","Full-Yr.","12/31/1999","10","Optimal Switch + 5% Sa
uto","90%","MARCS","F100","Simple","Half-Month","2/14/1998","12","Optimal Switch","9876543.21","None","2nd Year 3rd Mon
```

Test Planning Design & Development (cont'd)

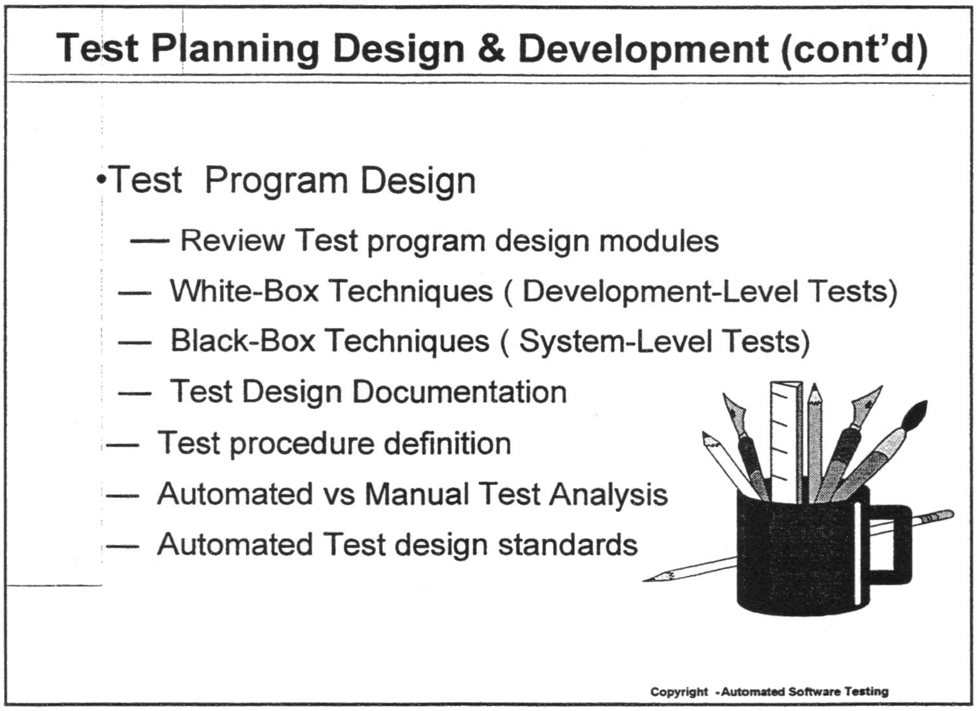
- Test Program Design
  - — Review Test program design modules
  - — White-Box Techniques (Development-Level Tests)
  - — Black-Box Techniques (System-Level Tests)
  - — Test Design Documentation
  - — Test procedure definition
  - — Automated vs Manual Test Analysis
  - — Automated Test design standards





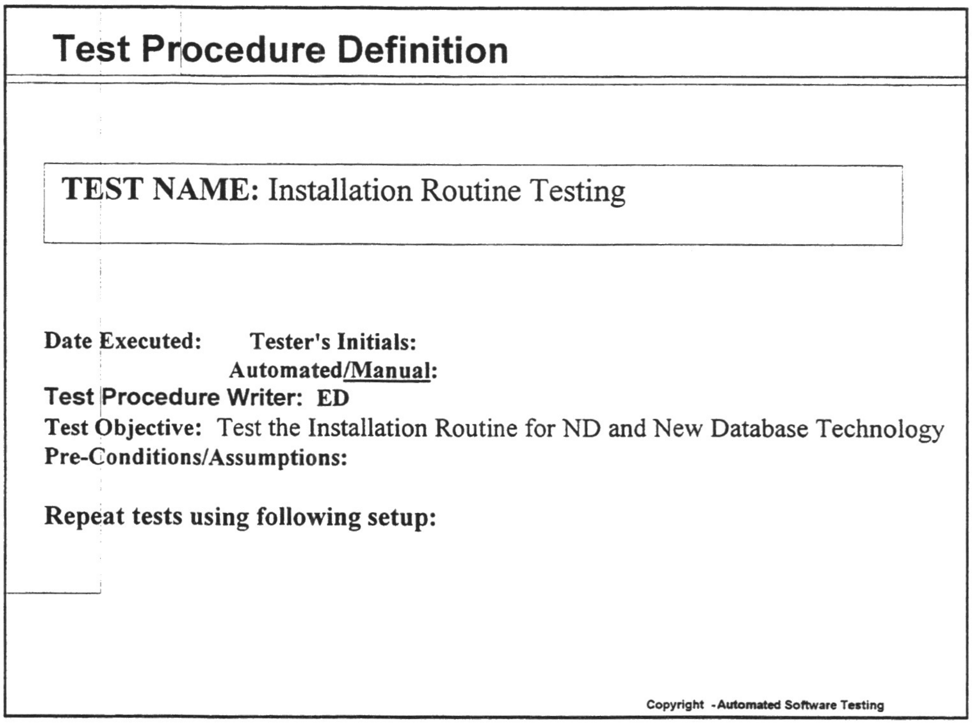
Case Study: Test Planning and Design

- Evaluate what to automate (pg. 262)
  - Not everything can be/should be automated
  - Automated Tool Expertise is required



Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - Black-Box Techniques (System-Level Tests)
  - Test Design Documentation
  - Test procedure definition
  - Automated vs Manual Test Analysis
  - Automated Test design standards



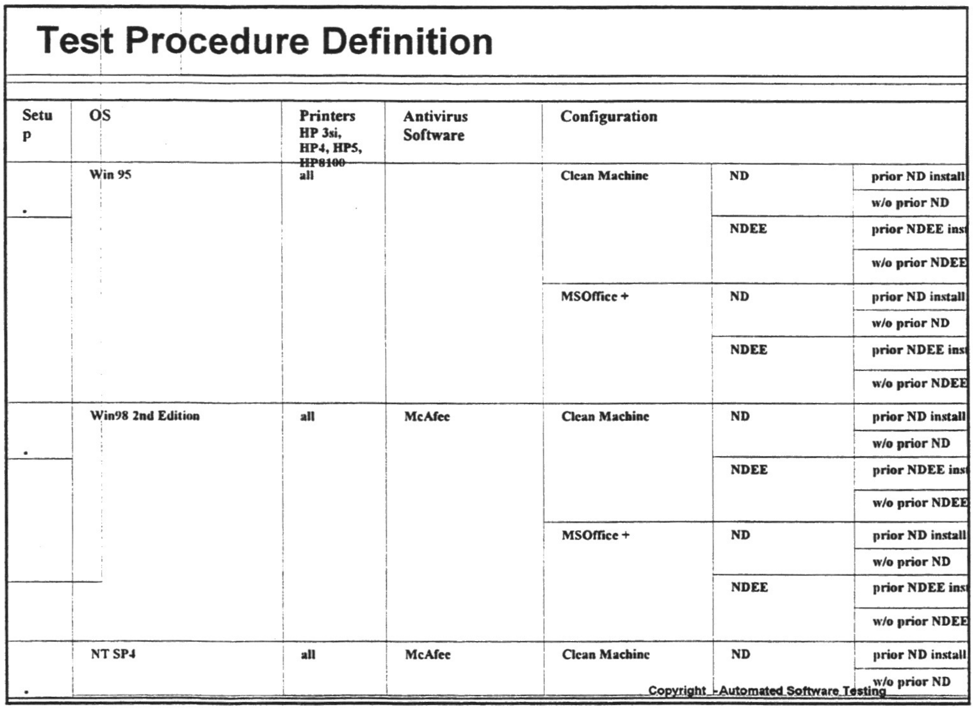
Test Procedure Definition

TEST NAME: Installation Routine Testing

Date Executed: **Tester's Initials:**
Automated/<u>Manual</u>:
Test Procedure Writer: ED
Test Objective: Test the Installation Routine for ND and New Database Technology
Pre-Conditions/Assumptions:

Repeat tests using following setup:

Copyright - Automated Software Testing



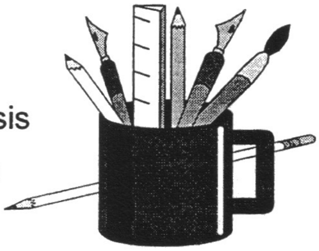
Test Procedure Definition

Setup	OS	Printers HP 3si, HP4, HP5, ~~HP8100~~	Antivirus Software	Configuration		
•	Win 95	all		Clean Machine	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE ins
						w/o prior NDEE
				MSOffice +	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE ins
						w/o prior NDEE
•	Win98 2nd Edition	all	McAfee	Clean Machine	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE ins
						w/o prior NDEE
				MSOffice +	ND	prior ND install
						w/o prior ND
					NDEE	prior NDEE ins
						w/o prior NDEE
•	NT SP4	all	McAfee	Clean Machine	ND	prior ND install
						w/o prior ND

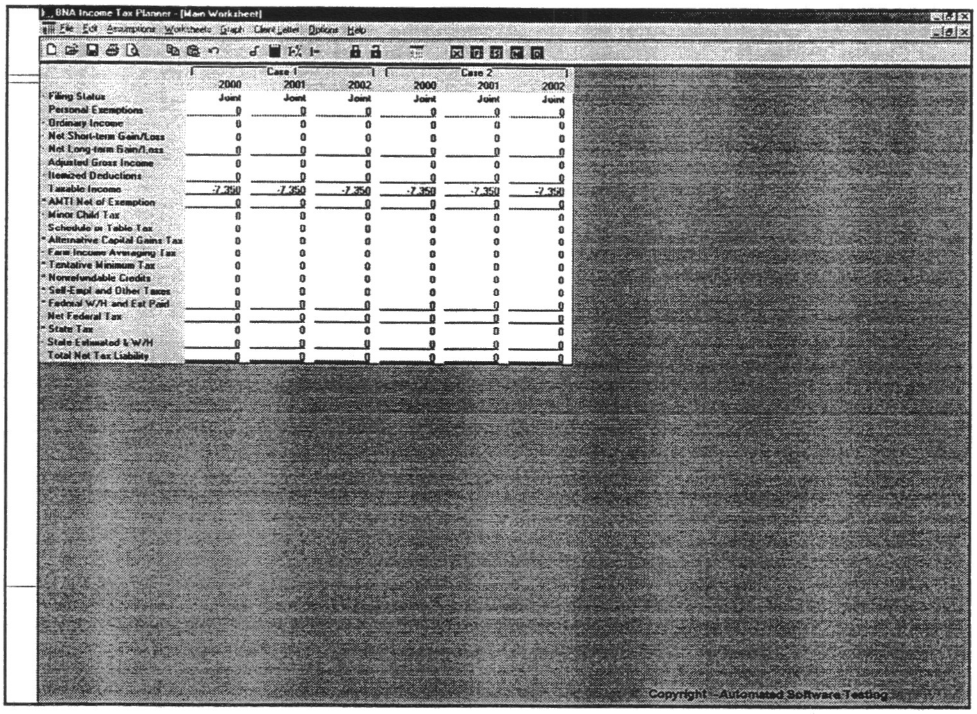
Copyright - Automated Software Testing

Test Planning Design & Development (cont'd)

- Test Program Design
  - Review Test program design modules
  - White-Box Techniques (Development-Level Tests)
  - Black-Box Techniques (System-Level Tests)
  - Test Design Documentation
  - Test procedure definition
  - Automated vs Manual Test Analysis
  - Automated Test design standards

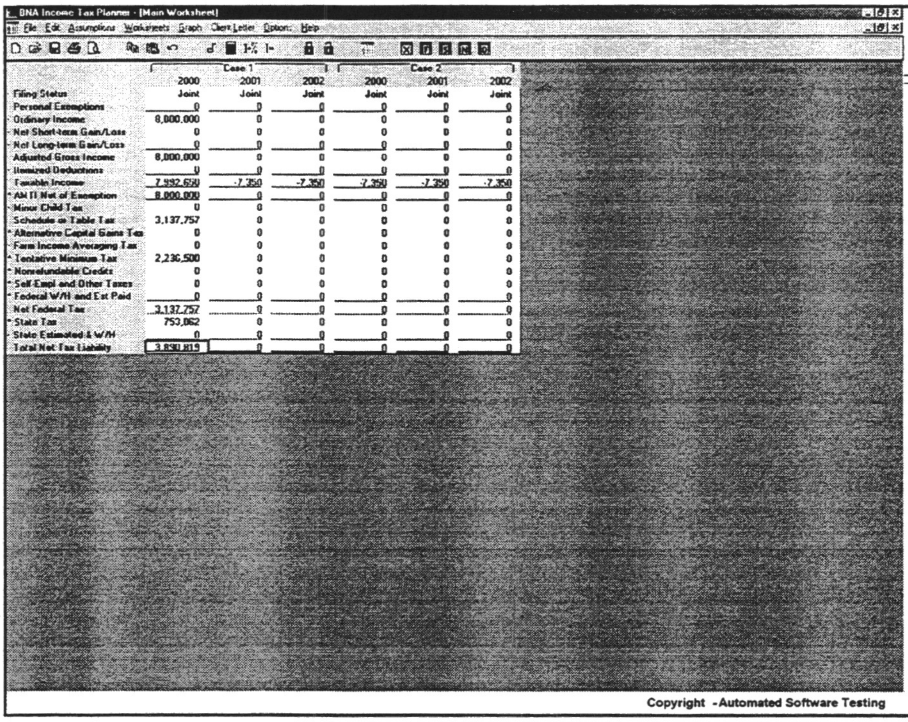


Copyright -Automated Software Testing



	Case 1			Case 2		
	2000	2001	2002	2000	2001	2002
Filing Status	Joint	Joint	Joint	Joint	Joint	Joint
Personal Exemptions	0	0	0	0	0	0
Ordinary Income	0	0	0	0	0	0
Net Short-term Gain/Loss	0	0	0	0	0	0
Net Long-term Gain/Loss	0	0	0	0	0	0
Adjusted Gross Income	0	0	0	0	0	0
Itemized Deductions	0	0	0	0	0	0
Taxable Income	-7,350	-7,350	-7,350	-7,350	-7,350	-7,350
* AMTI Net of Exemption	0	0	0	0	0	0
Minor Child Tax	0	0	0	0	0	0
Schedule or Table Tax	0	0	0	0	0	0
* Alternative Capital Gains Tax	0	0	0	0	0	0
Farm Income Averaging Tax	0	0	0	0	0	0
* Tentative Minimum Tax	0	0	0	0	0	0
* Nonrefundable Credits	0	0	0	0	0	0
* Self-Empl and Other Taxes	0	0	0	0	0	0
* Federal W/H and Est Paid	0	0	0	0	0	0
Net Federal Tax	0	0	0	0	0	0
* State Tax	0	0	0	0	0	0
State Estimated & W/H	0	0	0	0	0	0
Total Net Tax Liability	0	0	0	0	0	0

Copyright -Automated Software Testing

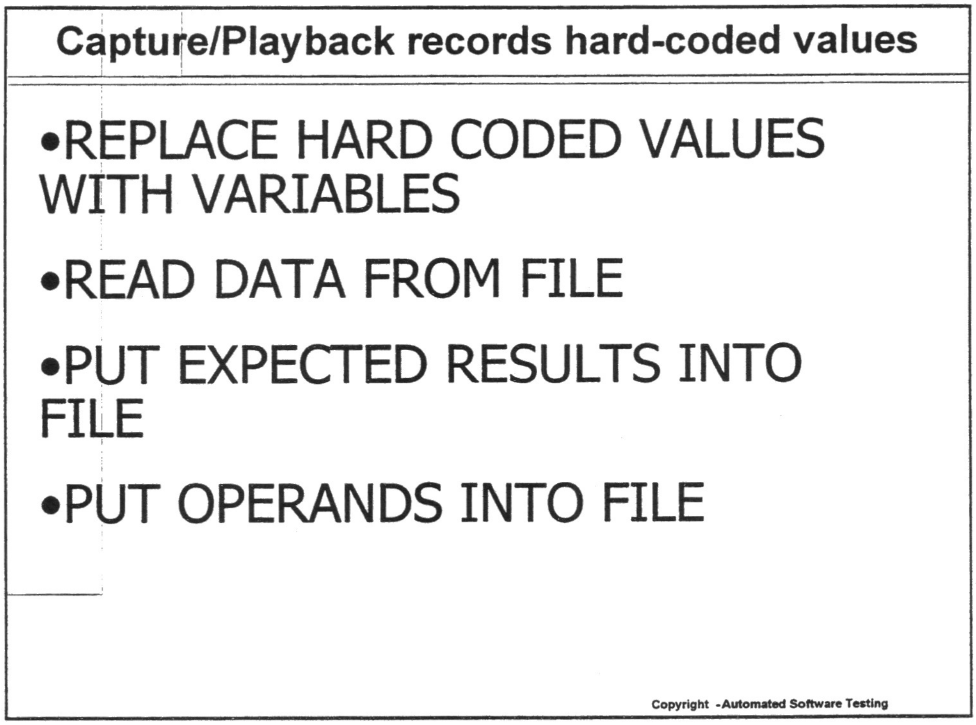

BNA Income Tax Planner - [Main Worksheet]
File Edit Assumptions Worksheets Graph Client Letter Options Help

	Case 1			Case 2		
	2000	2001	2002	2000	2001	2002
Filing Status	Joint	Joint	Joint	Joint	Joint	Joint
Personal Exemptions	0	0	0	0	0	0
Ordinary Income	8,000,000	0	0	0	0	0
Net Short-term Gain/Loss	0	0	0	0	0	0
Net Long-term Gain/Loss	0	0	0	0	0	0
Adjusted Gross Income	8,000,000	0	0	0	0	0
Itemized Deductions	0	0	0	0	0	0
Taxable Income	7,992,650	-7,350	-7,350	-7,350	-7,350	-7,350
*AMTI Net of Exemption	8,000,000	0	0	0	0	0
Minor Child Tax	0	0	0	0	0	0
Schedule or Table Tax	3,137,757	0	0	0	0	0
*Alternative Capital Gains Tax	0	0	0	0	0	0
Farm Income Averaging Tax	0	0	0	0	0	0
*Tentative Minimum Tax	2,236,500	0	0	0	0	0
*Nonrefundable Credits	0	0	0	0	0	0
*Self Empl and Other Taxes	0	0	0	0	0	0
*Federal W/H and Est Paid	0	0	0	0	0	0
Net Federal Tax	3,137,757	0	0	0	0	0
*State Tax	753,062	0	0	0	0	0
State Estimated & W/H	0	0	0	0	0	0
Total Net Tax Liability	3,890,819	0	0	0	0	0

Capture/Playback records hard-coded values

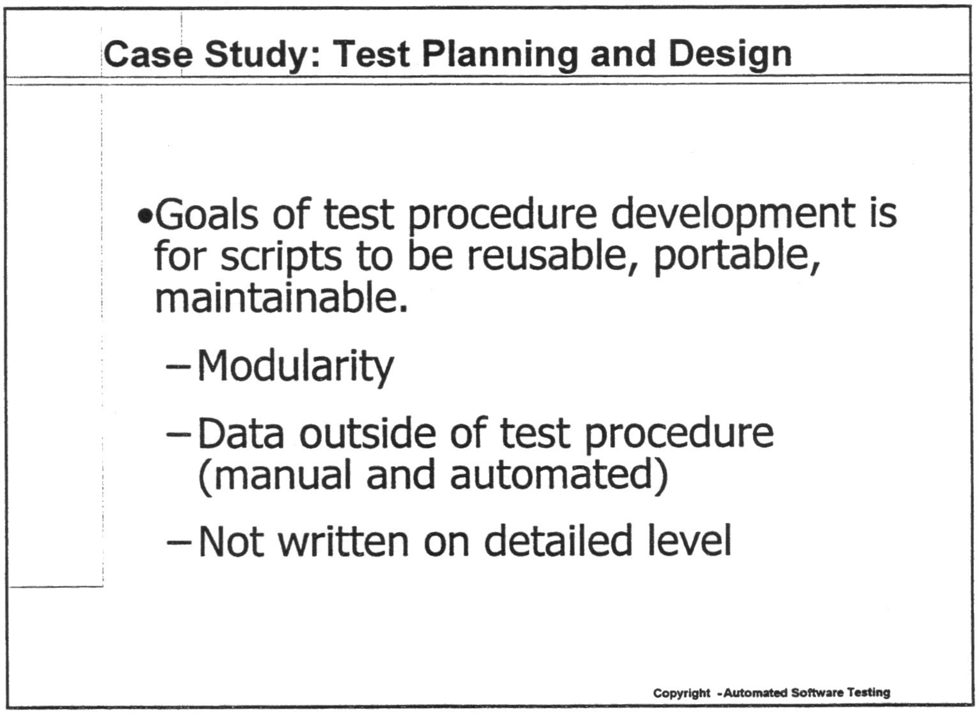
- ....
- Window SetContext, "Name=frmMDI", ""
-
- Window SetContext, "Name=frmWsTemplate", ""
- GenericObject Click, "Name=Spread", "Coords=213,138"
-
- Window SetContext, "Name=frmMDI", ""
- GenericObject Click, "Class=MDIClient;ClassIndex=1", "Coords=194,474"
-
- Window SetContext, "Name=frmWsTemplate", ""
- GenericObject DblClick, "Name=Spread", "Coords=212,64"
- InputKeys "**8000000**"
- GenericObject Click, "Name=Spread", "Coords=297,58"

- End Sub



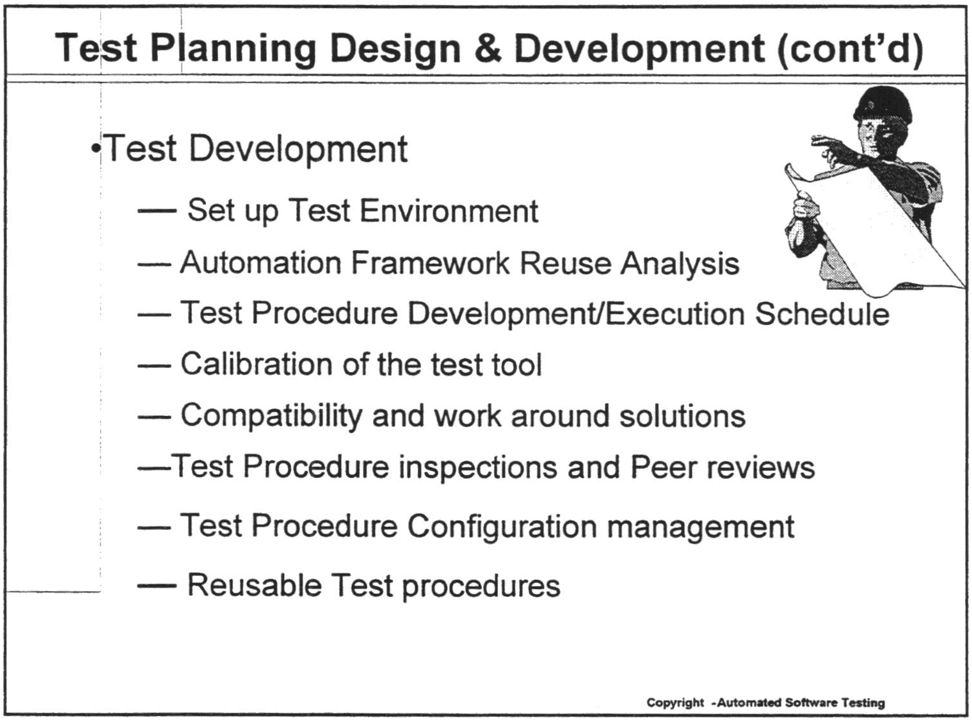
Capture/Playback records hard-coded values

- •REPLACE HARD CODED VALUES WITH VARIABLES
- •READ DATA FROM FILE
- •PUT EXPECTED RESULTS INTO FILE
- •PUT OPERANDS INTO FILE



Case Study: Test Planning and Design

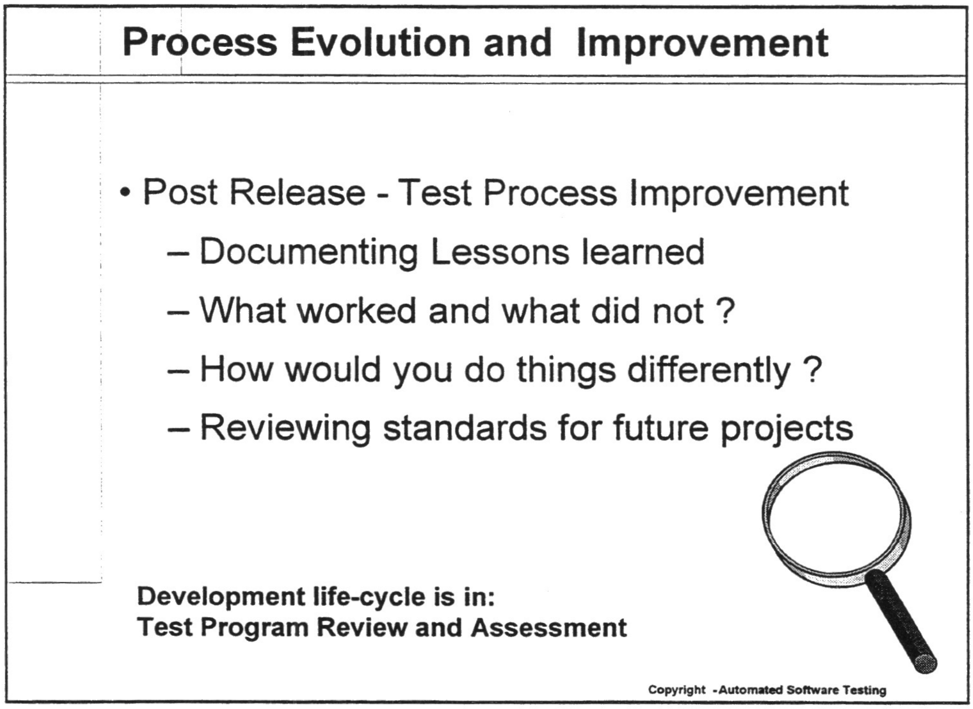
- •Goals of test procedure development is for scripts to be reusable, portable, maintainable.
  - –Modularity
  - –Data outside of test procedure (manual and automated)
  - –Not written on detailed level



Test Planning Design & Development (cont'd)

- Test Development
  - — Set up Test Environment
  - — Automation Framework Reuse Analysis
  - — Test Procedure Development/Execution Schedule
  - — Calibration of the test tool
  - — Compatibility and work around solutions
  - —Test Procedure inspections and Peer reviews
  - — Test Procedure Configuration management
  - — Reusable Test procedures

Copyright -Automated Software Testing



Process Evolution and Improvement

- Post Release - Test Process Improvement
  - – Documenting Lessons learned
  - – What worked and what did not ?
  - – How would you do things differently ?
  - – Reviewing standards for future projects

Development life-cycle is in:
Test Program Review and Assessment

Copyright -Automated Software Testing

ATLM

ATLM will help sort out tool issues

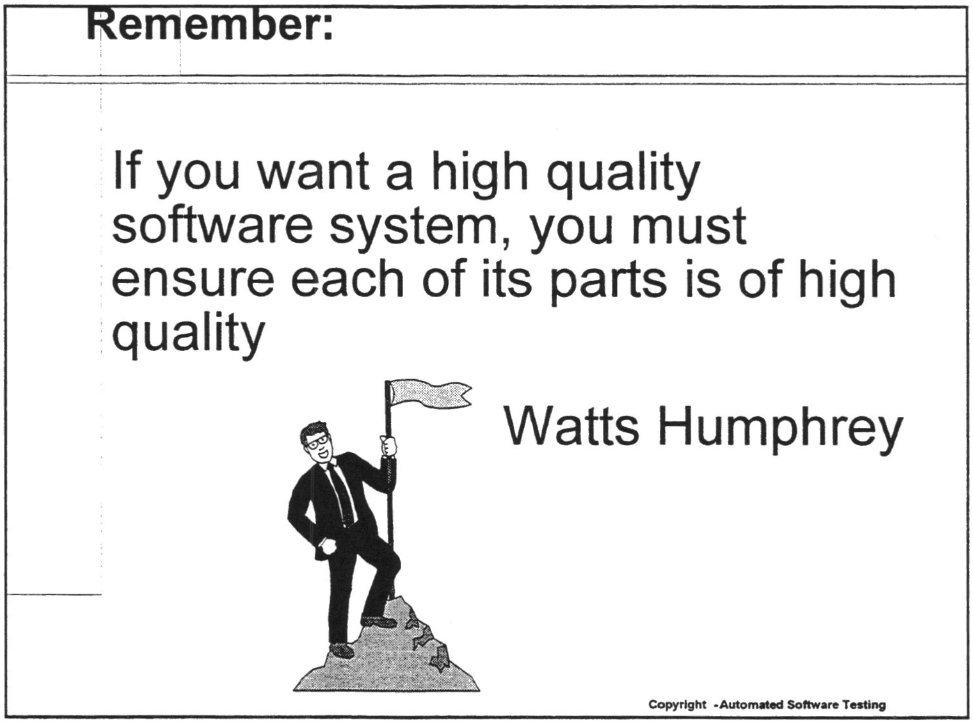
- What tools are available
- How to convince management to buy the tool
- How to evaluate tools
- How to incorporate tools into project and how to manage automated testing process
- Automated Test Design, Development, Execution
- Lessons Learned

details described in book "Automated Software Testing"

ATLM - Summary:

- Automated Testing needs to be parallel to system development
- Evaluate Testing Tools based on your environment and circumstances
- Training, training, training
- Manage expectations

•see www.stqemagazine.com (Sep/Oct '99) article on "Automated Testing Lessons Learned"

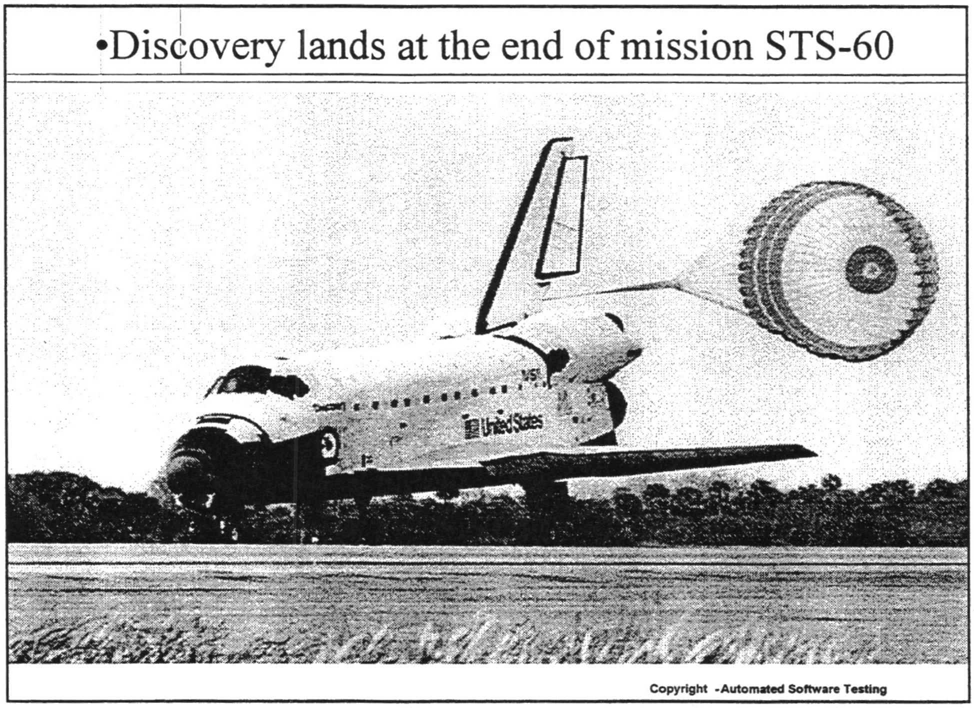


Remember:

If you want a high quality software system, you must ensure each of its parts is of high quality

Watts Humphrey

Copyright - Automated Software Testing



•Discovery lands at the end of mission STS-60

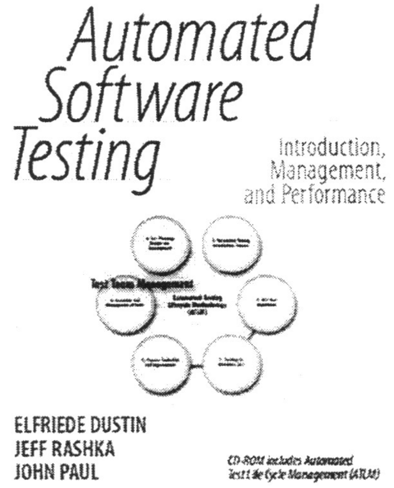
Copyright - Automated Software Testing

Workshop 2

Automated Testing Life-cycle Methodology

Copyright -Automated Software Testing

Book: Automated Software Testing



Automated Software Testing

Introduction, Management, and Performance

ELFRIEDE DUSTIN
JEFF RASHKA
JOHN PAUL

CD-ROM includes Automated Test Life Cycle Management (ATLM)

Lucky

Winner

Is.....

http://www.autotestco.com

Copyright -Automated Software Testing

QW2001 Workshop W3

Mr. Robert A. Sabourin
(AmiBug.Com)

Bug Priority And Severity

Key Points

- Bugs
- SQA Management
- SQA Process

Presentation Abstract

In this interactive half day tutorial the concepts of defect priority and severity are explored.

The fundamental question in software engineering "How do you know when you are finished?" is examined.

The journey begins on a freezing cold Canadian Winter day on an elevator ride during which Robert Sabourin accidentally overheads some strangers discussing problems with the most important software project taking place in the company! The class invited to help Robert in his job and to effectively define and identify characteristics of the problem, most importantly the priority and severity of the issue!

The concept of the four-quadrants of priority and severity are taught and the class is clearly shown how business factors influence the quadrant of a bug! The class includes a review of some actual defect arrival graphs from recent commercial product development efforts and provides an answer to the fundamental question of software engineering. Practical aspects of bug tracking, defect logging and bug review meetings are included.

Workshop 3

About the Author

Robert Sabourin has been involved in all aspects of development, testing and management of software engineering projects. Robert graduated from McGill University in 1982. Since writing his first program in 1972, Robert has become an accomplished software engineering management expert. He is presently the President of AmiBug.Com, Inc.; a Montreal-based international firm specializing in software engineering and and software quality assurance training, management consulting and professional development. AmiBug helps companies set up software engineering and quality assurance teams and process through a combination of training and management consulting. Robert was the Director of Research and Development at Purkinje Inc where he was charged with developing world class critical medical software used by clinicians at the point of care. Previously, Robert managed Software Development at Alis Technologies for over ten years. He has built several successful software development teams and champions the implementation of "light effective process" to achieve excellence in delivering on-time, on-quality, on-budget commercial software solutions.

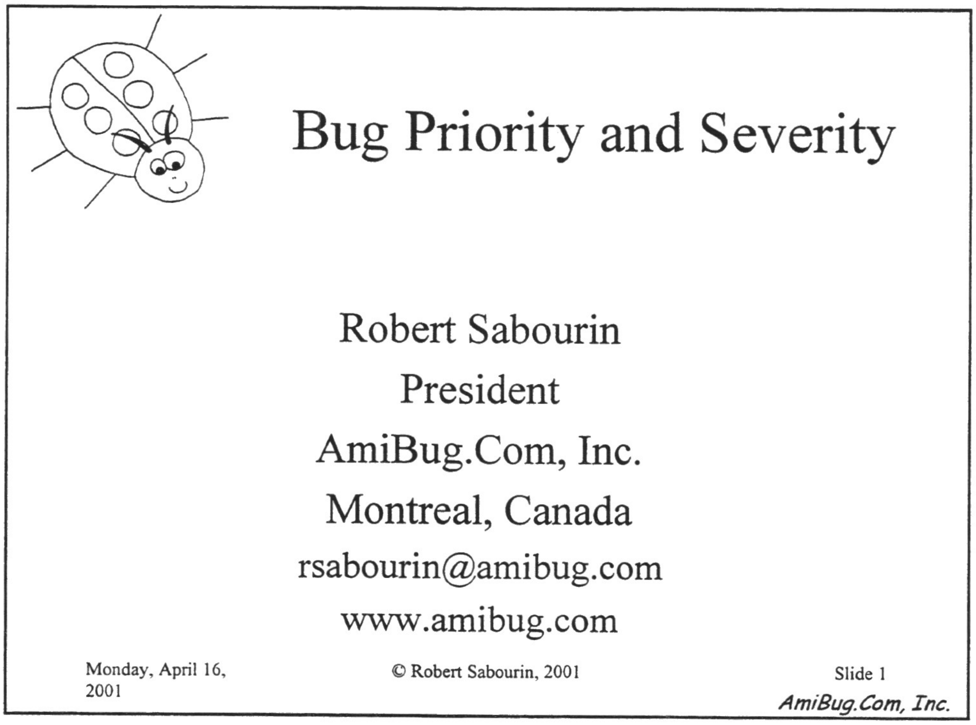
Robert has championed many complex international multilingual software development and globalization efforts involving several intricate business partnerships and relationships including international government (Czech, Egypt, France, Morocco, Algeria...) and commercial entities (Microsoft, IBM, AT&T, HP, Thompson CSF, Olivetti...). Systems included concurrent coordinated multilingual multiplatform product releases.

Robert's pioneering work with Infolytica Corporation led to the development of the first commercially available platform independent graphics standard GKS and several toolkits which allowed for cross platform development and porting of complex CAD, Graphics, Analysis and Non-Destructive Simulation systems.

Robert is a frequent guest lecturer at McGill University where he relates theoretical aspects of Software Engineering to real world examples with practical hands-on demonstrations.

In 1999, Robert completed a short book illustrated by his daughter Catherine entitled"I Am a Bug" (ISBN 0-9685774-0-7).

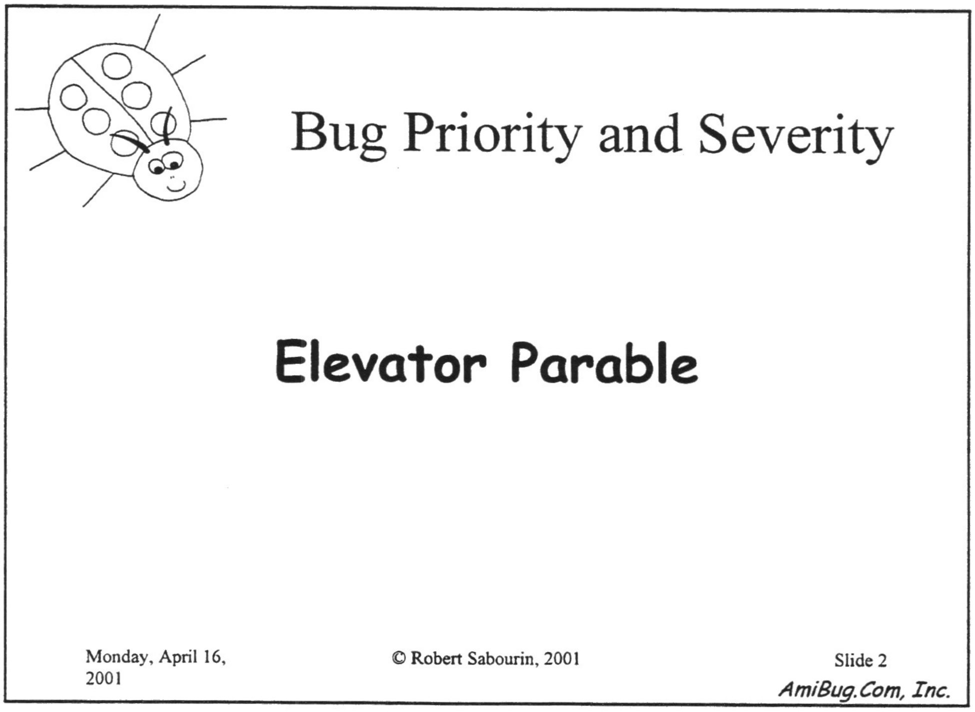
Robert has received professional recognition for many accomplishments over the years. At TEPR 2000 - award for best electronic patient record product to EHS using the Purkinje CNC component. Byte Middle-East's 1992 Product of the Year for the AVT-710 product family achieving a ZERO FIELD REPORTED software defect rate with over 15,000 units installed. (Project involved over 27-man month's effort!); Quebec Order of Engineers' recognition for creating and managing the Alis R&D Policy Guide - Development Framework and process.



Bug Priority and Severity

Robert Sabourin
President
AmiBug.Com, Inc.
Montreal, Canada
rsabourin@amibug.com
www.amibug.com

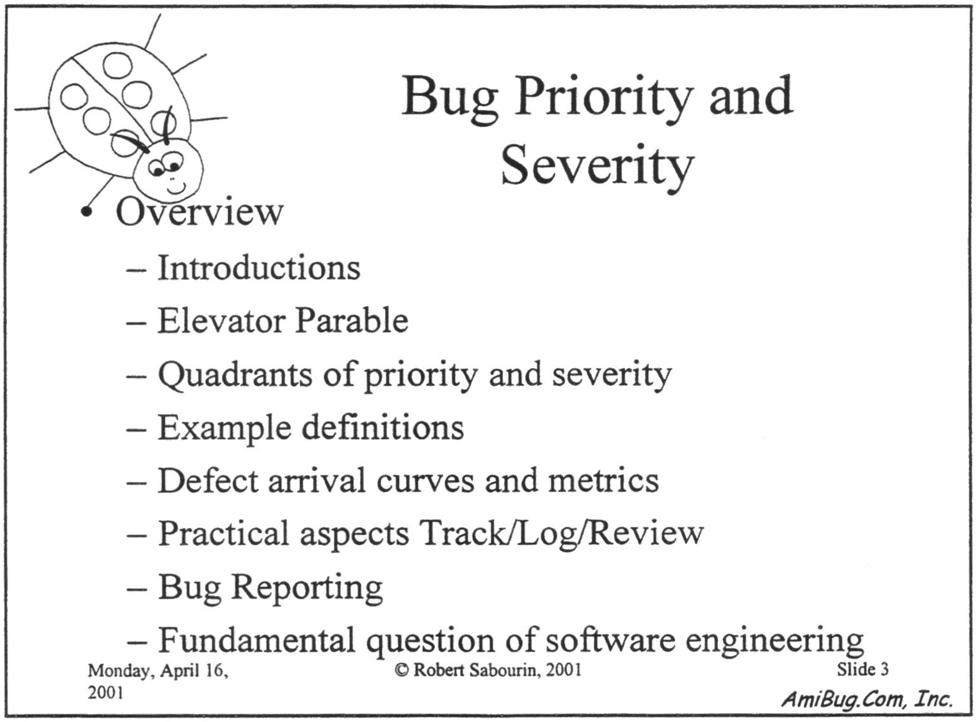
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 1
AmiBug.Com, Inc.



Bug Priority and Severity

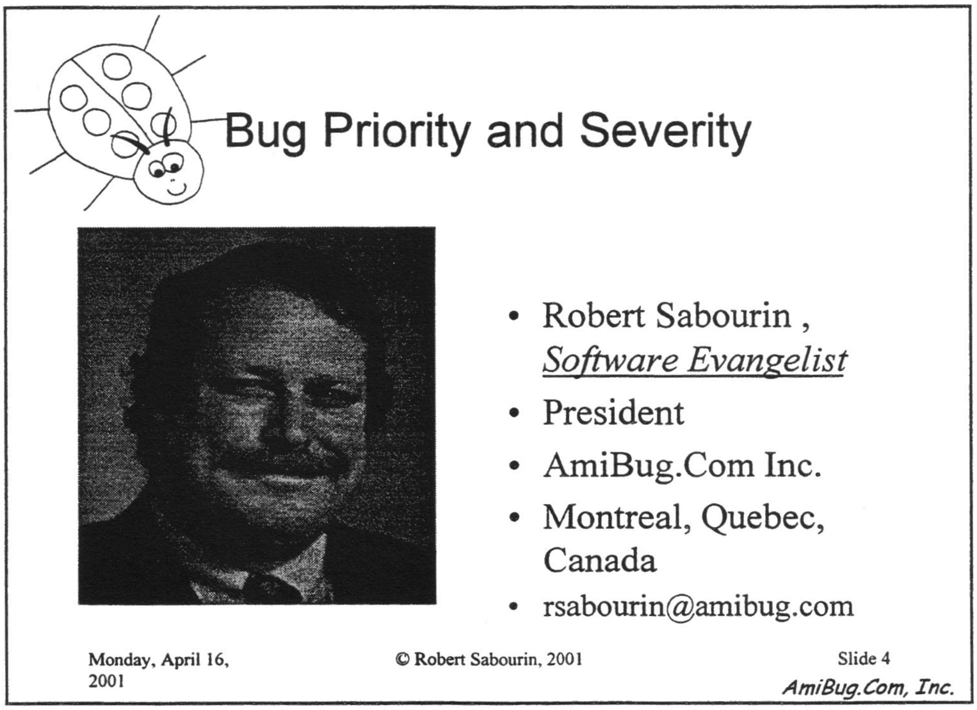
Elevator Parable

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 2
AmiBug.Com, Inc.



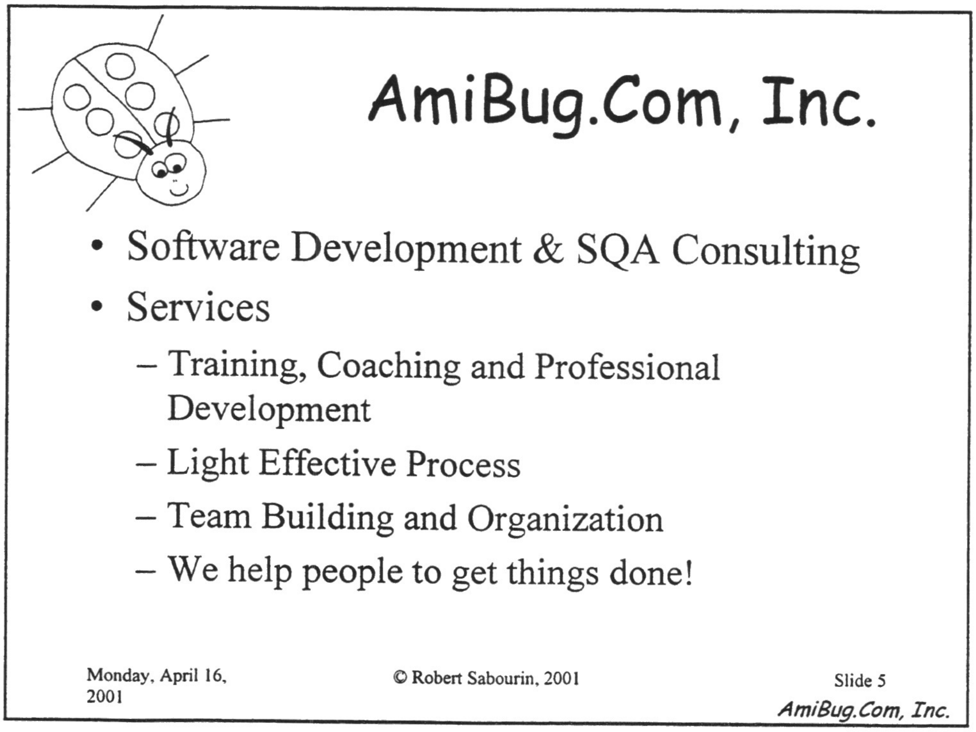
Bug Priority and Severity

- Overview
  - Introductions
  - Elevator Parable
  - Quadrants of priority and severity
  - Example definitions
  - Defect arrival curves and metrics
  - Practical aspects Track/Log/Review
  - Bug Reporting
  - Fundamental question of software engineering



Bug Priority and Severity

- Robert Sabourin , *Software Evangelist*
- President
- AmiBug.Com Inc.
- Montreal, Quebec, Canada
- rsabourin@amibug.com

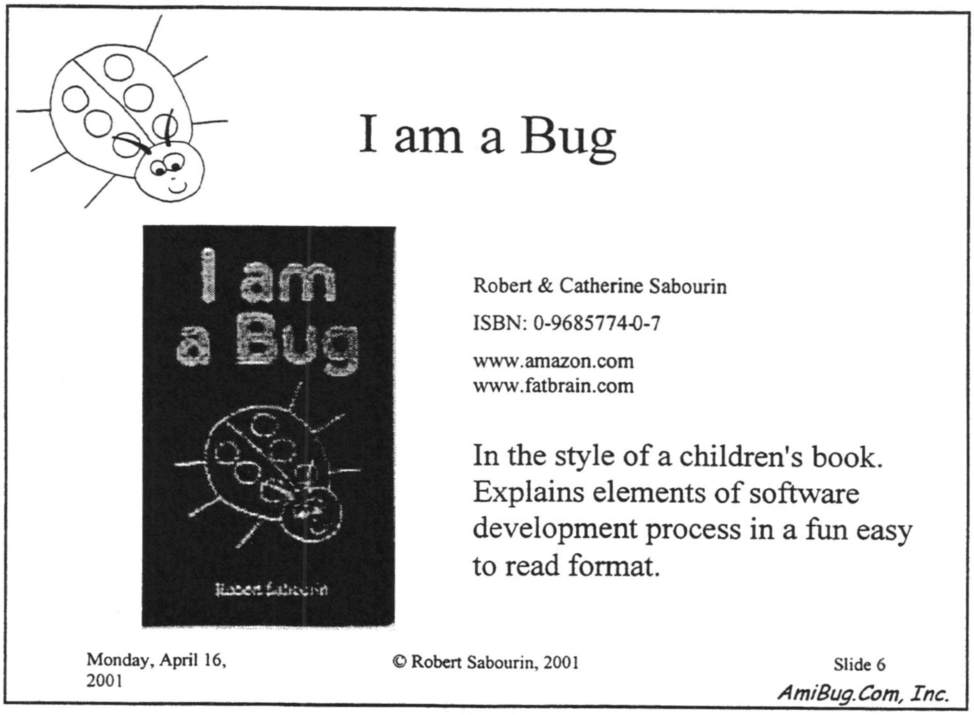


AmiBug.Com, Inc.

- Software Development & SQA Consulting
- Services
  - Training, Coaching and Professional Development
  - Light Effective Process
  - Team Building and Organization
  - We help people to get things done!

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 5

AmiBug.Com, Inc.



I am a Bug

I am
a Bug
Robert Sabourin

Robert & Catherine Sabourin

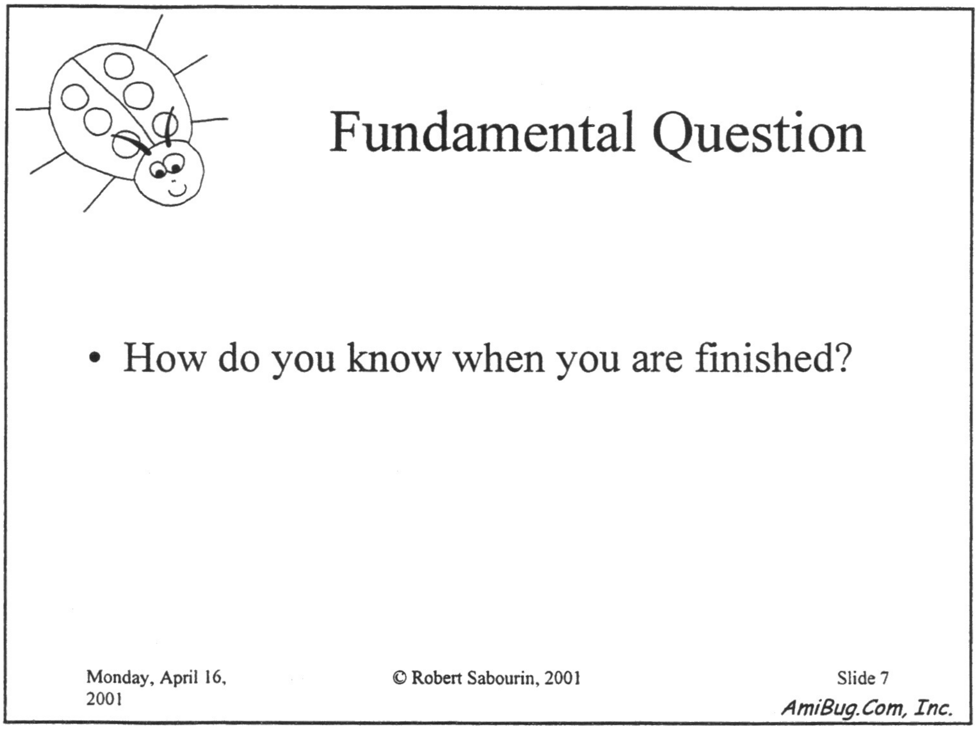
ISBN: 0-9685774-0-7

www.amazon.com
www.fatbrain.com

In the style of a children's book. Explains elements of software development process in a fun easy to read format.

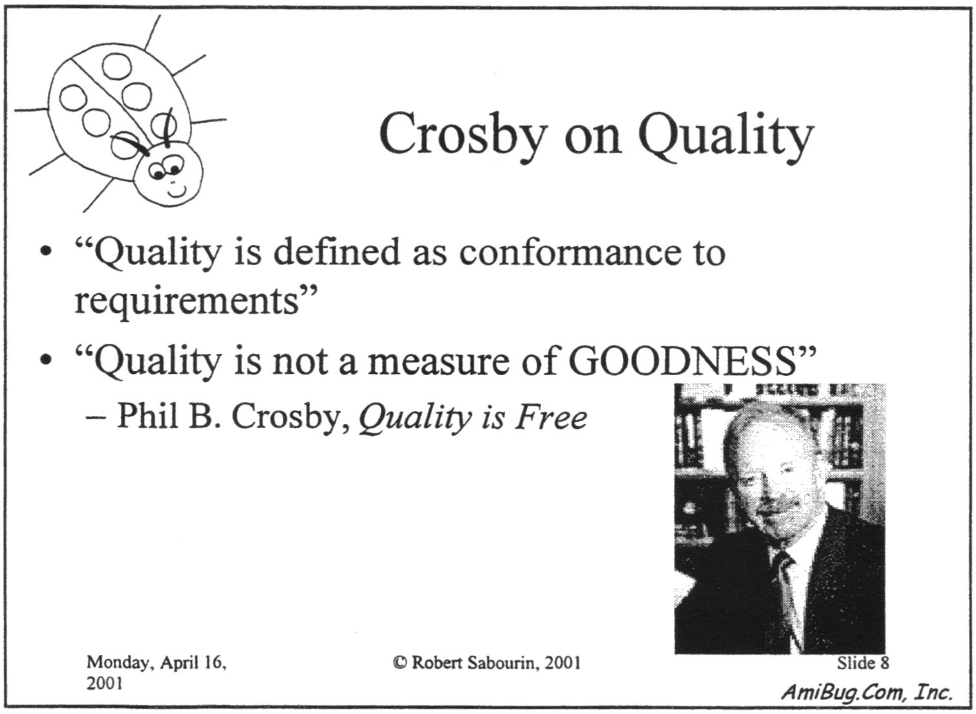
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 6

AmiBug.Com, Inc.



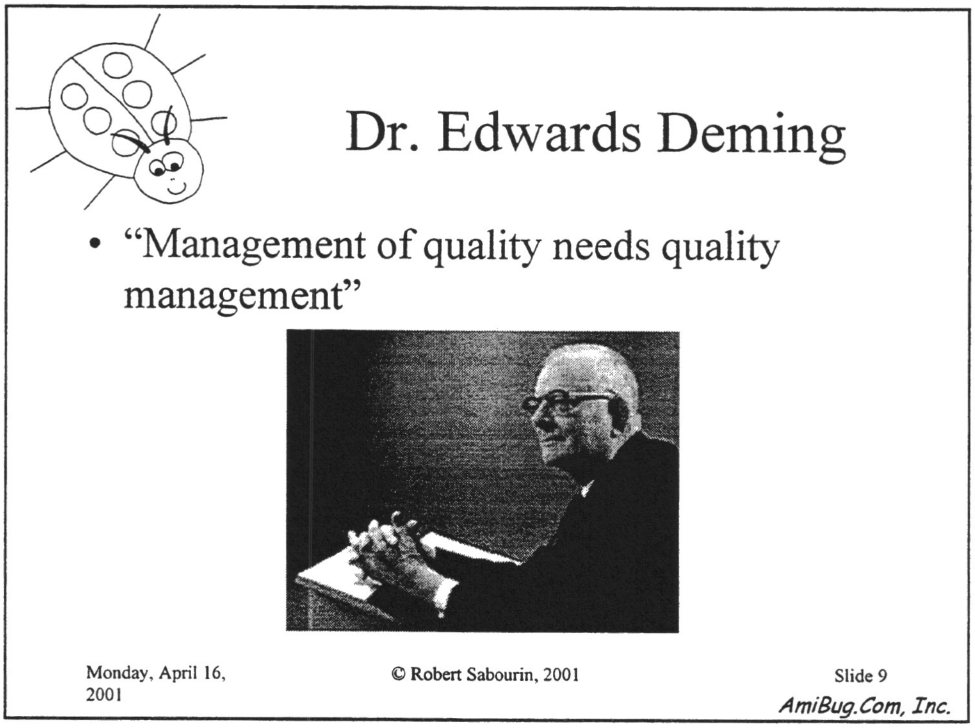
Fundamental Question

- How do you know when you are finished?



Crosby on Quality

- “Quality is defined as conformance to requirements”
- “Quality is not a measure of GOODNESS”
  - Phil B. Crosby, *Quality is Free*



Dr. Edwards Deming

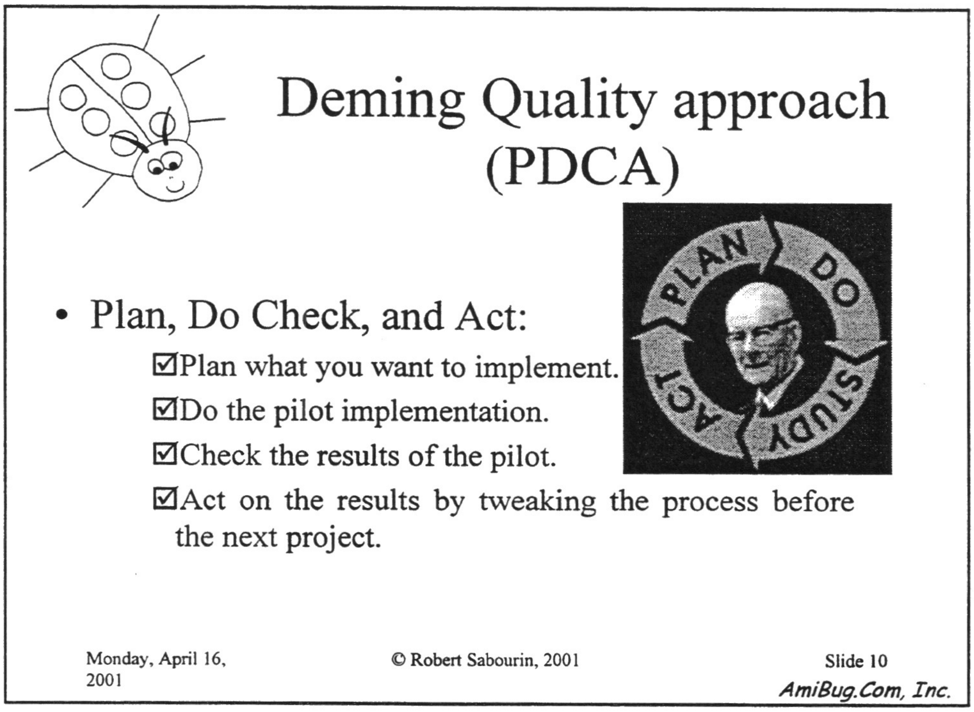
- "Management of quality needs quality management"

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 9

AmiBug.Com, Inc.



Deming Quality approach (PDCA)

PLAN
DO
STUDY
ACT

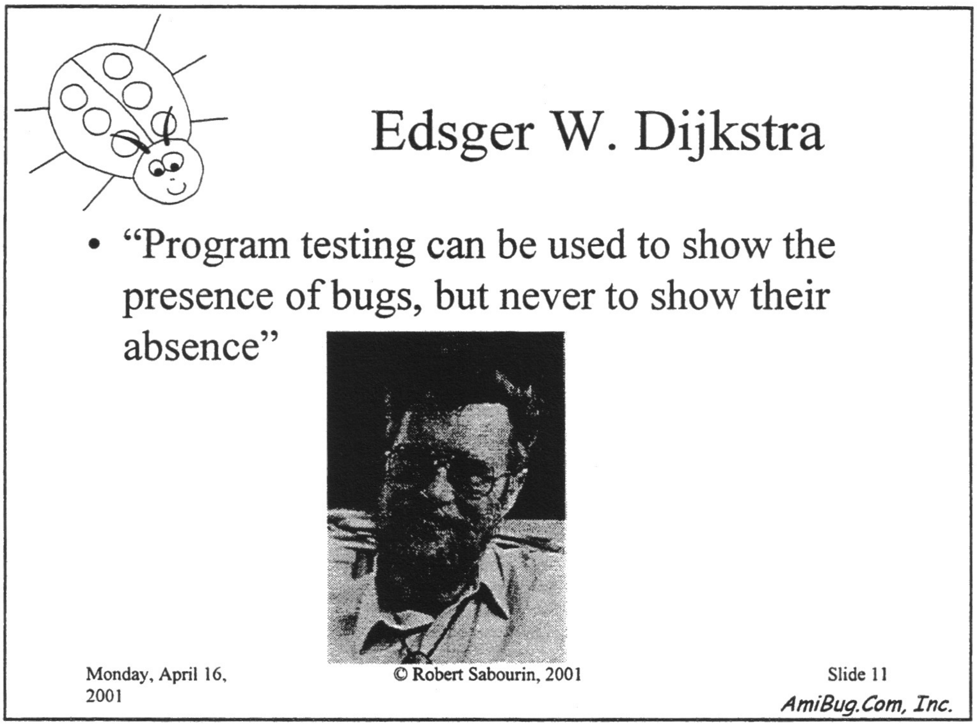
- Plan, Do Check, and Act:
  - ☑Plan what you want to implement.
  - ☑Do the pilot implementation.
  - ☑Check the results of the pilot.
  - ☑Act on the results by tweaking the process before the next project.

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 10

AmiBug.Com, Inc.

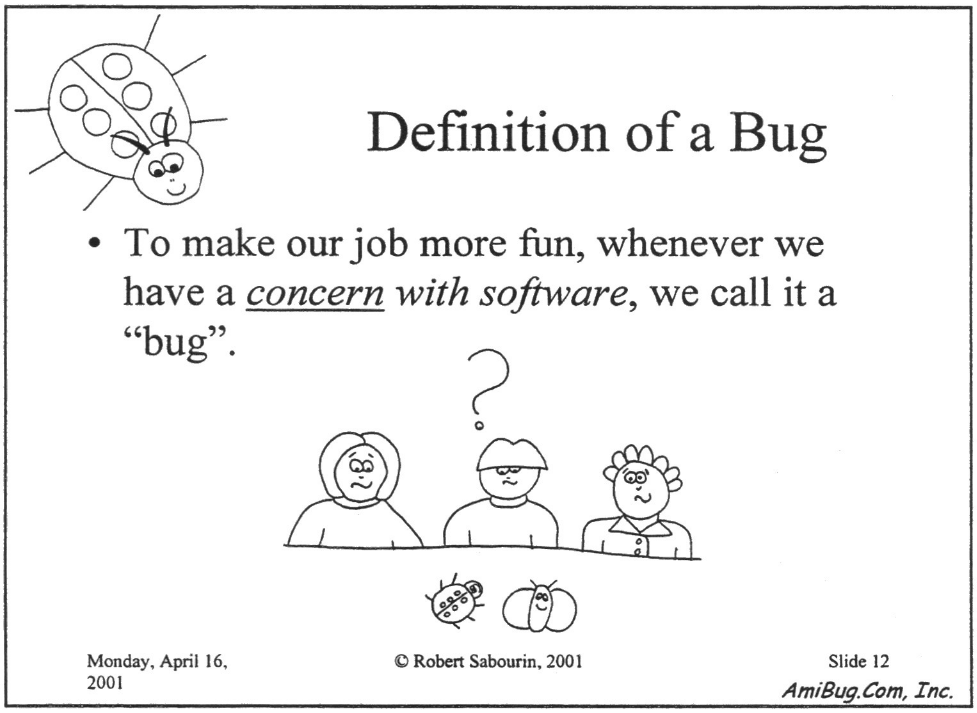


Edsger W. Dijkstra

- "Program testing can be used to show the presence of bugs, but never to show their absence"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 11

AmiBug.Com, Inc.

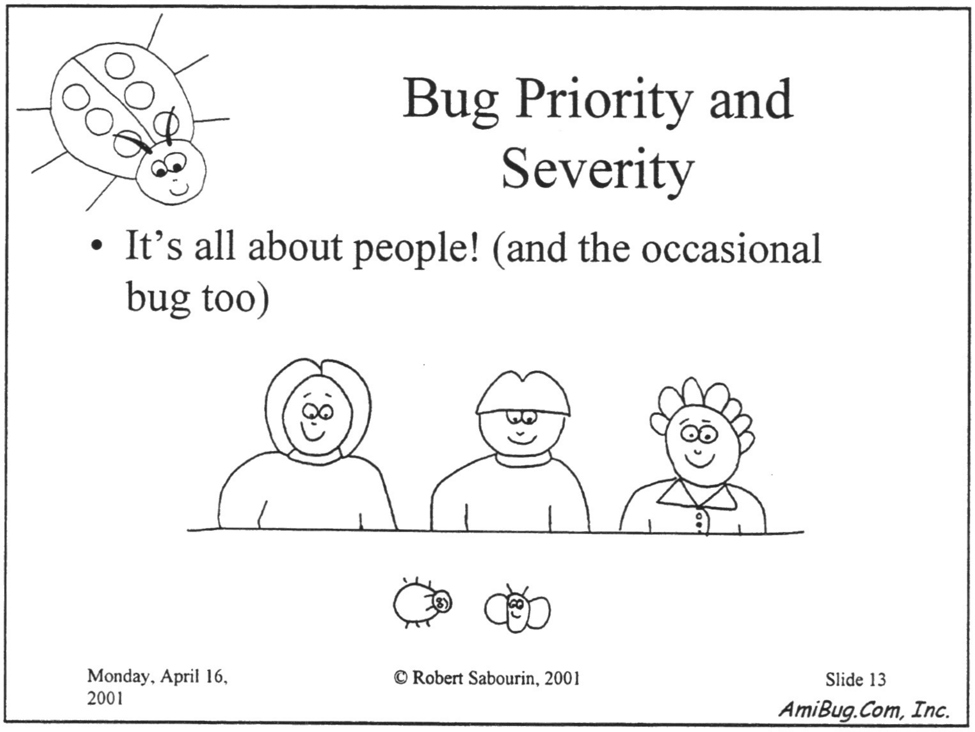


Definition of a Bug

- To make our job more fun, whenever we have a *concern with software*, we call it a "bug".

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 12

AmiBug.Com, Inc.



Bug Priority and Severity

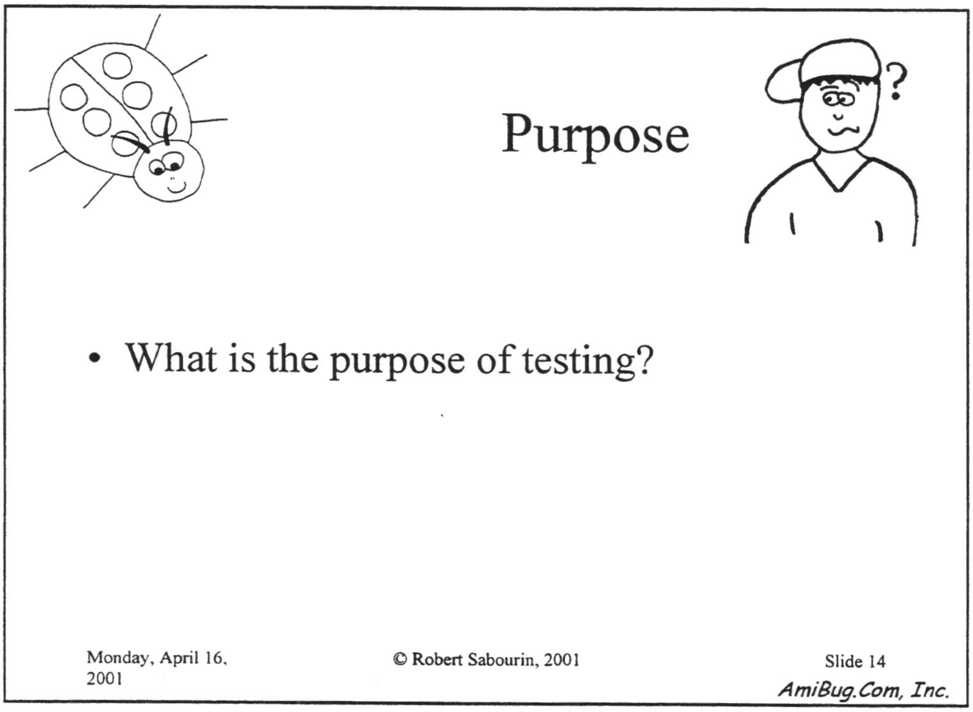
- It's all about people! (and the occasional bug too)

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 13

AmiBug.Com, Inc.



Purpose

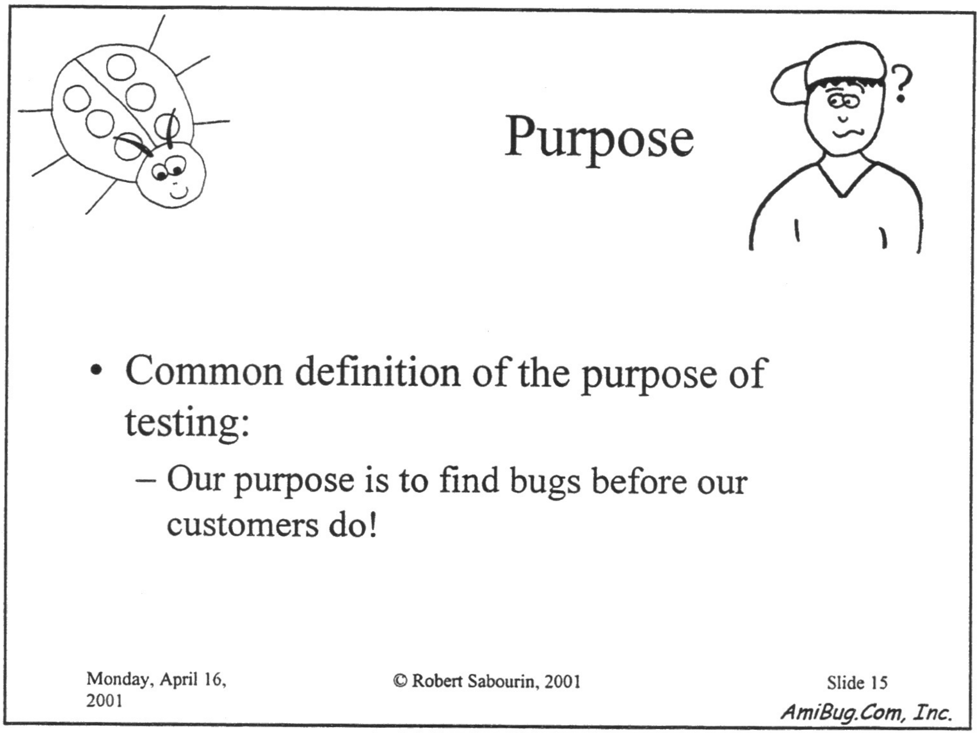
- What is the purpose of testing?

Monday, April 16, 2001

© Robert Sabourin, 2001

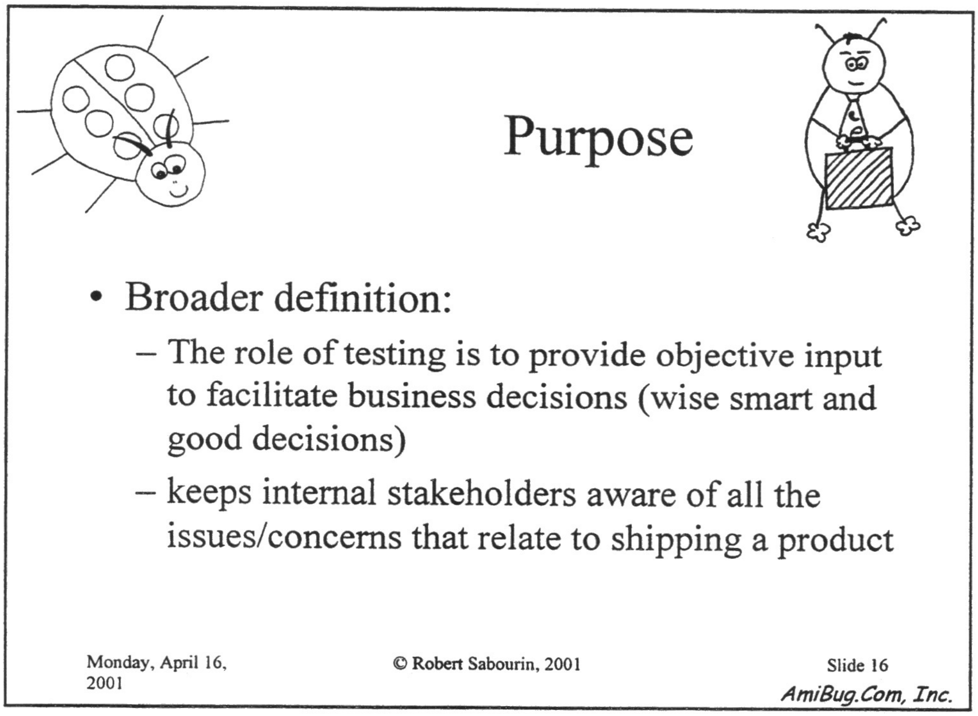
Slide 14

AmiBug.Com, Inc.



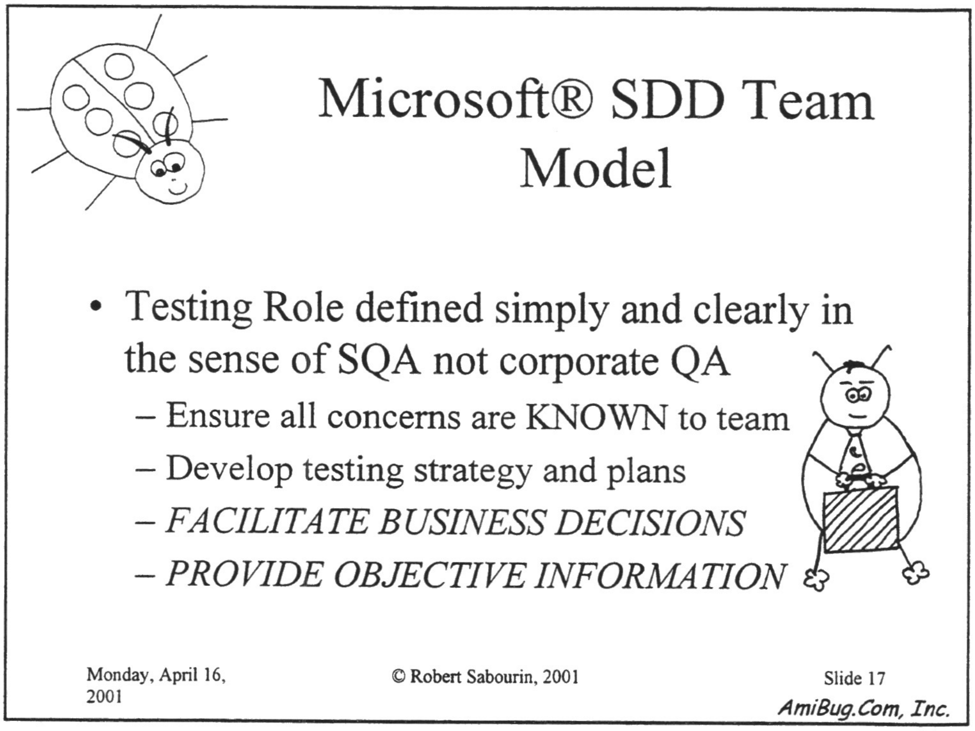
Purpose

- Common definition of the purpose of testing:
  - Our purpose is to find bugs before our customers do!



Purpose

- Broader definition:
  - The role of testing is to provide objective input to facilitate business decisions (wise smart and good decisions)
  - keeps internal stakeholders aware of all the issues/concerns that relate to shipping a product

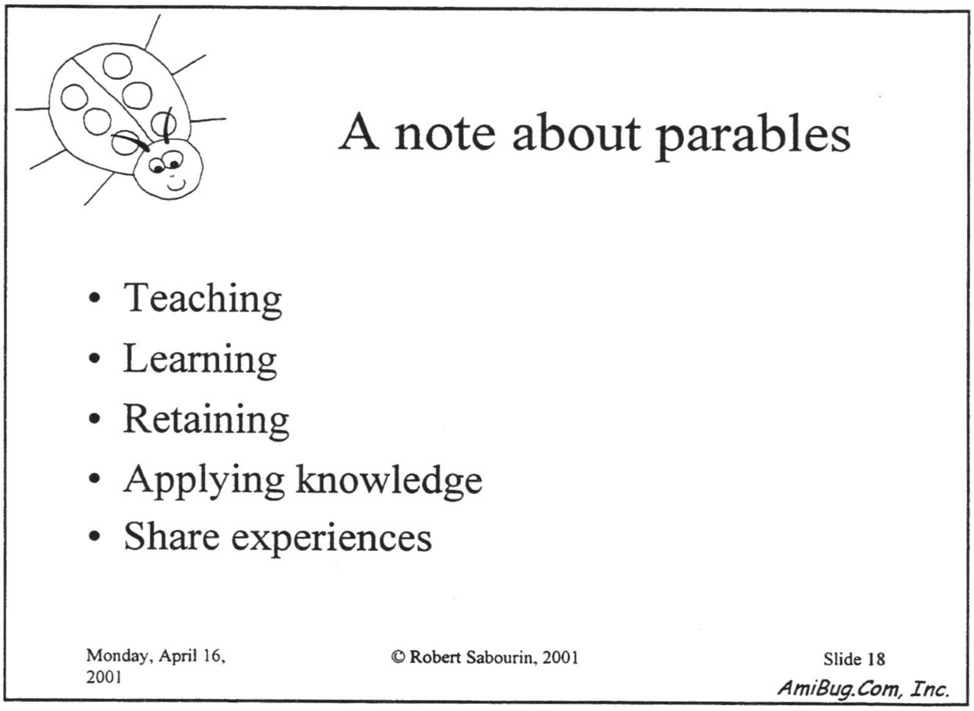


Microsoft® SDD Team Model

- Testing Role defined simply and clearly in the sense of SQA not corporate QA
  - Ensure all concerns are KNOWN to team
  - Develop testing strategy and plans
  - *FACILITATE BUSINESS DECISIONS*
  - *PROVIDE OBJECTIVE INFORMATION*

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 17

AmiBug.Com, Inc.

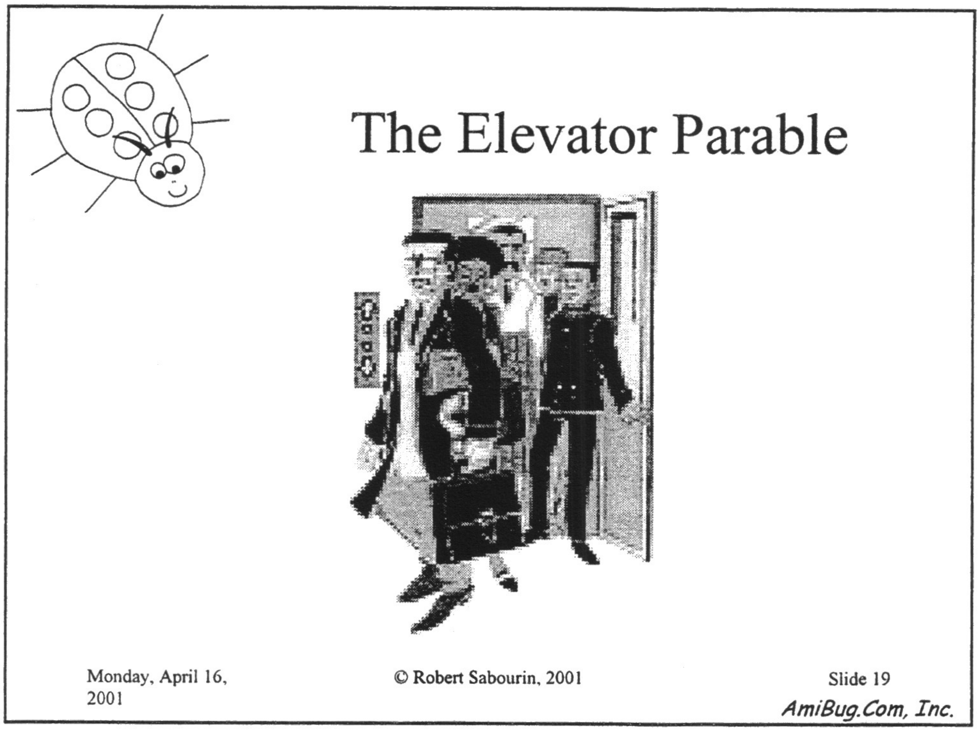


A note about parables

- Teaching
- Learning
- Retaining
- Applying knowledge
- Share experiences

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 18

AmiBug.Com, Inc.

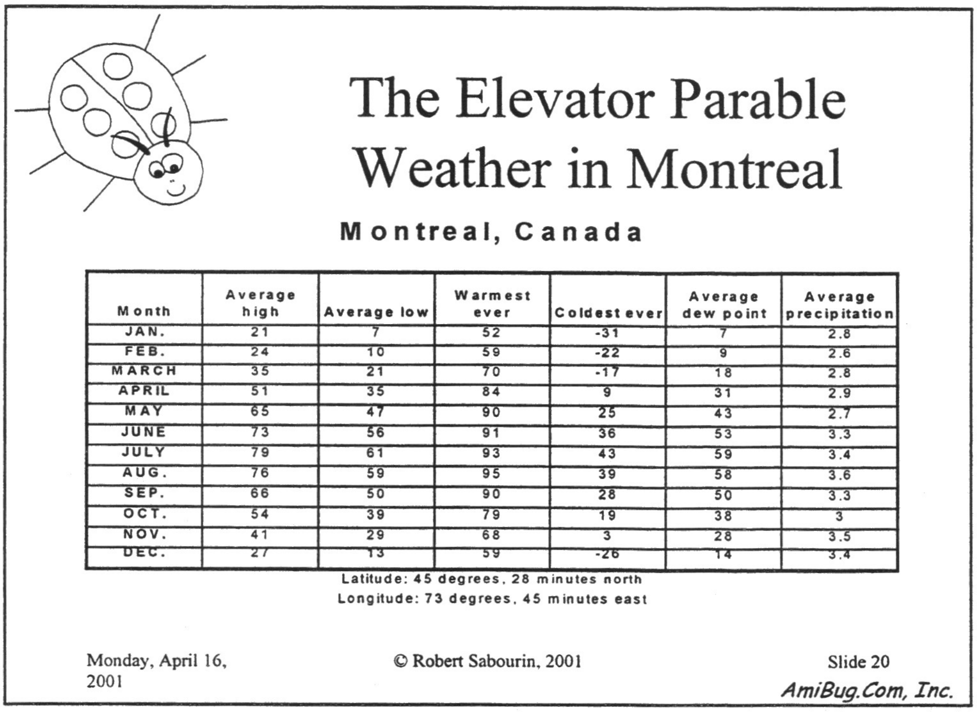


The Elevator Parable

Monday, April 16, 2001

Slide 19

AmiBug.Com, Inc.



The Elevator Parable Weather in Montreal

Montreal, Canada

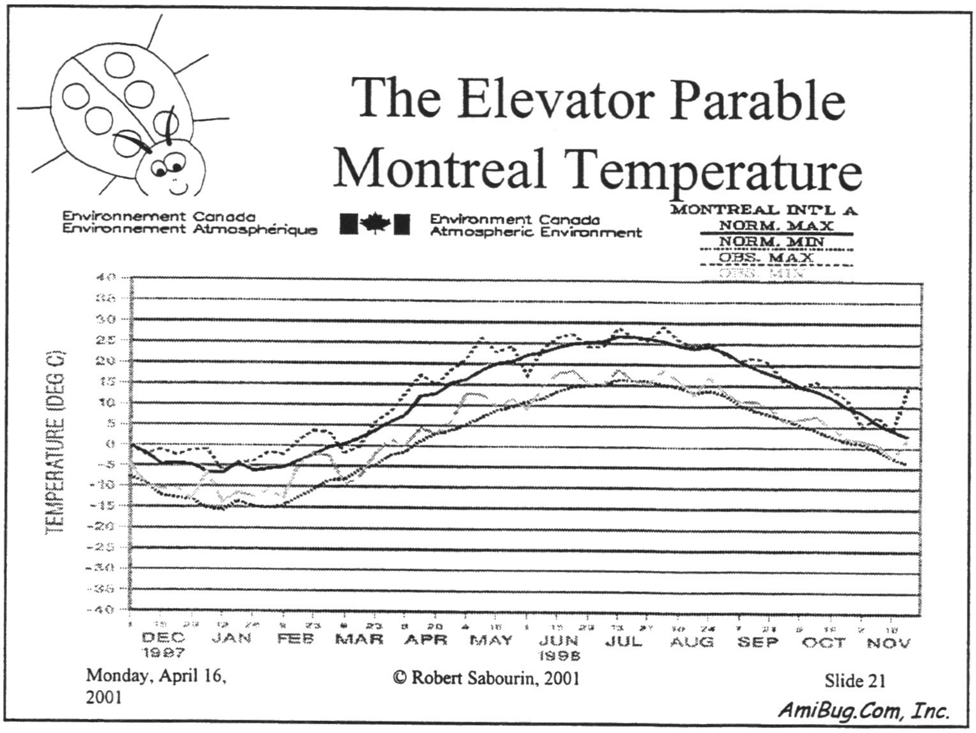
Month	Average high	Average low	Warmest ever	Coldest ever	Average dew point	Average precipitation
JAN.	21	7	52	-31	7	2.8
FEB.	24	10	59	-22	9	2.6
MARCH	35	21	70	-17	18	2.8
APRIL	51	35	84	9	31	2.9
MAY	65	47	90	25	43	2.7
JUNE	73	56	91	36	53	3.3
JULY	79	61	93	43	59	3.4
AUG.	76	59	95	39	58	3.6
SEP.	66	50	90	28	50	3.3
OCT.	54	39	79	19	38	3
NOV.	41	29	68	3	28	3.5
DEC.	27	13	59	-26	14	3.4

Latitude: 45 degrees, 28 minutes north
Longitude: 73 degrees, 45 minutes east

Monday, April 16, 2001

Slide 20

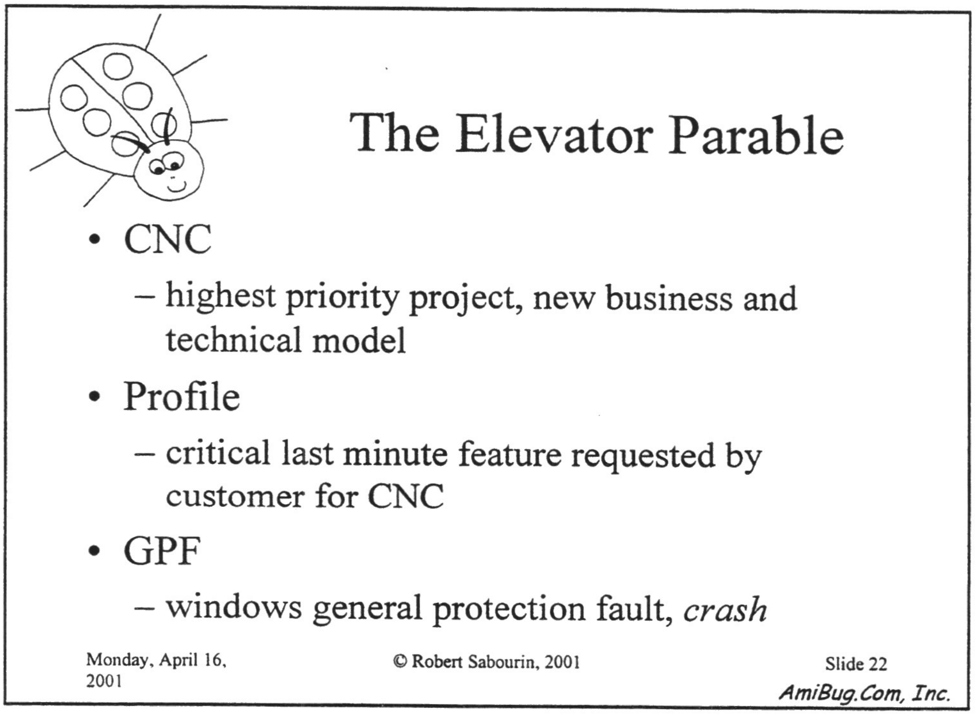
AmiBug.Com, Inc.



The Elevator Parable
Montreal Temperature

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 21

AmiBug.Com, Inc.

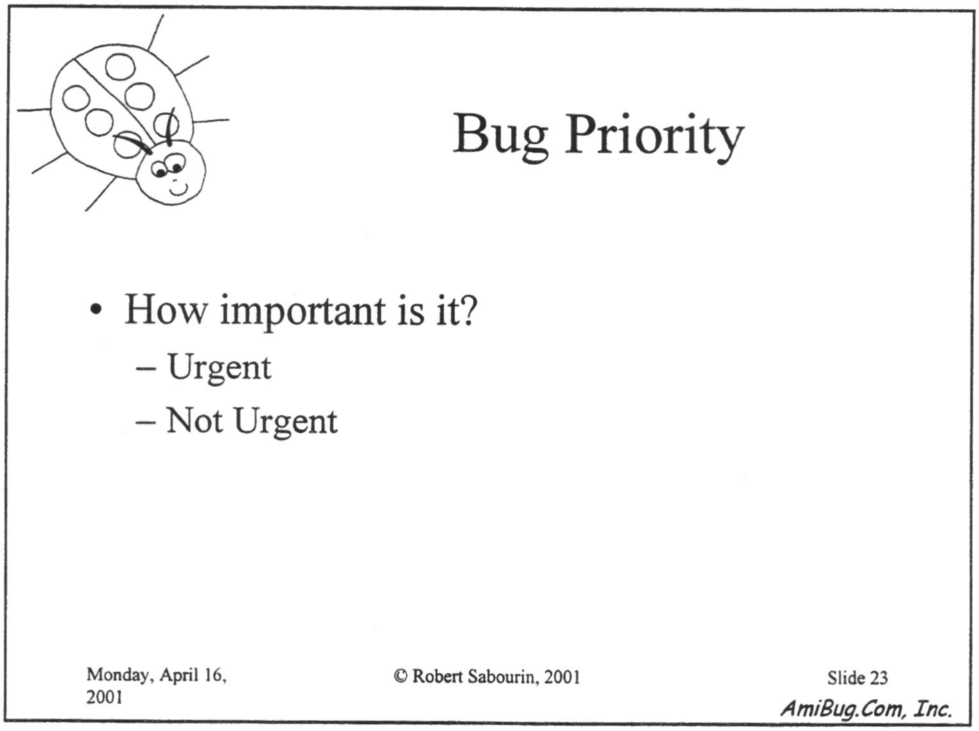


The Elevator Parable

- CNC
  - highest priority project, new business and technical model
- Profile
  - critical last minute feature requested by customer for CNC
- GPF
  - windows general protection fault, *crash*

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 22

AmiBug.Com, Inc.

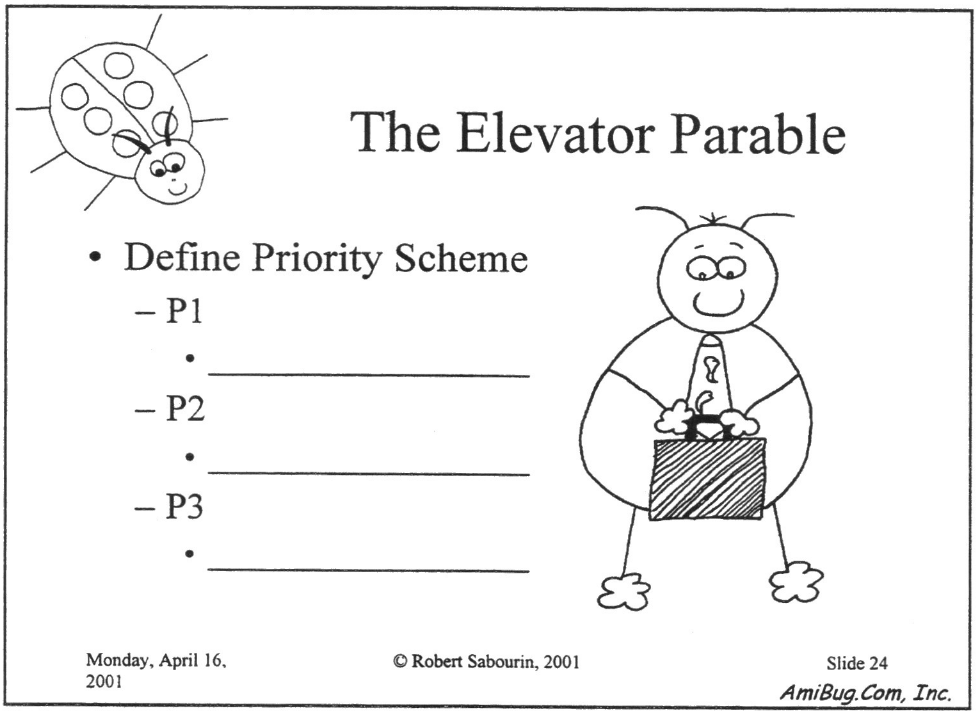


Bug Priority

- How important is it?
  - Urgent
  - Not Urgent

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 23

AmiBug.Com, Inc.

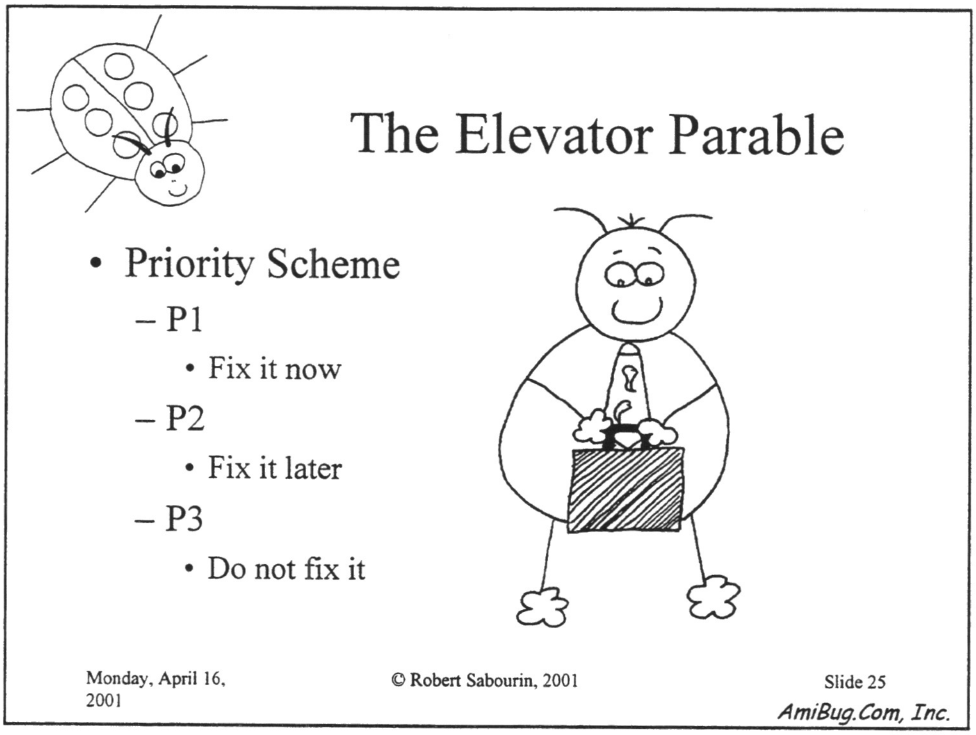


The Elevator Parable

- Define Priority Scheme
  - P1
    - ______________________
  - P2
    - ______________________
  - P3
    - ______________________

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 24

AmiBug.Com, Inc.

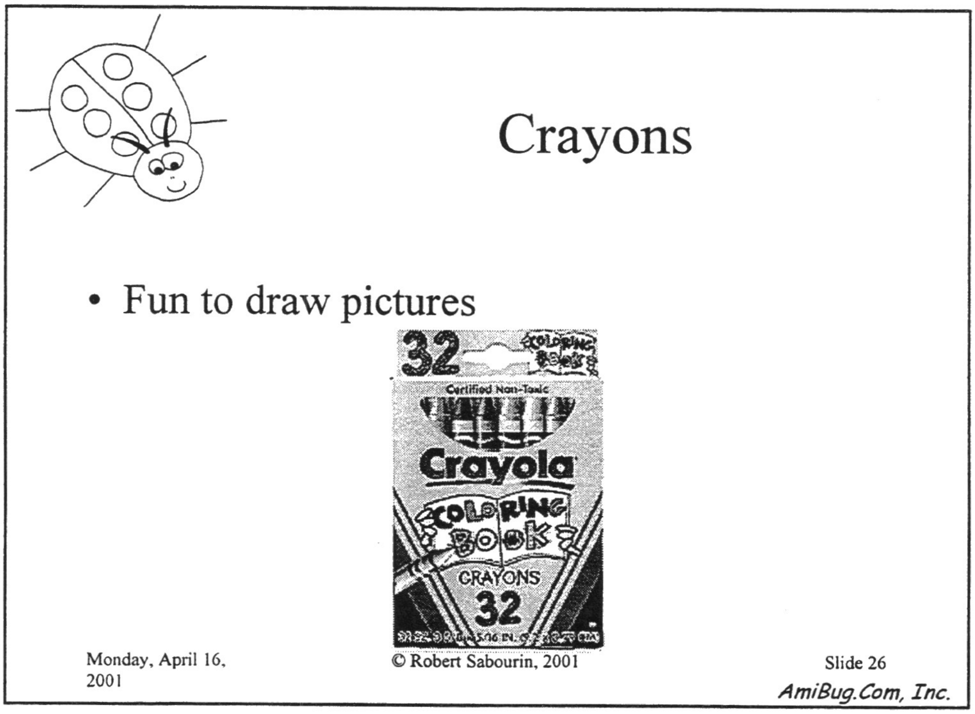


The Elevator Parable

- Priority Scheme
  - P1
    - Fix it now
  - P2
    - Fix it later
  - P3
    - Do not fix it

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 25

AmiBug.Com, Inc.

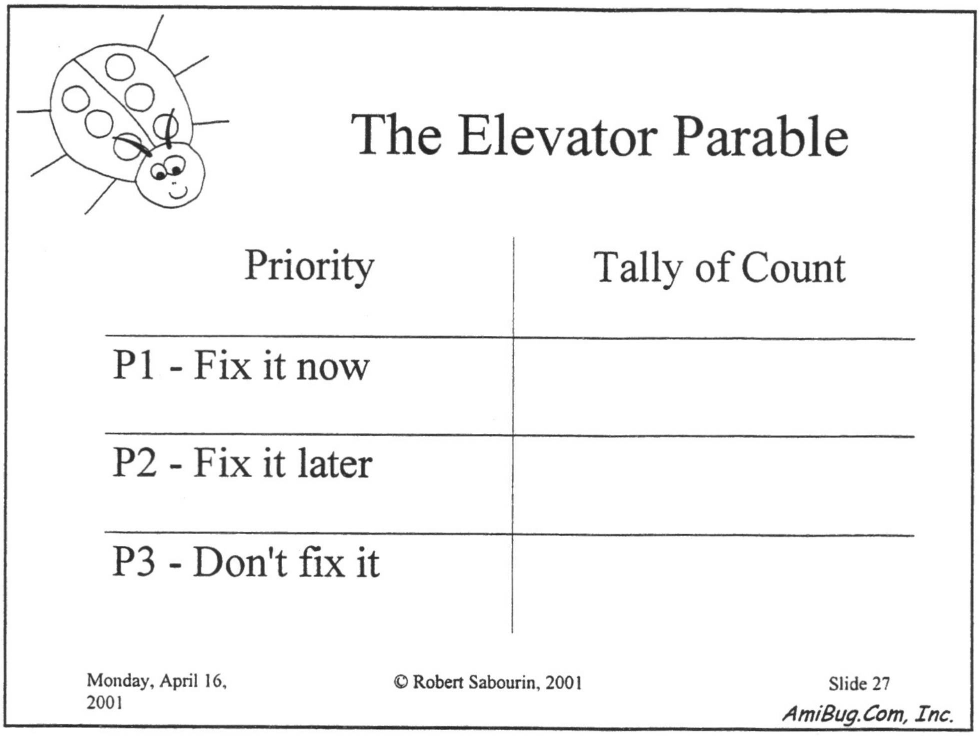


Crayons

- Fun to draw pictures

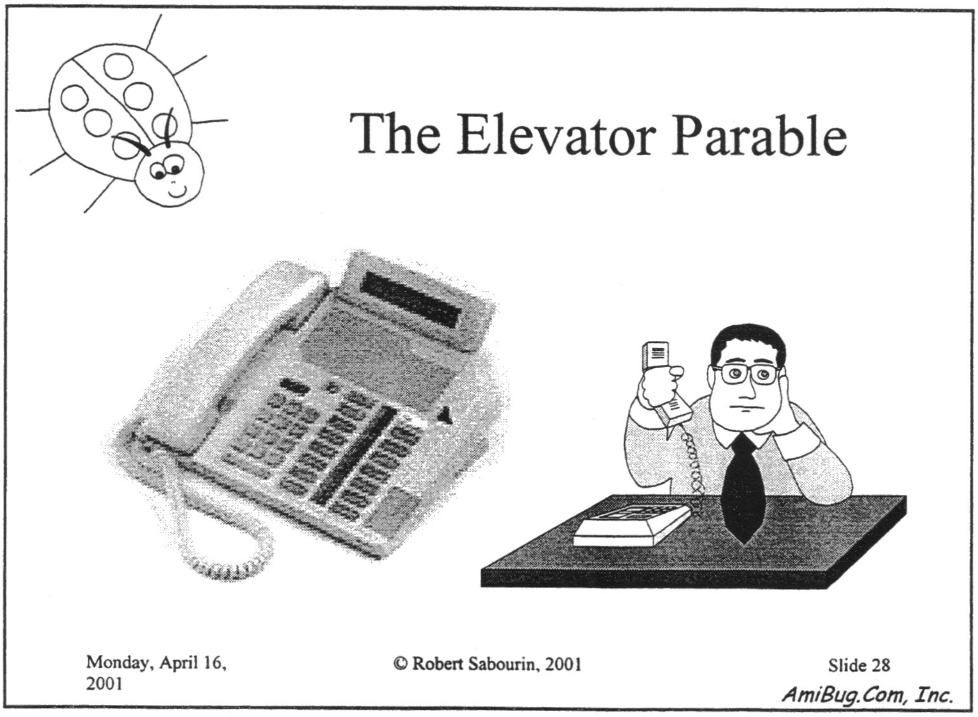
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 26

AmiBug.Com, Inc.

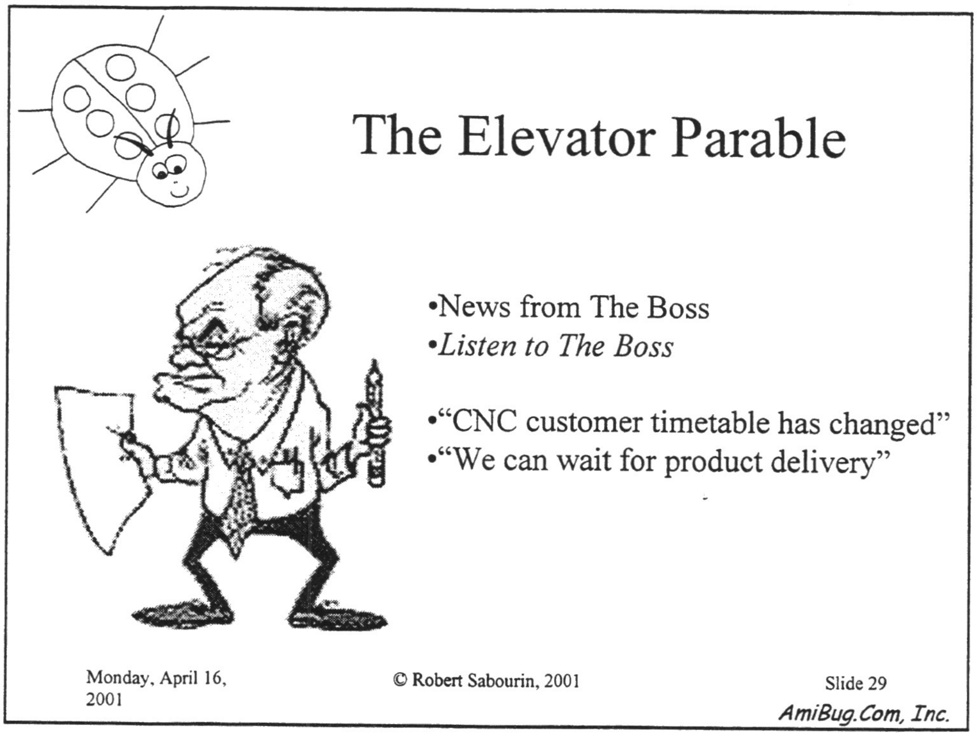


The Elevator Parable

Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	



The Elevator Parable



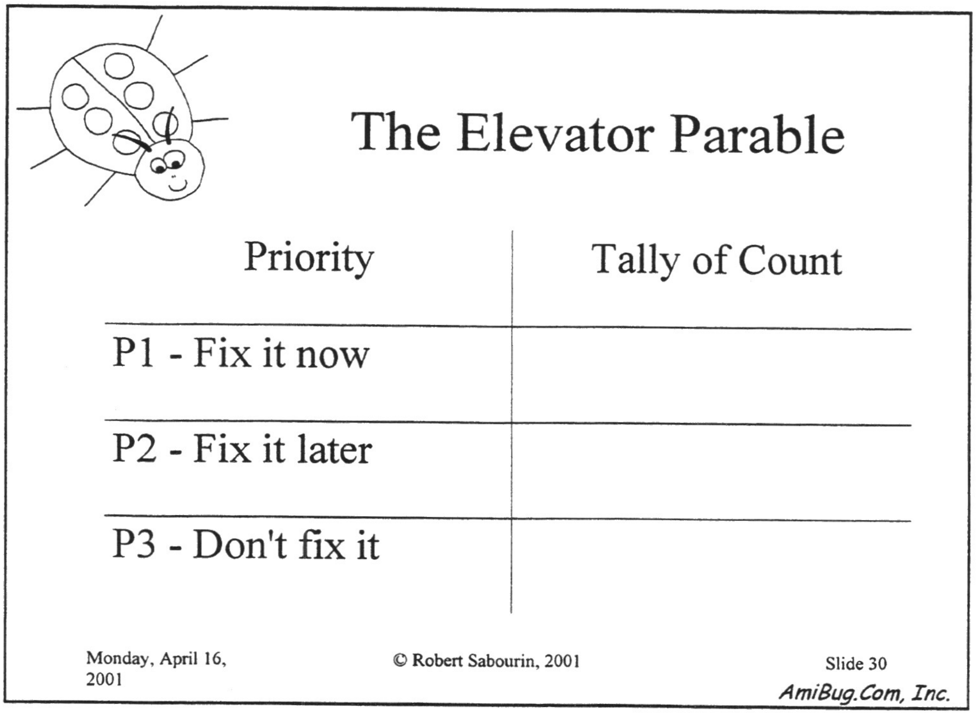
The Elevator Parable

- News from The Boss
- *Listen to The Boss*

- "CNC customer timetable has changed"
- "We can wait for product delivery"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 29

AmiBug.Com, Inc.

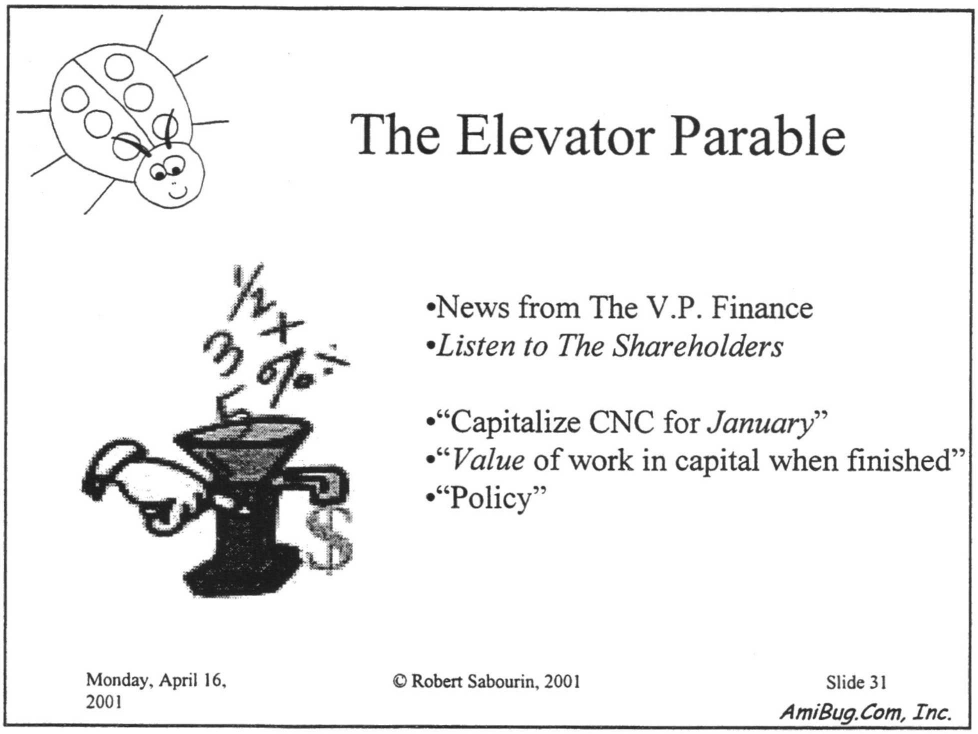


The Elevator Parable

Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 30

AmiBug.Com, Inc.



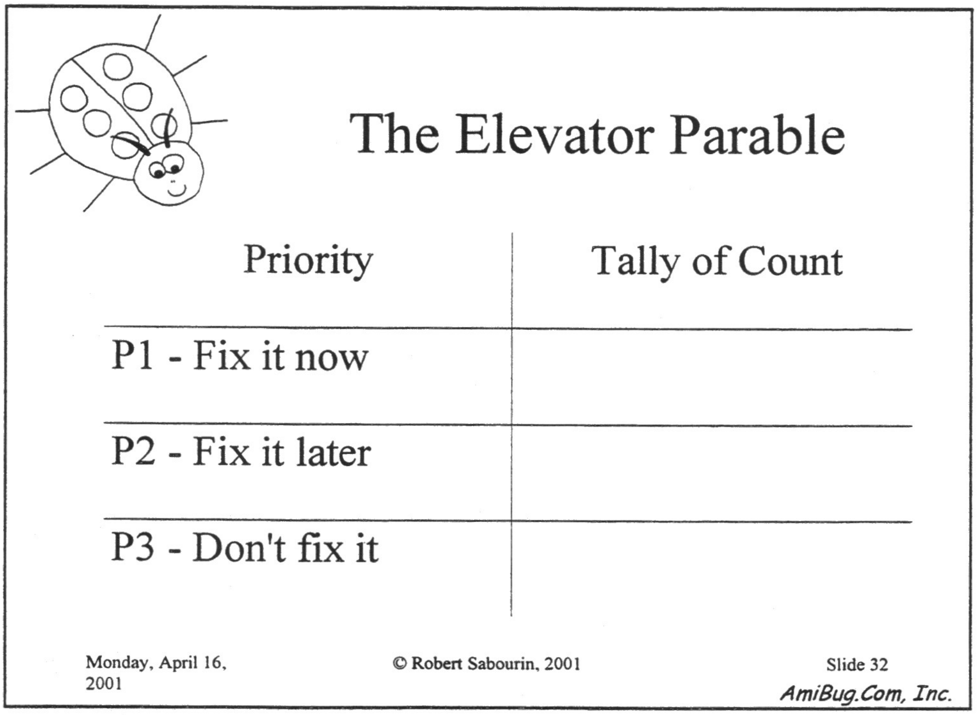
The Elevator Parable

- •News from The V.P. Finance
- •*Listen to The Shareholders*

- •"Capitalize CNC for *January*"
- •"*Value* of work in capital when finished"
- •"Policy"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 31

AmiBug.Com, Inc.

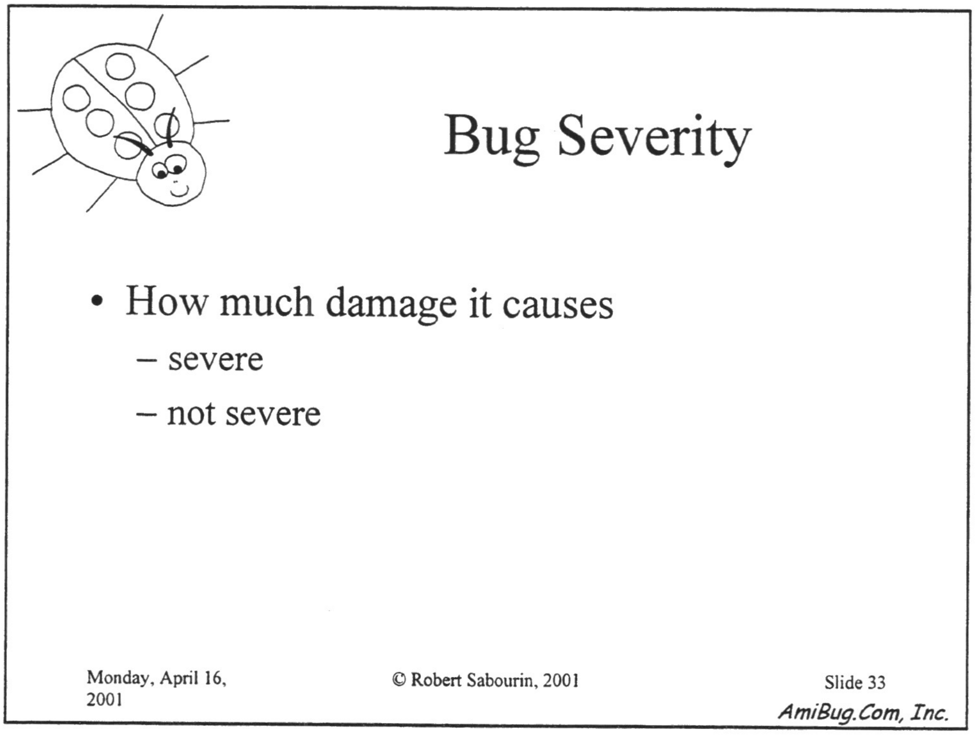


The Elevator Parable

Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 32

AmiBug.Com, Inc.

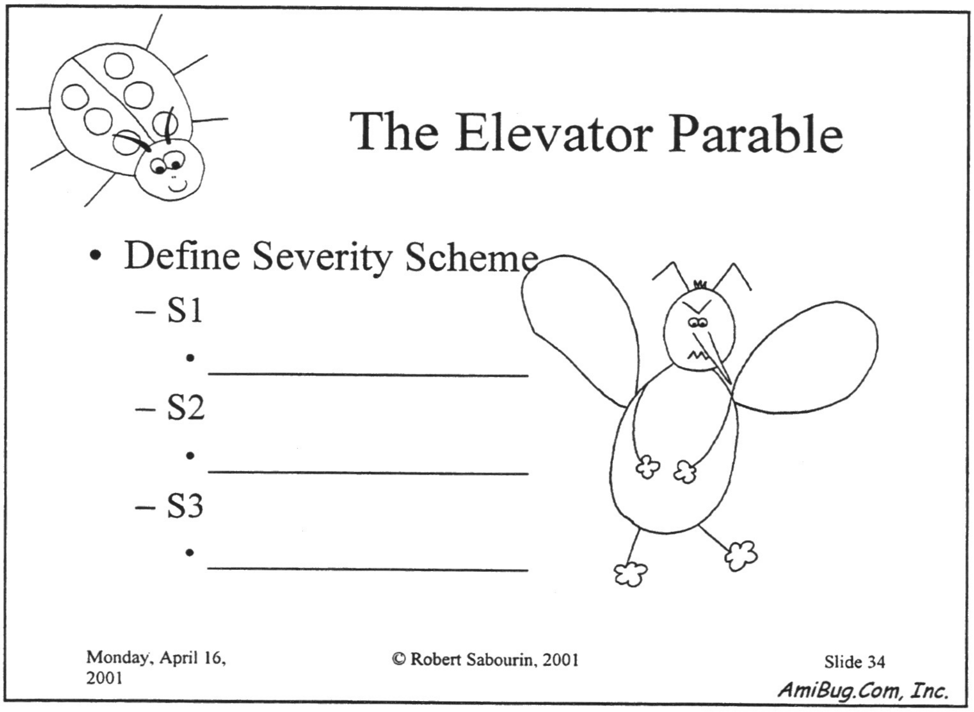


Bug Severity

- How much damage it causes
  - severe
  - not severe

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 33

AmiBug.Com, Inc.

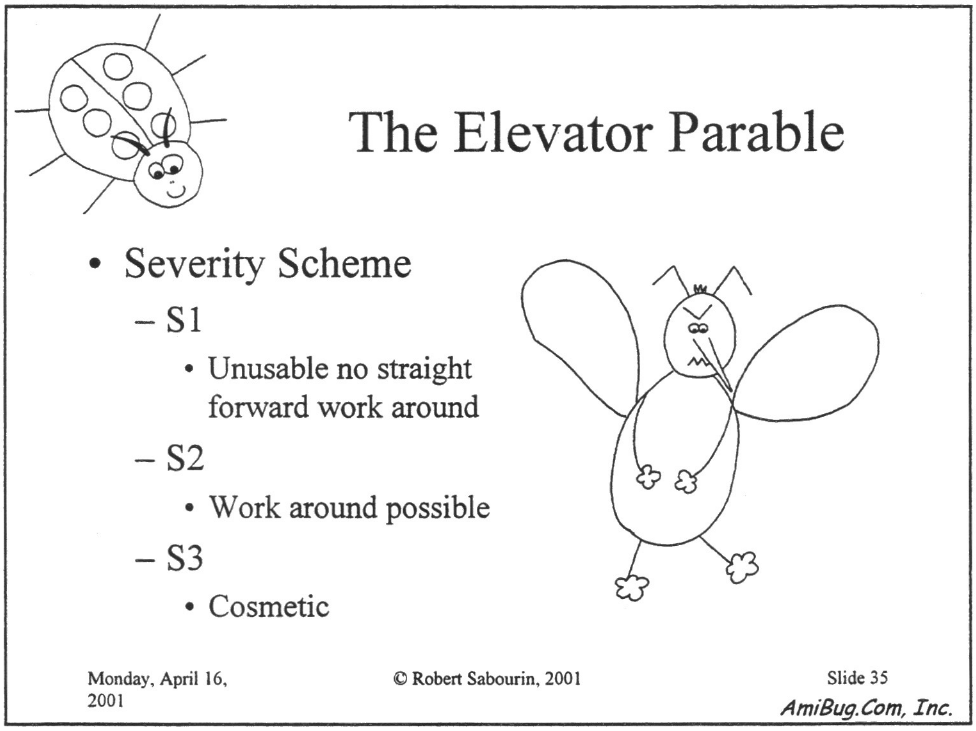


The Elevator Parable

- Define Severity Scheme
  - S1
    - ____________________
  - S2
    - ____________________
  - S3
    - ____________________

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 34

AmiBug.Com, Inc.

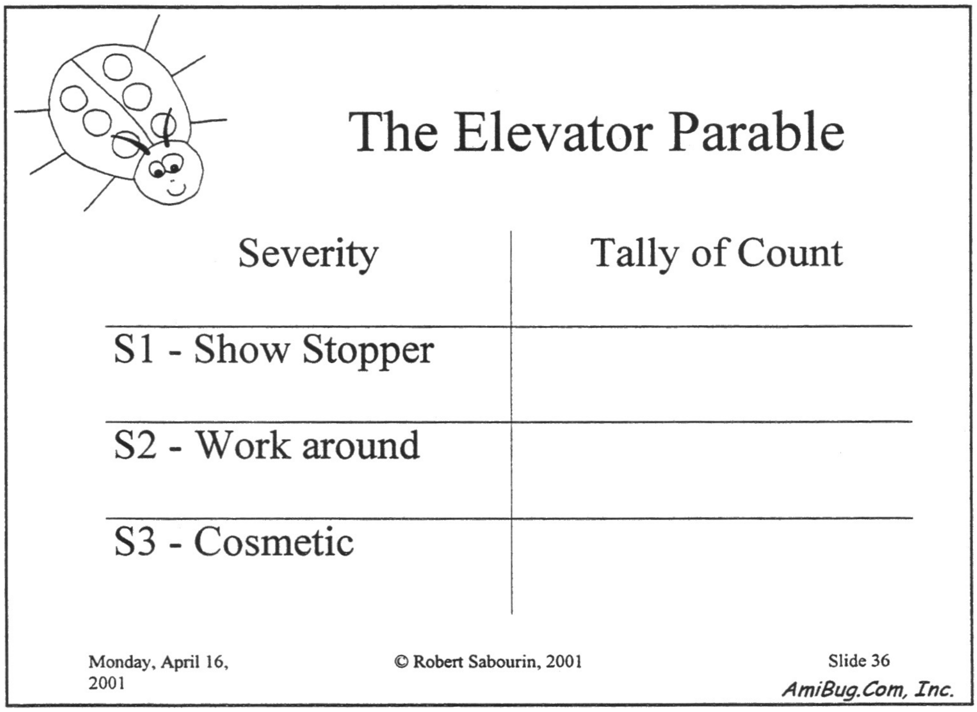


The Elevator Parable

- Severity Scheme
  - S1
    - Unusable no straight forward work around
  - S2
    - Work around possible
  - S3
    - Cosmetic

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 35

AmiBug.Com, Inc.

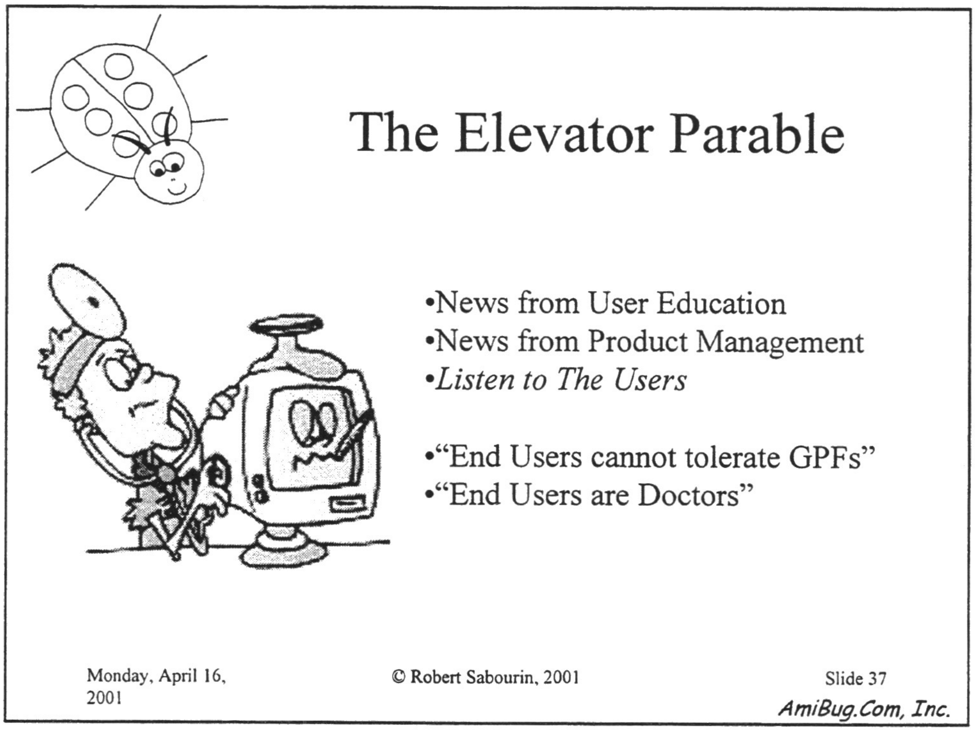


The Elevator Parable

Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 36

AmiBug.Com, Inc.



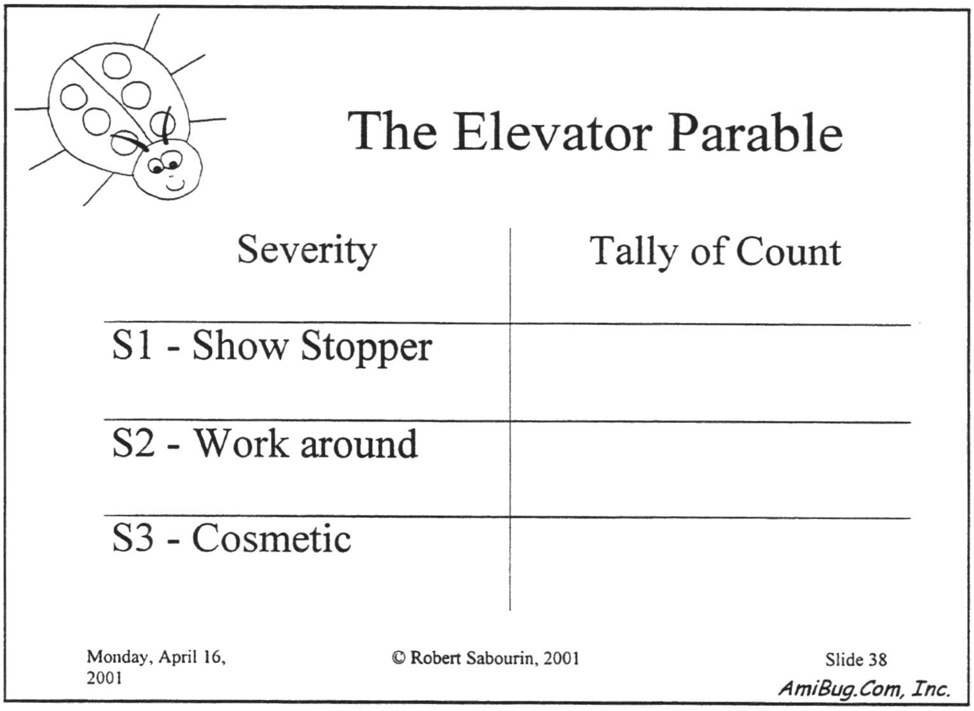
The Elevator Parable

- •News from User Education
- •News from Product Management
- •*Listen to The Users*

- •"End Users cannot tolerate GPFs"
- •"End Users are Doctors"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 37

AmiBug.Com, Inc.

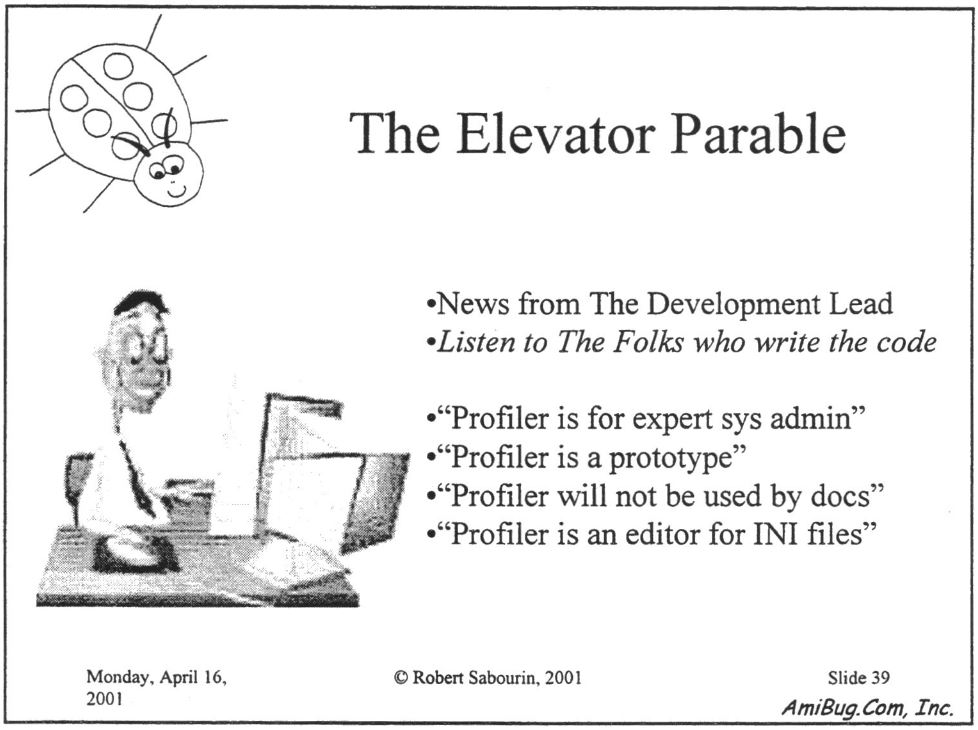


The Elevator Parable

Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 38

AmiBug.Com, Inc.



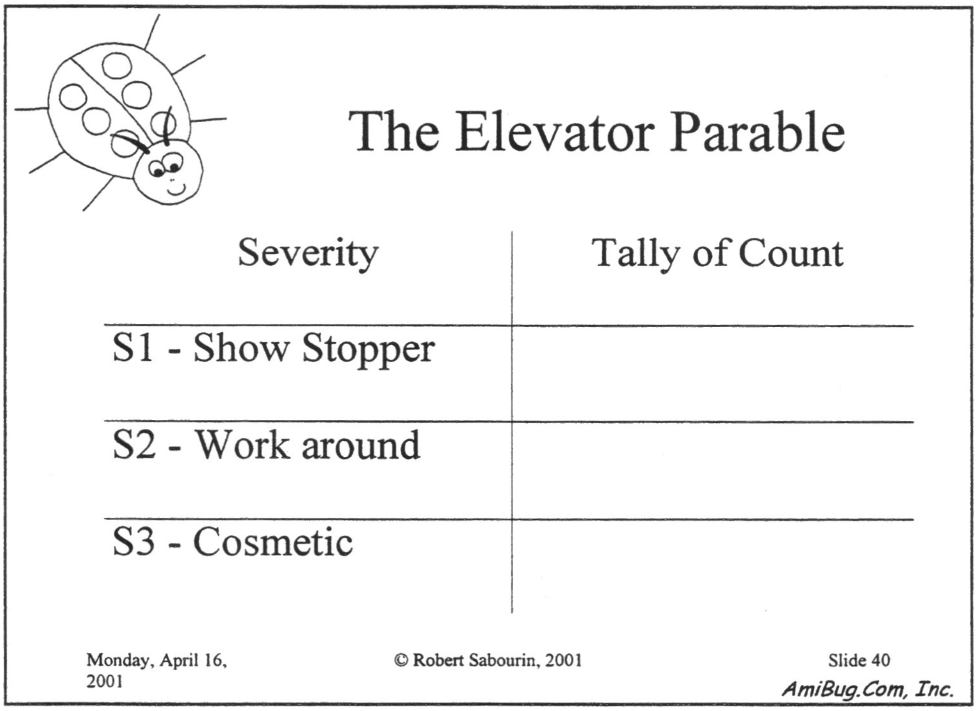
The Elevator Parable

- •News from The Development Lead
- •*Listen to The Folks who write the code*

- •"Profiler is for expert sys admin"
- •"Profiler is a prototype"
- •"Profiler will not be used by docs"
- •"Profiler is an editor for INI files"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 39

AmiBug.Com, Inc.

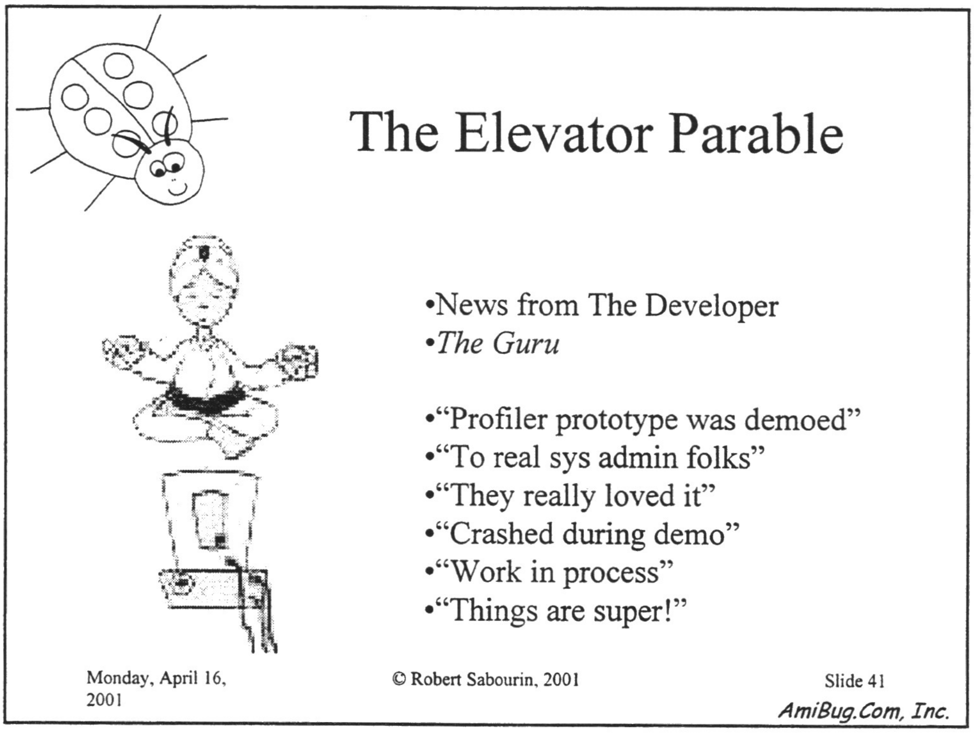


The Elevator Parable

Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 40

AmiBug.Com, Inc.



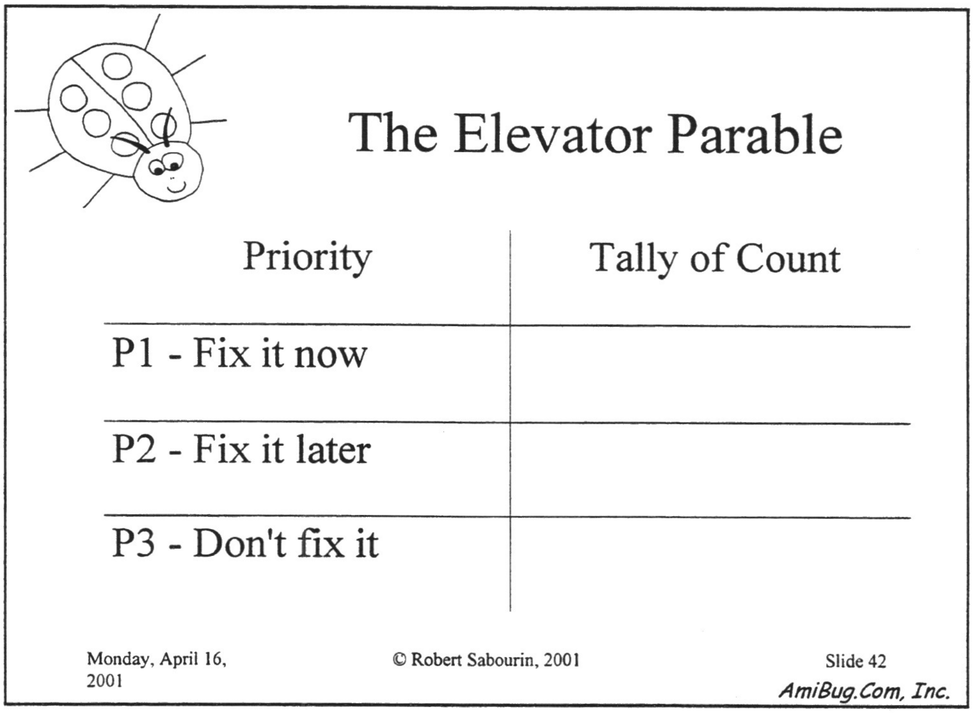
The Elevator Parable

- News from The Developer
- *The Guru*

- "Profiler prototype was demoed"
- "To real sys admin folks"
- "They really loved it"
- "Crashed during demo"
- "Work in process"
- "Things are super!"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 41

AmiBug.Com, Inc.

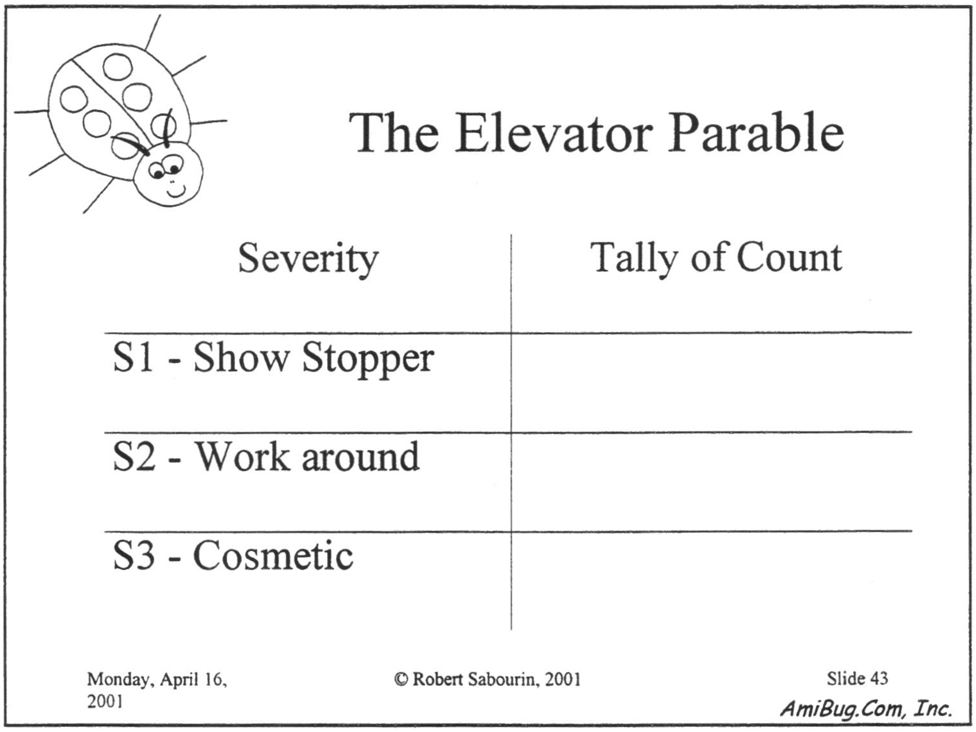


The Elevator Parable

Priority	Tally of Count
P1 - Fix it now	
P2 - Fix it later	
P3 - Don't fix it	

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 42

AmiBug.Com, Inc.

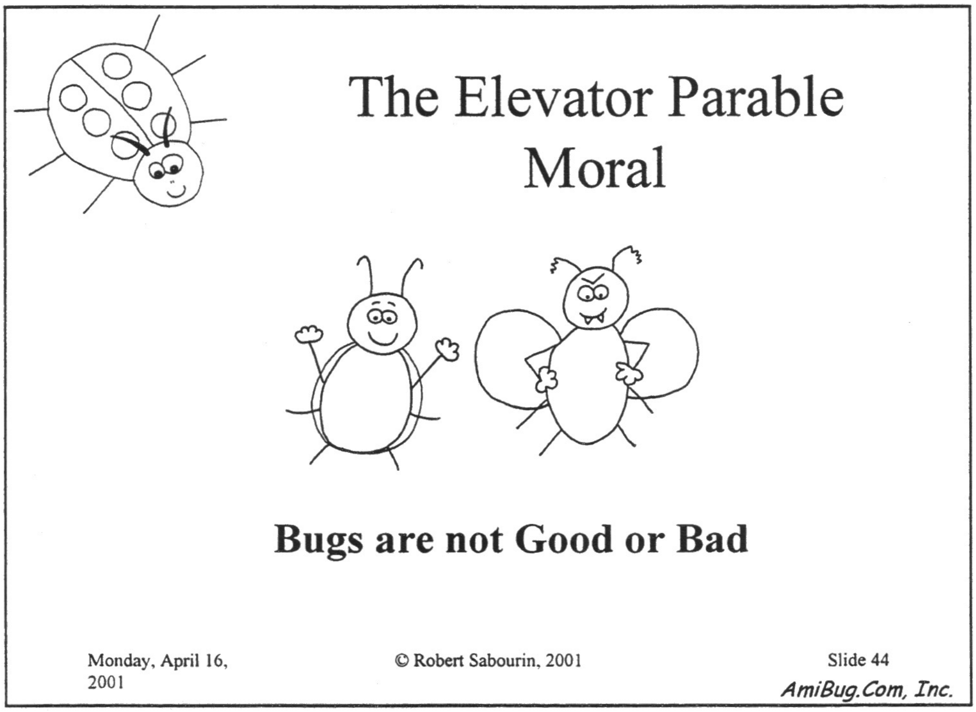


The Elevator Parable

Severity	Tally of Count
S1 - Show Stopper	
S2 - Work around	
S3 - Cosmetic	

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 43

AmiBug.Com, Inc.

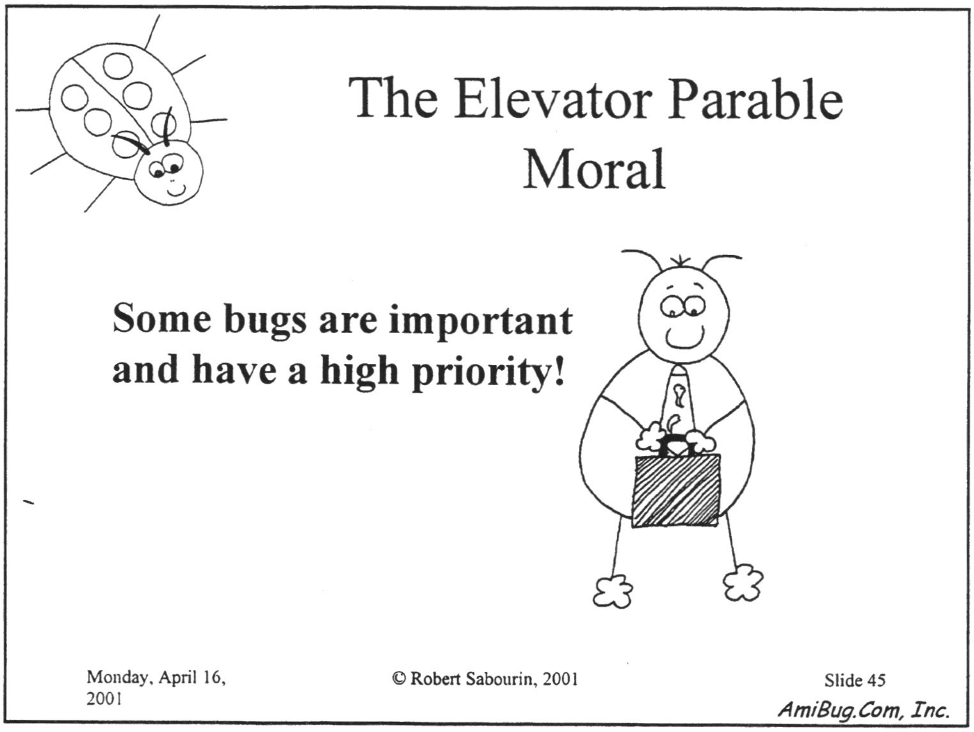


The Elevator Parable Moral

Bugs are not Good or Bad

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 44

AmiBug.Com, Inc.

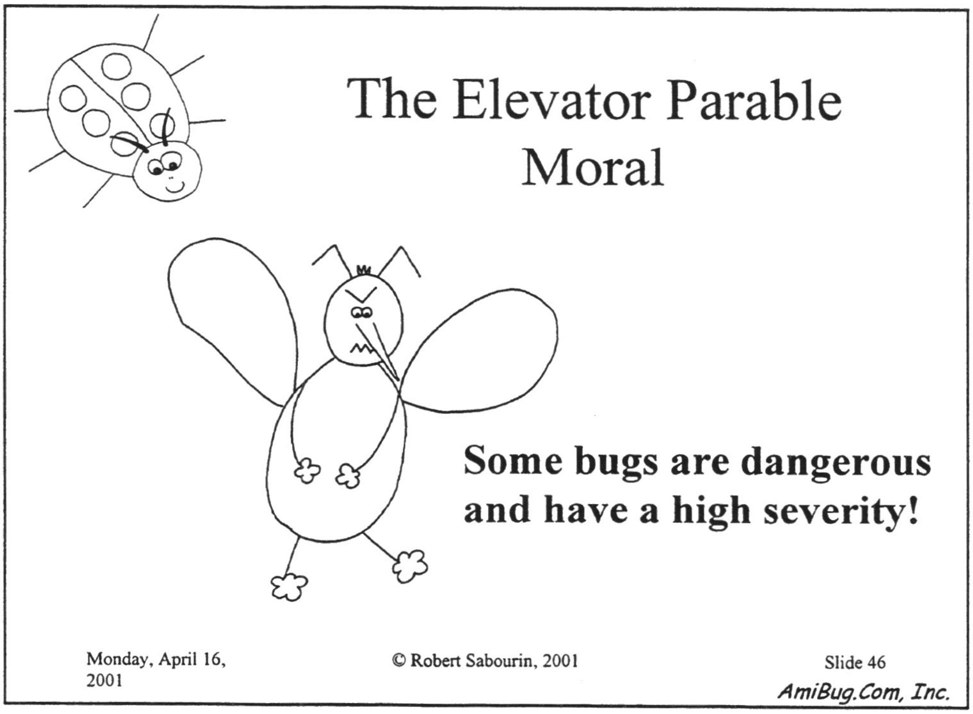


The Elevator Parable Moral

Some bugs are important and have a high priority!

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 45

AmiBug.Com, Inc.

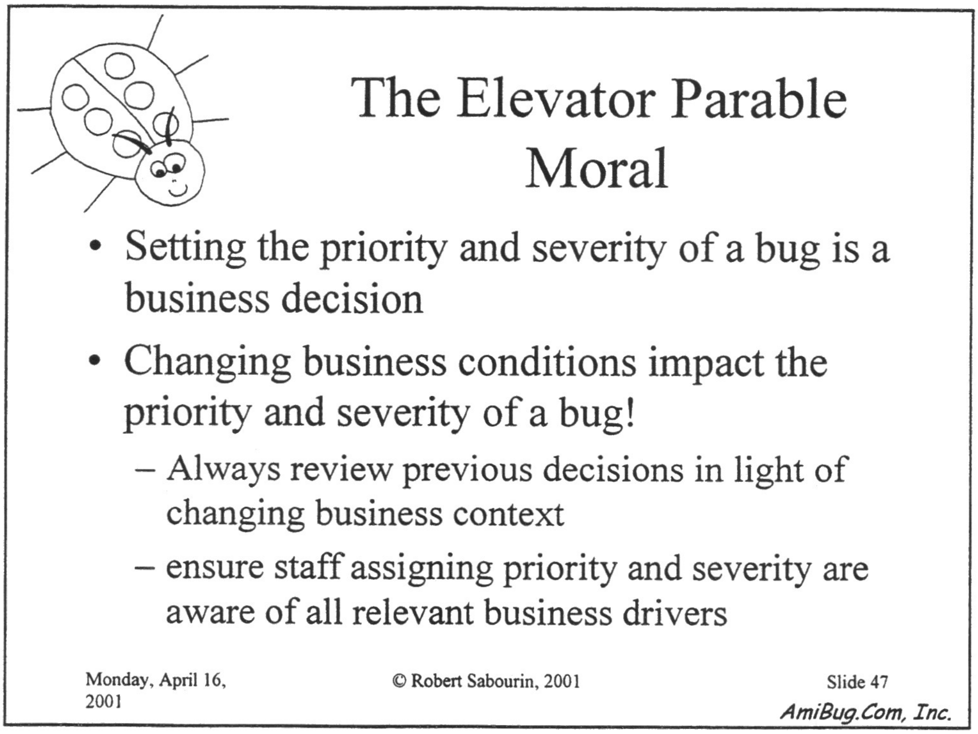


The Elevator Parable Moral

Some bugs are dangerous and have a high severity!

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 46

AmiBug.Com, Inc.

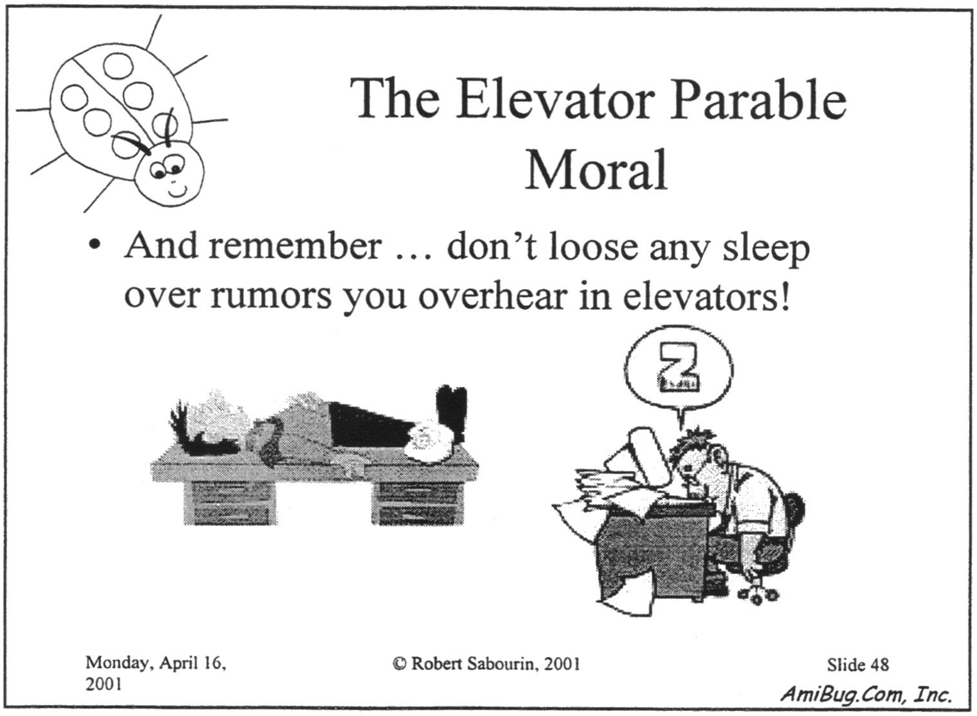


The Elevator Parable Moral

- Setting the priority and severity of a bug is a business decision
- Changing business conditions impact the priority and severity of a bug!
  - Always review previous decisions in light of changing business context
  - ensure staff assigning priority and severity are aware of all relevant business drivers

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 47

AmiBug.Com, Inc.

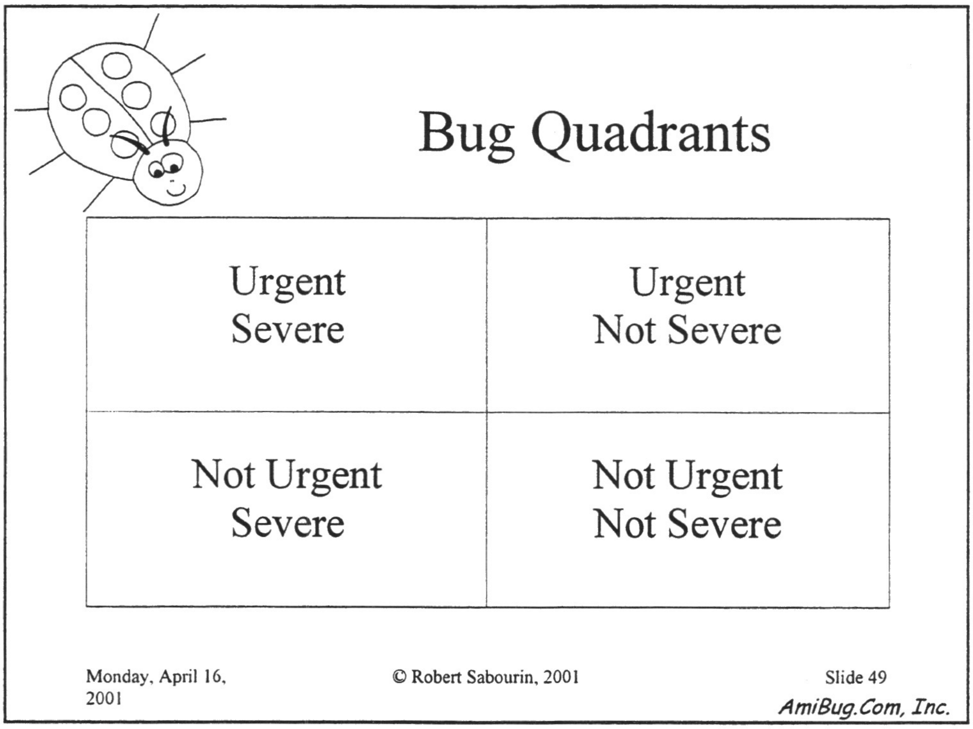


The Elevator Parable Moral

- And remember … don’t loose any sleep over rumors you overhear in elevators!

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 48

AmiBug.Com, Inc.



Bug Quadrants

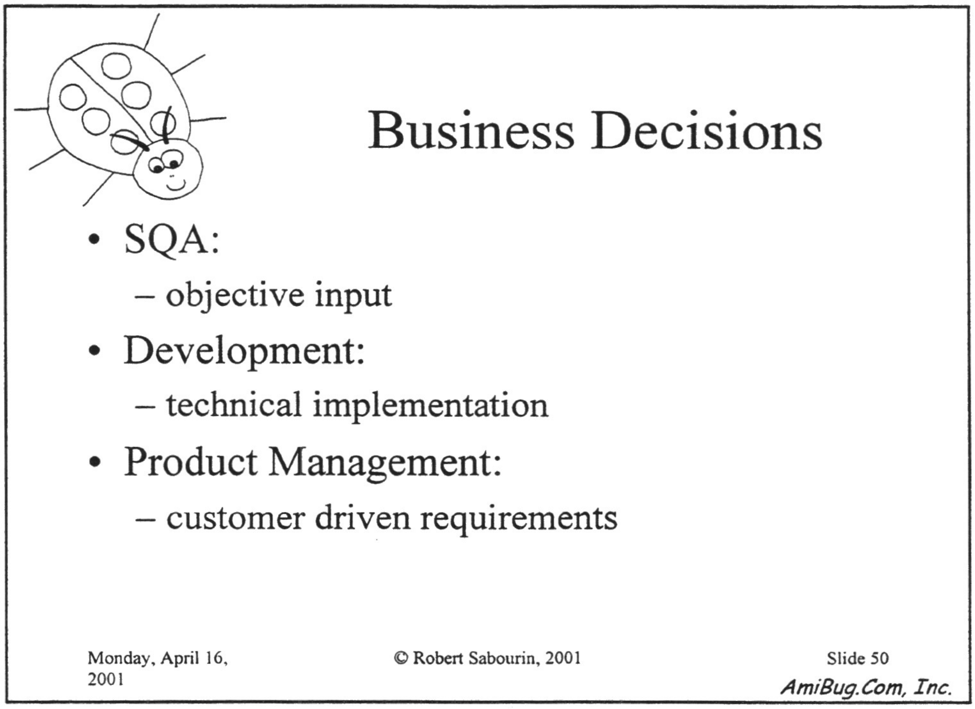
Urgent Severe	Urgent Not Severe
Not Urgent Severe	Not Urgent Not Severe

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 49

AmiBug.Com, Inc.



Business Decisions

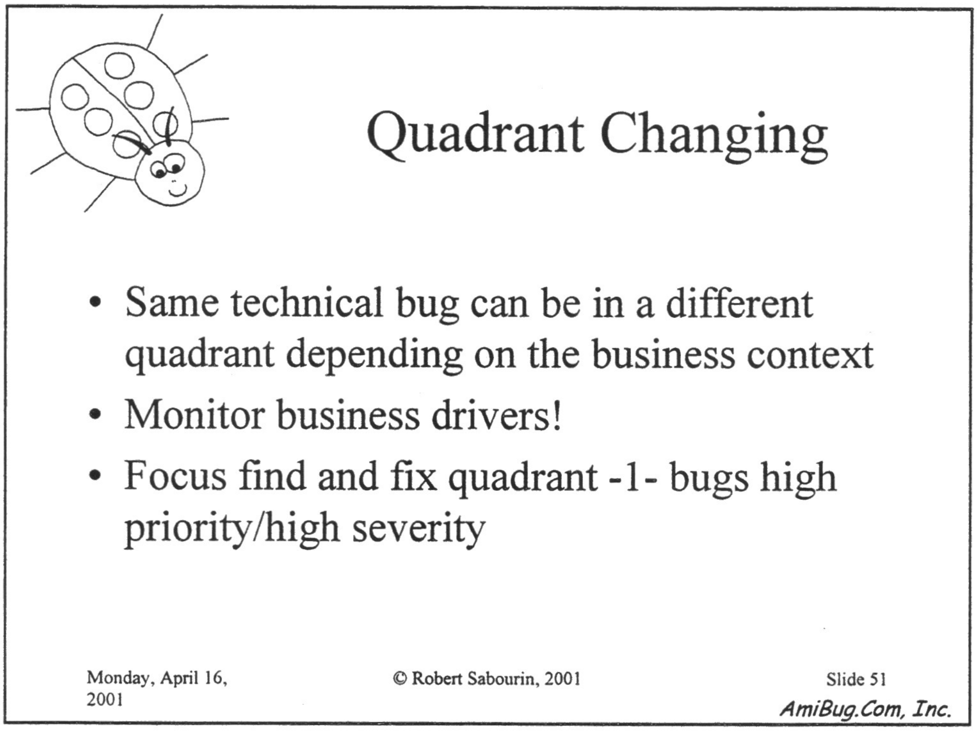
- SQA:
  - objective input
- Development:
  - technical implementation
- Product Management:
  - customer driven requirements

Monday, April 16, 2001

© Robert Sabourin, 2001

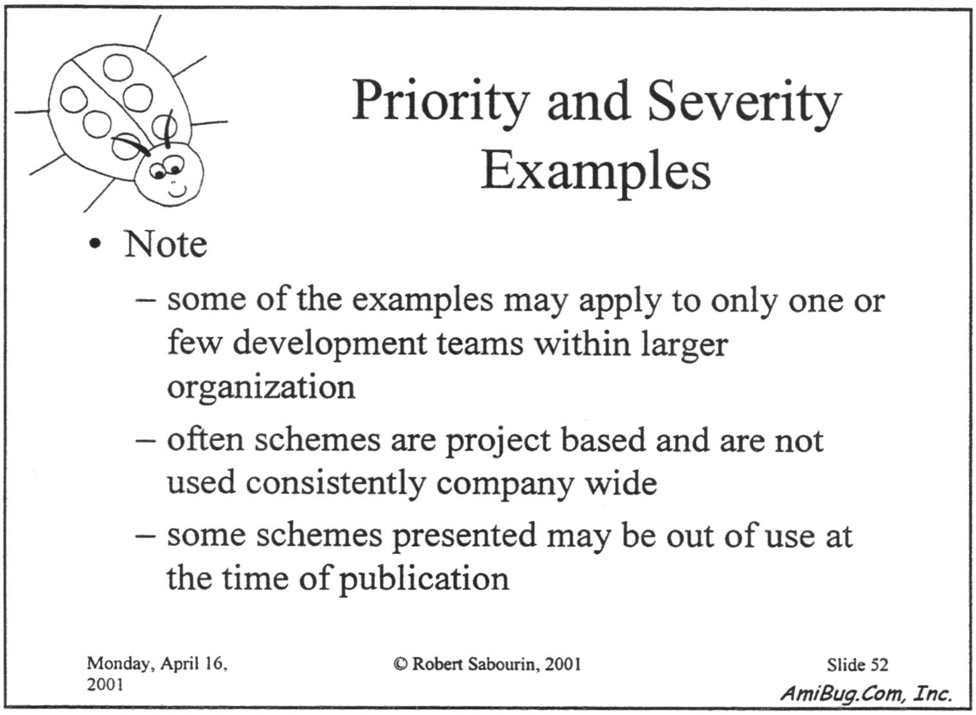
Slide 50

AmiBug.Com, Inc.



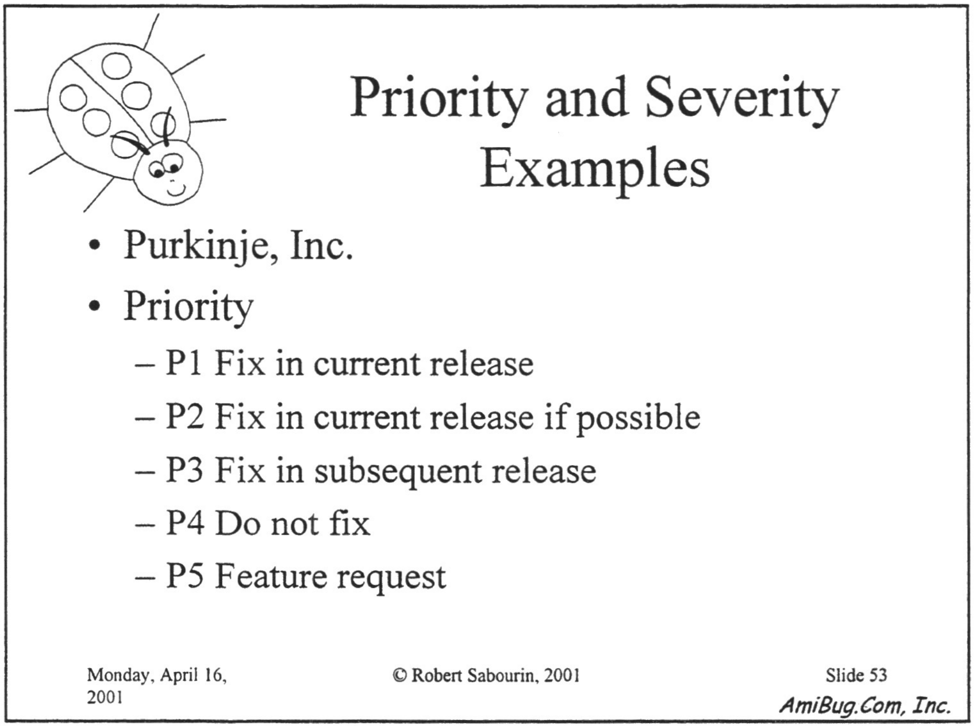
Quadrant Changing

- Same technical bug can be in a different quadrant depending on the business context
- Monitor business drivers!
- Focus find and fix quadrant -1- bugs high priority/high severity



Priority and Severity Examples

- Note
  - some of the examples may apply to only one or few development teams within larger organization
  - often schemes are project based and are not used consistently company wide
  - some schemes presented may be out of use at the time of publication

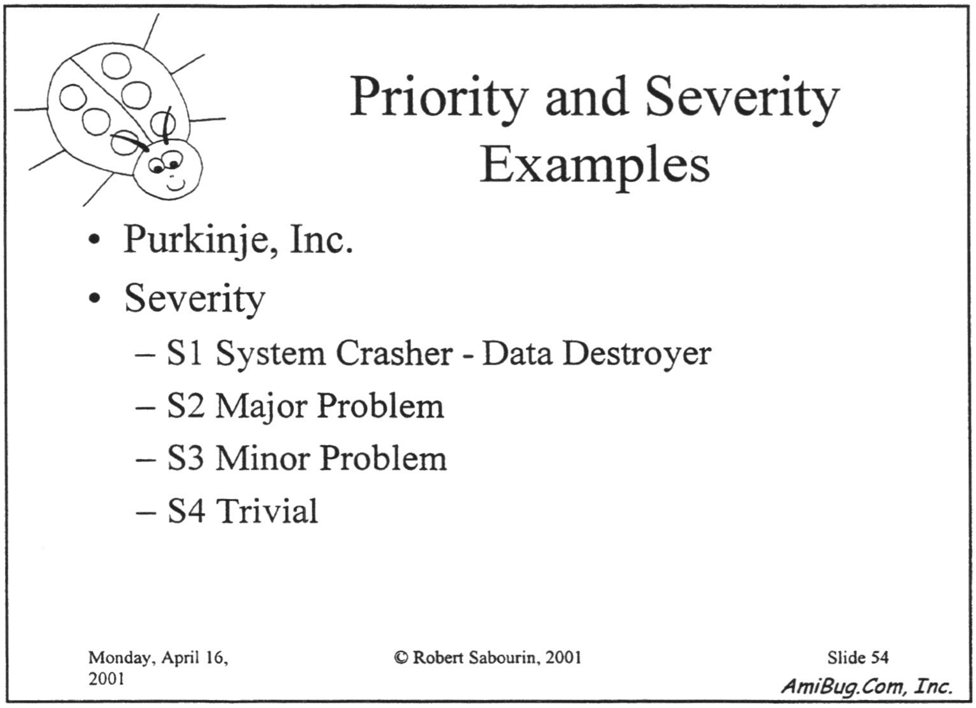


Priority and Severity Examples

- Purkinje, Inc.
- Priority
  - – P1 Fix in current release
  - – P2 Fix in current release if possible
  - – P3 Fix in subsequent release
  - – P4 Do not fix
  - – P5 Feature request

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 53

AmiBug.Com, Inc.

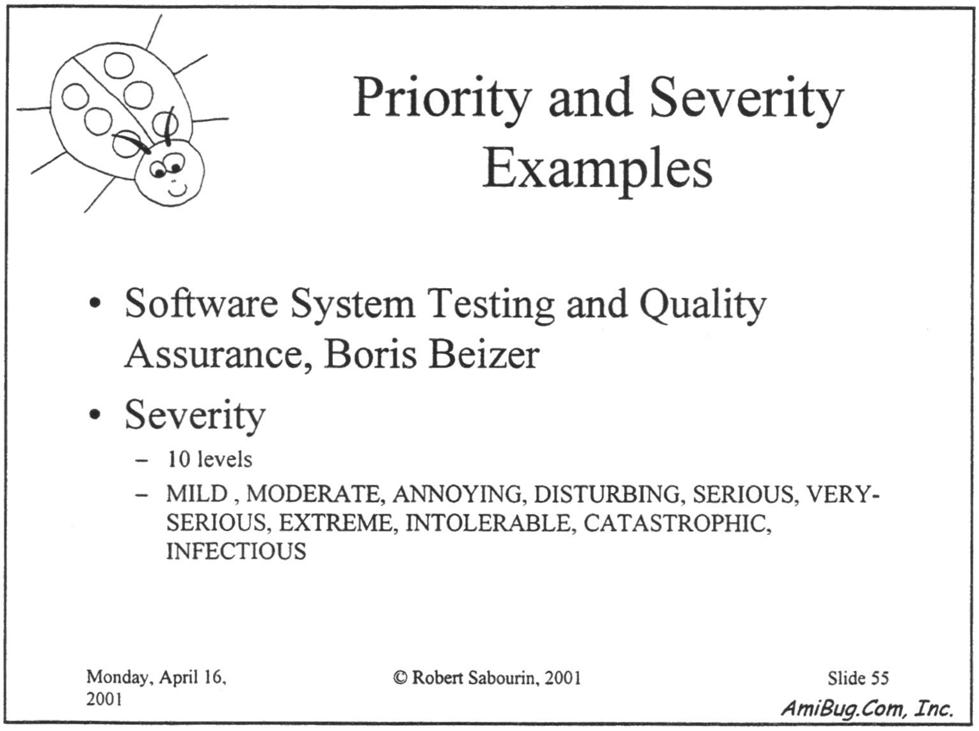


Priority and Severity Examples

- Purkinje, Inc.
- Severity
  - – S1 System Crasher - Data Destroyer
  - – S2 Major Problem
  - – S3 Minor Problem
  - – S4 Trivial

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 54

AmiBug.Com, Inc.

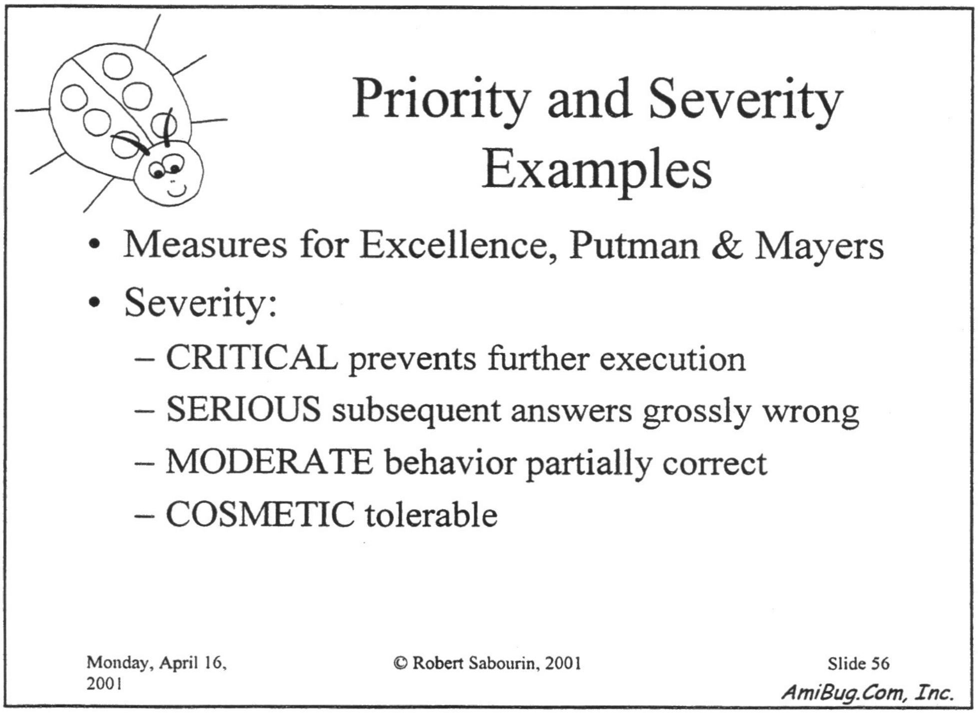


Priority and Severity Examples

- Software System Testing and Quality Assurance, Boris Beizer
- Severity
  - 10 levels
  - MILD , MODERATE, ANNOYING, DISTURBING, SERIOUS, VERY-SERIOUS, EXTREME, INTOLERABLE, CATASTROPHIC, INFECTIOUS

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 55

AmiBug.Com, Inc.

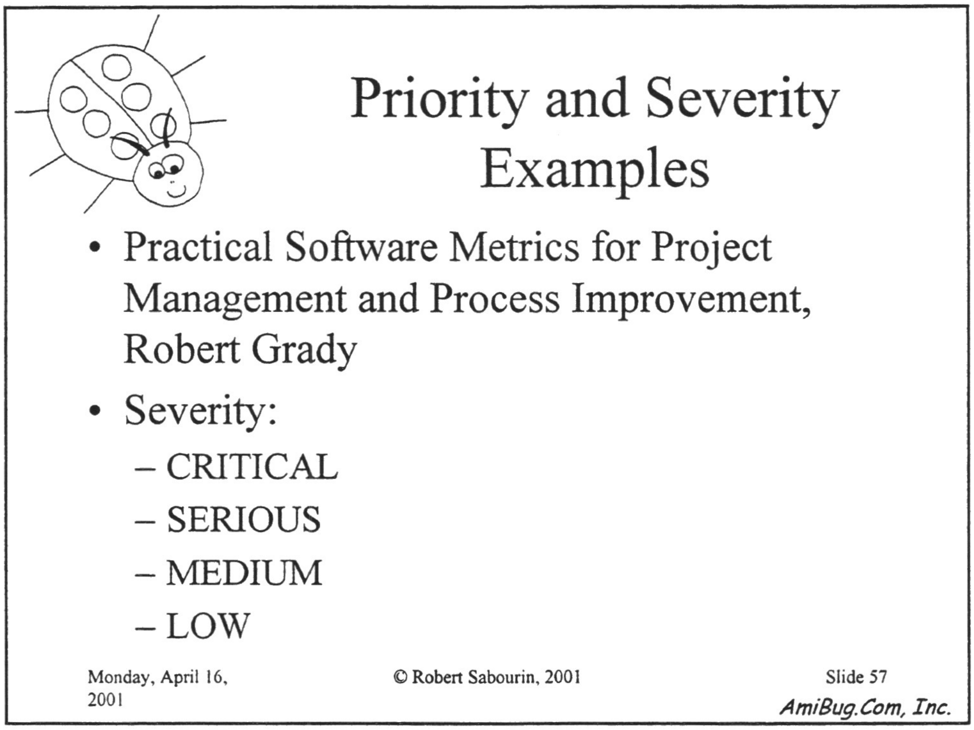


Priority and Severity Examples

- Measures for Excellence, Putman & Mayers
- Severity:
  - CRITICAL prevents further execution
  - SERIOUS subsequent answers grossly wrong
  - MODERATE behavior partially correct
  - COSMETIC tolerable

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 56

AmiBug.Com, Inc.

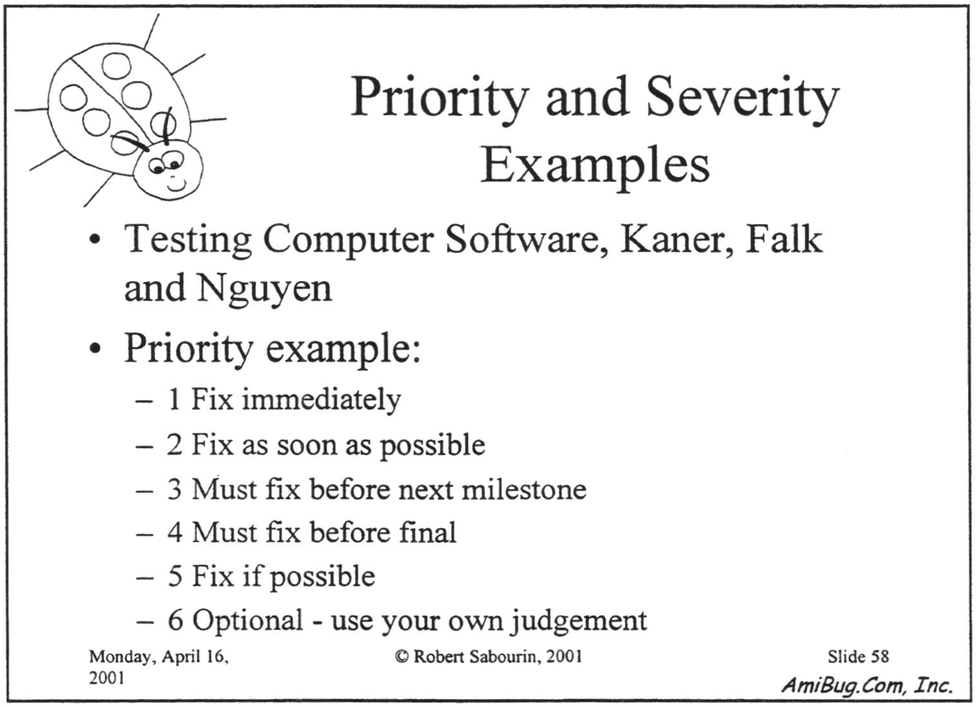


Priority and Severity Examples

- Practical Software Metrics for Project Management and Process Improvement, Robert Grady
- Severity:
  - CRITICAL
  - SERIOUS
  - MEDIUM
  - LOW

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 57

AmiBug.Com, Inc.

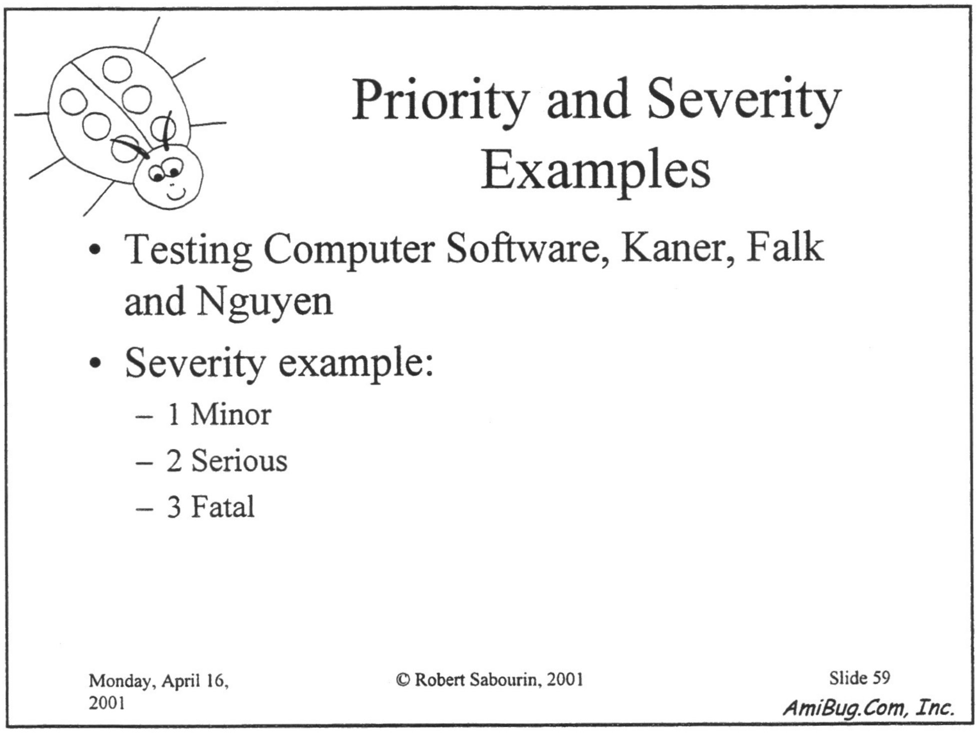


Priority and Severity Examples

- Testing Computer Software, Kaner, Falk and Nguyen
- Priority example:
  - 1 Fix immediately
  - 2 Fix as soon as possible
  - 3 Must fix before next milestone
  - 4 Must fix before final
  - 5 Fix if possible
  - 6 Optional - use your own judgement

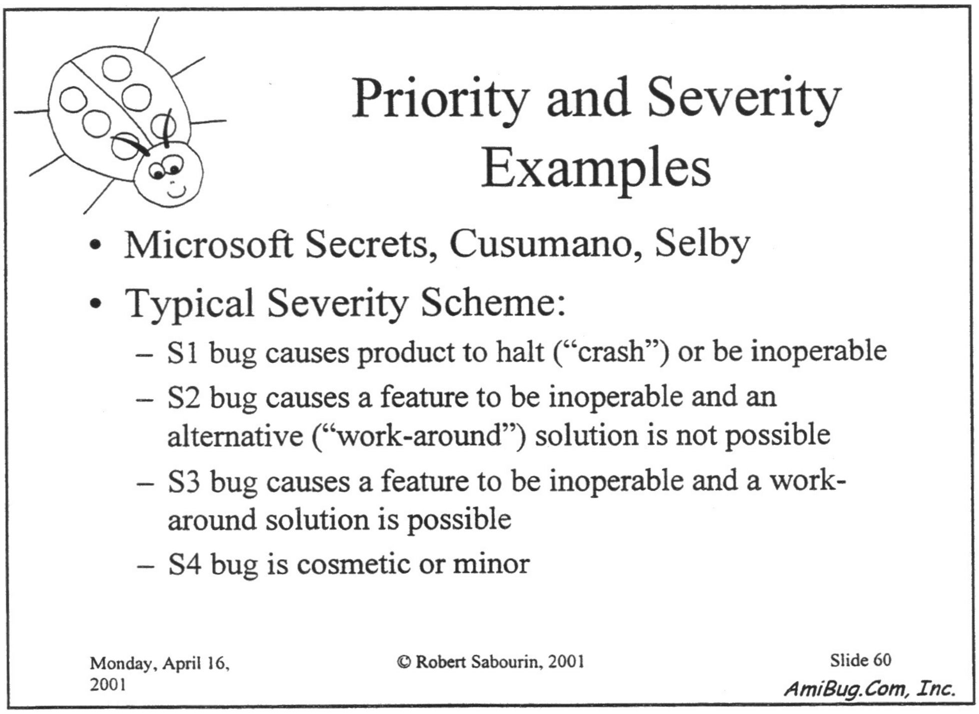
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 58

AmiBug.Com, Inc.



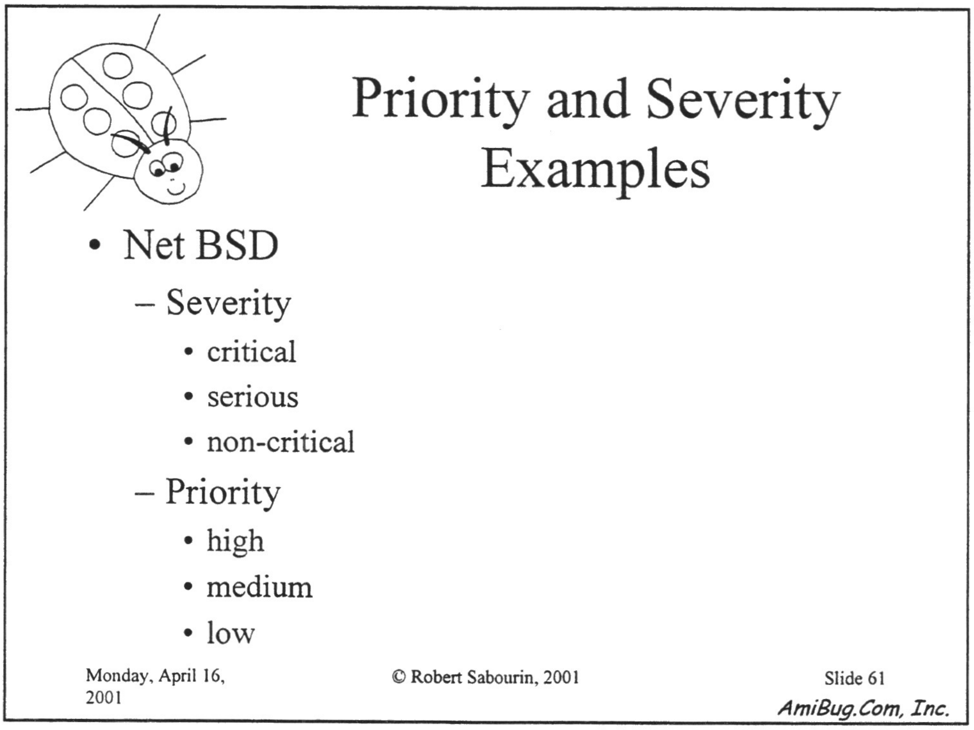
Priority and Severity Examples

- Testing Computer Software, Kaner, Falk and Nguyen
- Severity example:
  - 1 Minor
  - 2 Serious
  - 3 Fatal



Priority and Severity Examples

- Microsoft Secrets, Cusumano, Selby
- Typical Severity Scheme:
  - S1 bug causes product to halt ("crash") or be inoperable
  - S2 bug causes a feature to be inoperable and an alternative ("work-around") solution is not possible
  - S3 bug causes a feature to be inoperable and a work-around solution is possible
  - S4 bug is cosmetic or minor

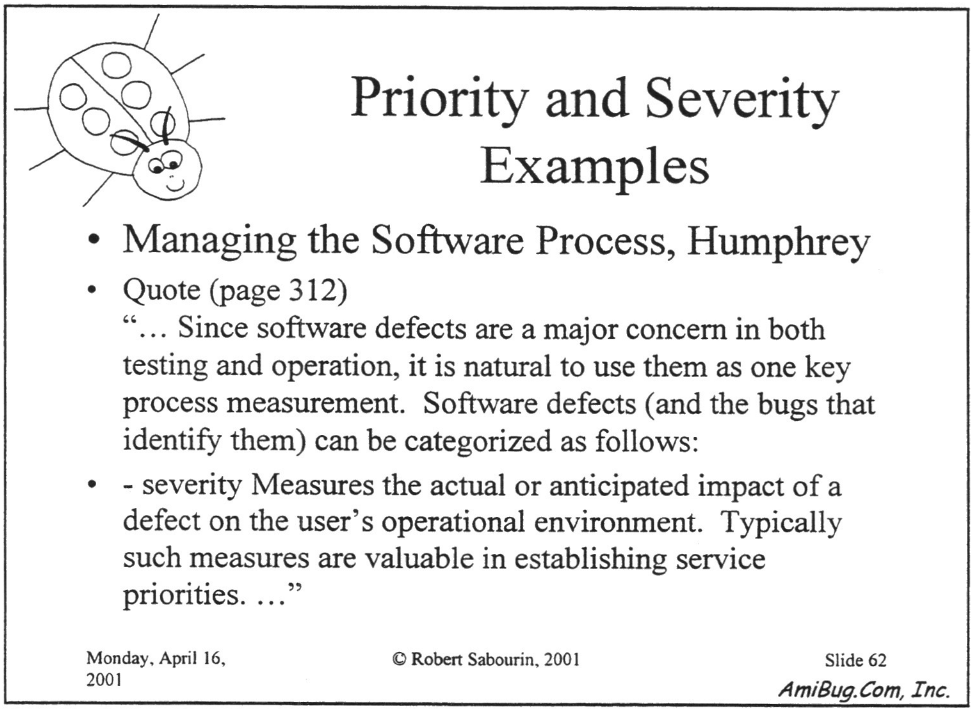


Priority and Severity Examples

- Net BSD
  - Severity
    - critical
    - serious
    - non-critical
  - Priority
    - high
    - medium
    - low

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 61

AmiBug.Com, Inc.

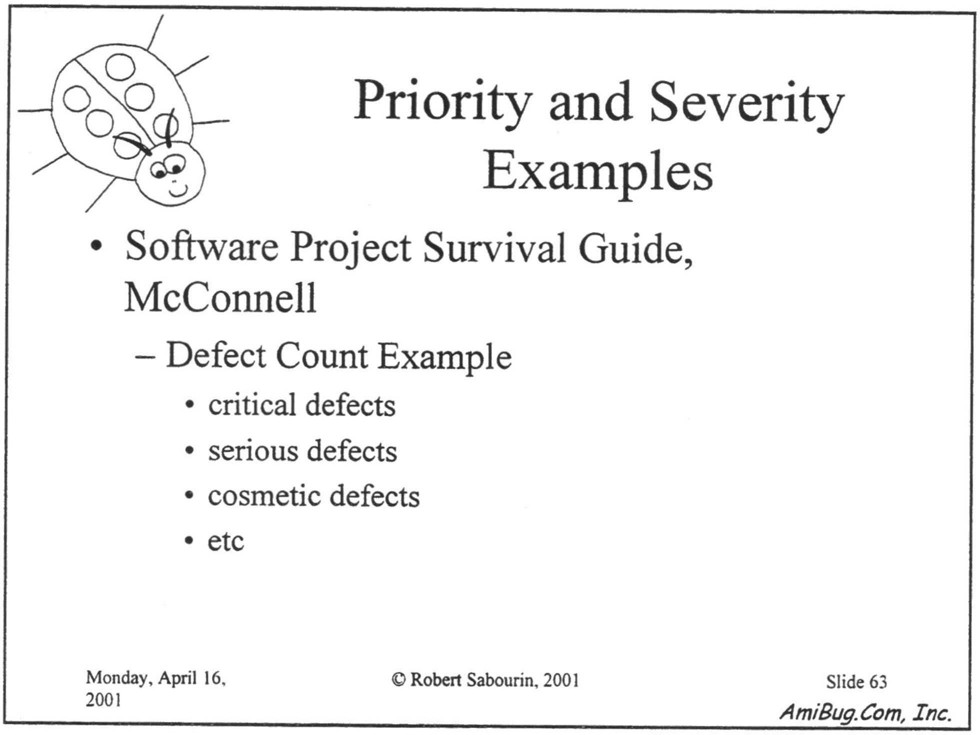


Priority and Severity Examples

- Managing the Software Process, Humphrey
- Quote (page 312)
  "… Since software defects are a major concern in both testing and operation, it is natural to use them as one key process measurement. Software defects (and the bugs that identify them) can be categorized as follows:
- - severity Measures the actual or anticipated impact of a defect on the user's operational environment. Typically such measures are valuable in establishing service priorities. …"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 62

AmiBug.Com, Inc.



Priority and Severity Examples

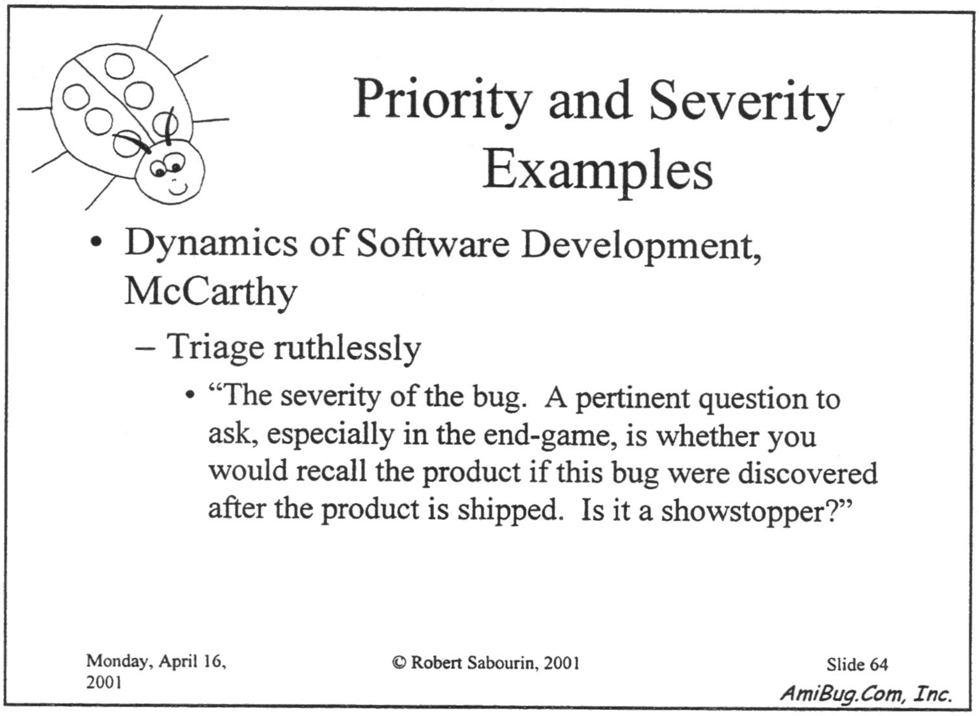
- Software Project Survival Guide, McConnell
  - Defect Count Example
    - critical defects
    - serious defects
    - cosmetic defects
    - etc

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 63

AmiBug.Com, Inc.



Priority and Severity Examples

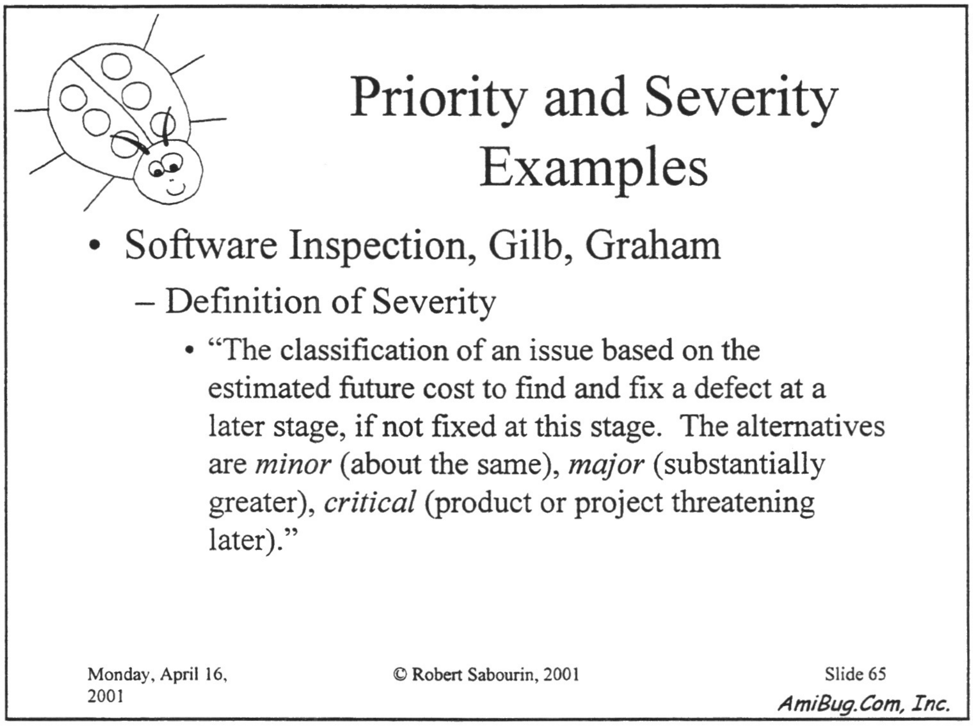
- Dynamics of Software Development, McCarthy
  - Triage ruthlessly
    - "The severity of the bug. A pertinent question to ask, especially in the end-game, is whether you would recall the product if this bug were discovered after the product is shipped. Is it a showstopper?"

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 64

AmiBug.Com, Inc.

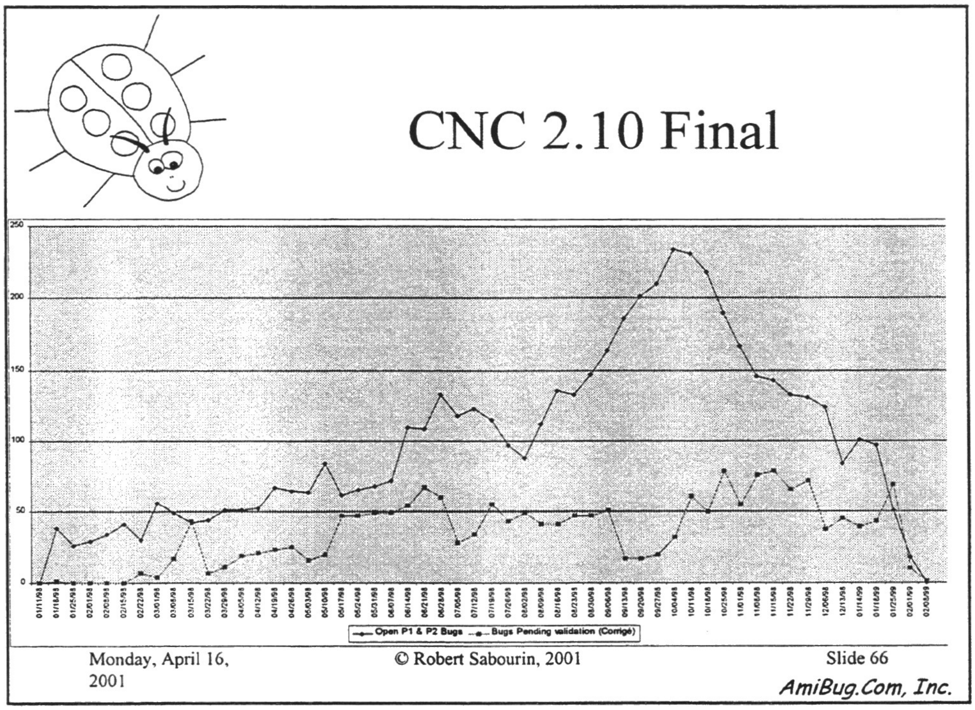


Priority and Severity Examples

- Software Inspection, Gilb, Graham
  - Definition of Severity
    - "The classification of an issue based on the estimated future cost to find and fix a defect at a later stage, if not fixed at this stage. The alternatives are *minor* (about the same), *major* (substantially greater), *critical* (product or project threatening later)."

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 65

AmiBug.Com, Inc.



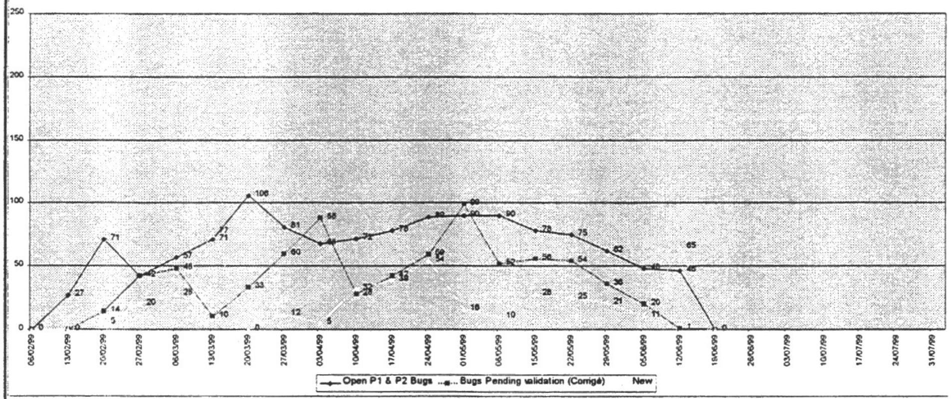
CNC 2.10 Final

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 66

AmiBug.Com, Inc.

CNC 2.20 Final

Defect Status

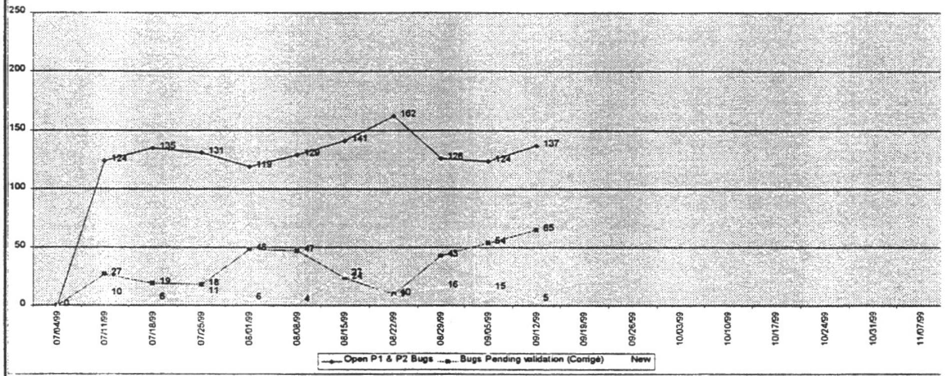


Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 67

AmiBug.Com, Inc.

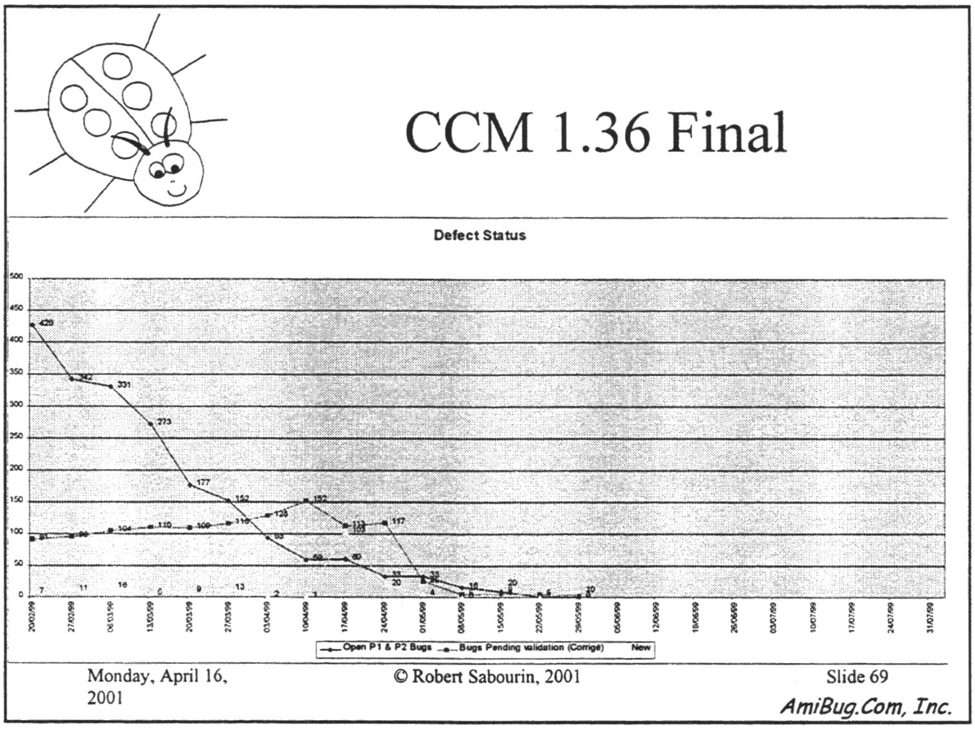
CNC 2.3 In Progress

Defect Status



Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 68

AmiBug.Com, Inc.



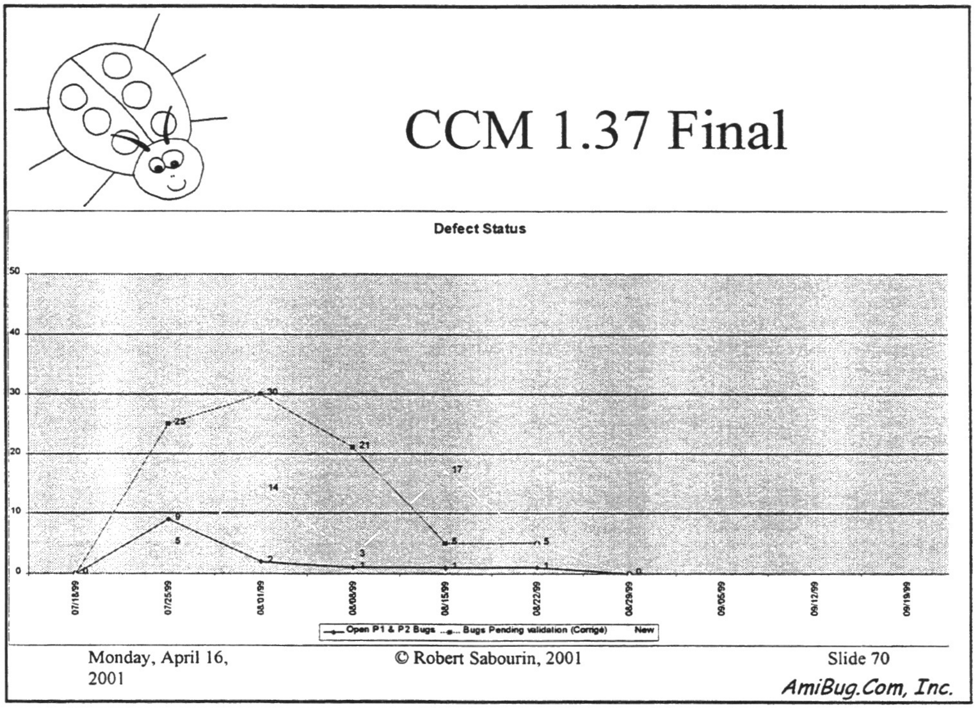
CCM 1.36 Final

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 69

AmiBug.Com, Inc.



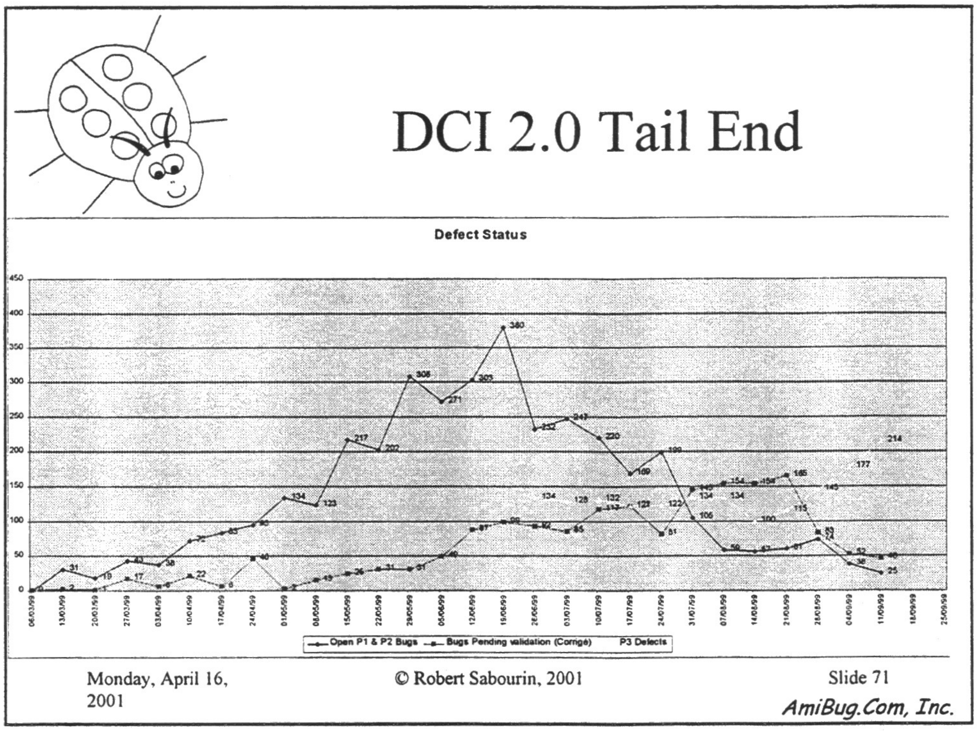
CCM 1.37 Final

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 70

AmiBug.Com, Inc.



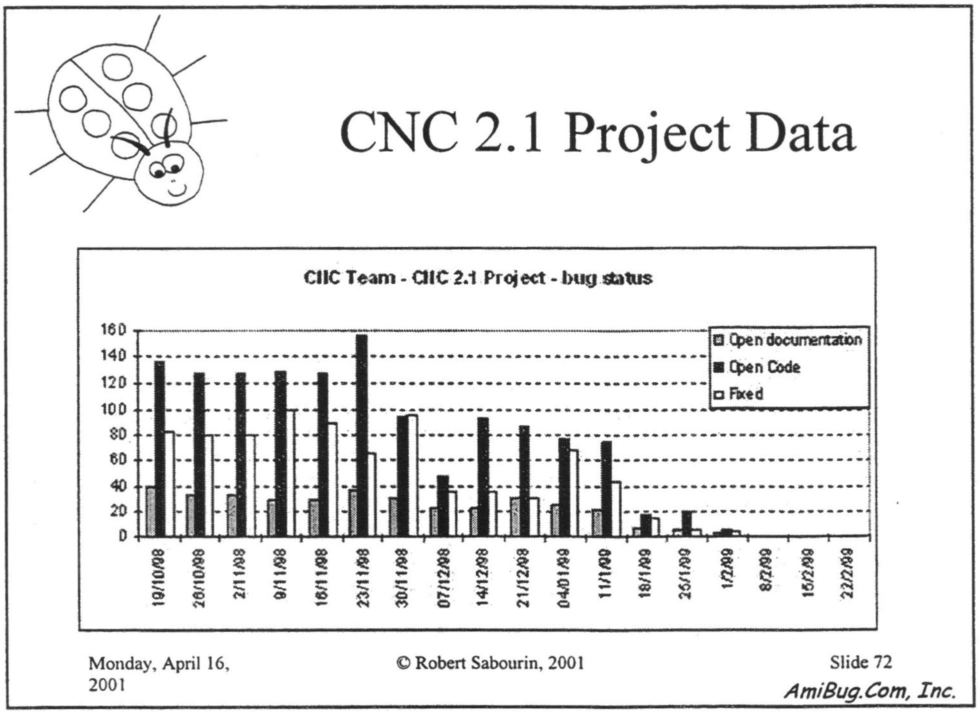
DCI 2.0 Tail End

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 71

AmiBug.Com, Inc.



CNC 2.1 Project Data

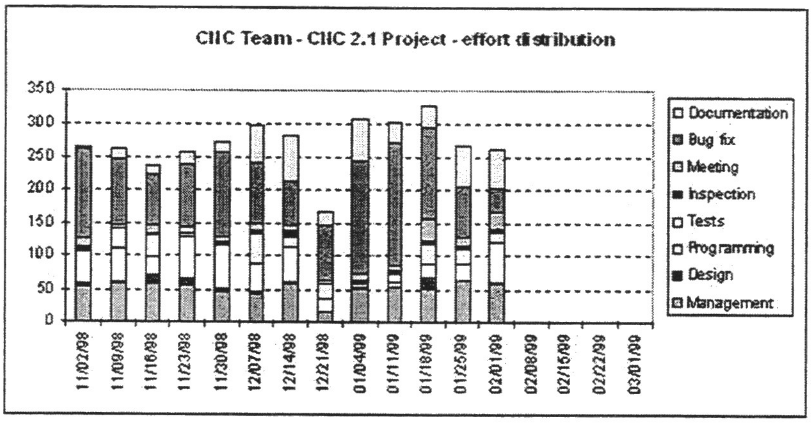
Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 72

AmiBug.Com, Inc.

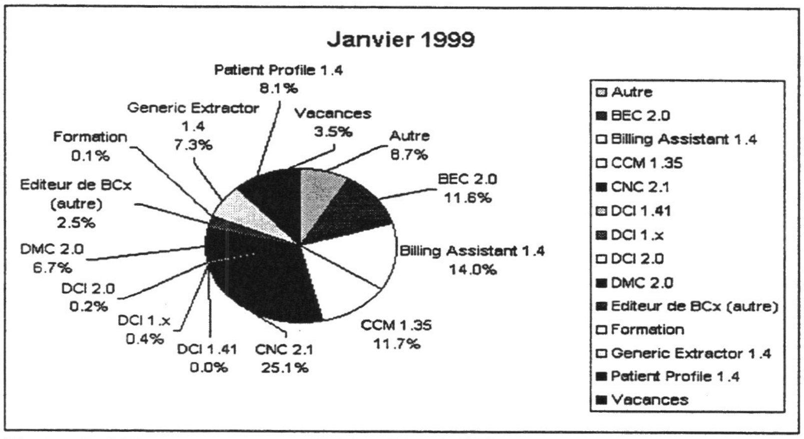
CNC 2.1 Project Data



Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 73

AmiBug.Com, Inc.

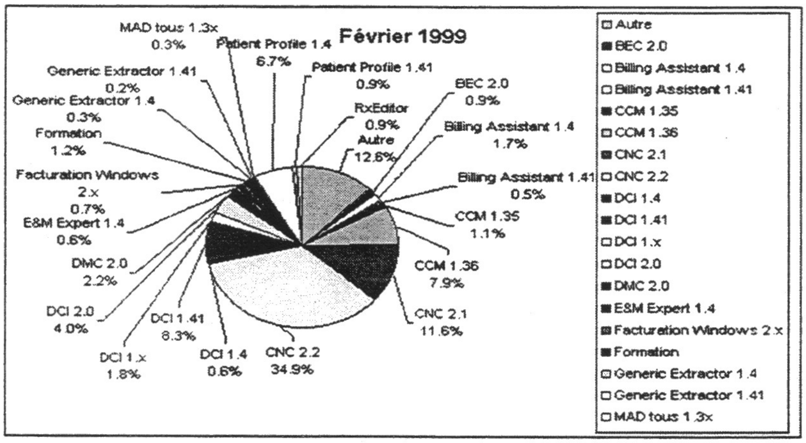
SQA Effort Distributions



Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 74

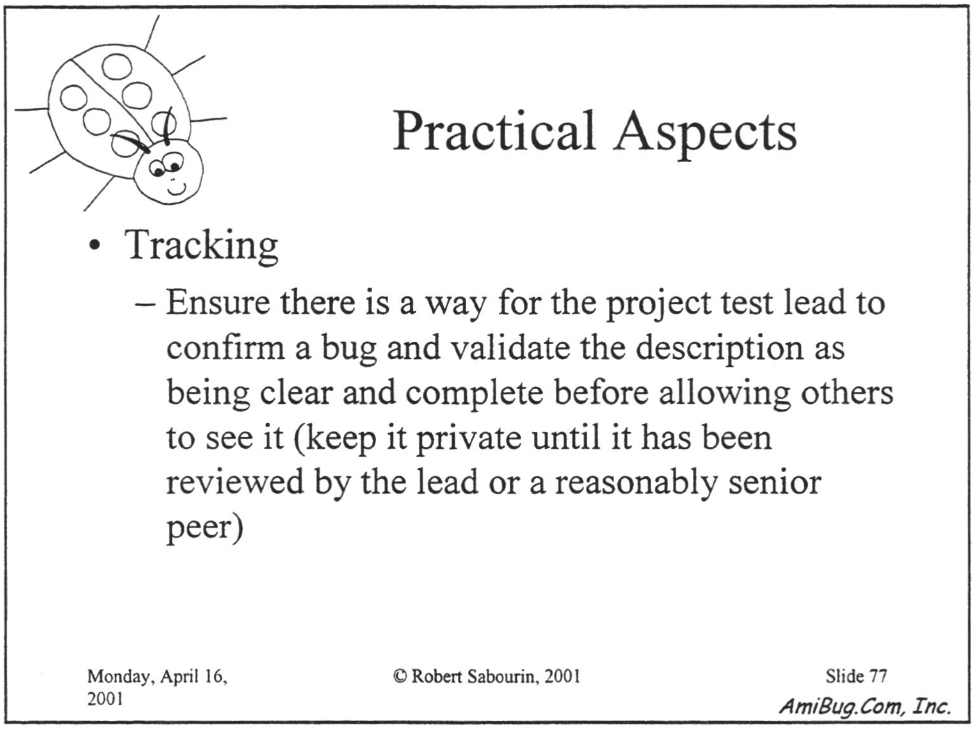
AmiBug.Com, Inc.

SQA Effort Distributions



Practical Aspects

- Tracking
  - Publicize data in graphical or tabular form as much as possible, practical and politically acceptable
  - doors, coffee machines, near the laser printer, near the photocopier
  - update status regularly (at least in sync with builds to SQA)

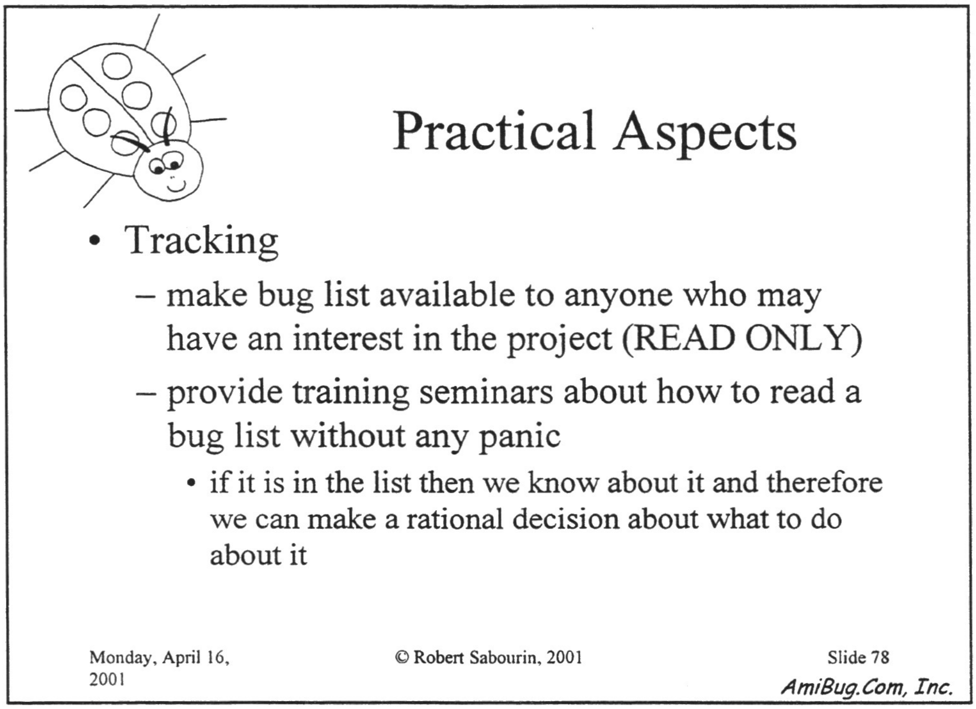


Practical Aspects

- Tracking
  - Ensure there is a way for the project test lead to confirm a bug and validate the description as being clear and complete before allowing others to see it (keep it private until it has been reviewed by the lead or a reasonably senior peer)

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 77

AmiBug.Com, Inc.

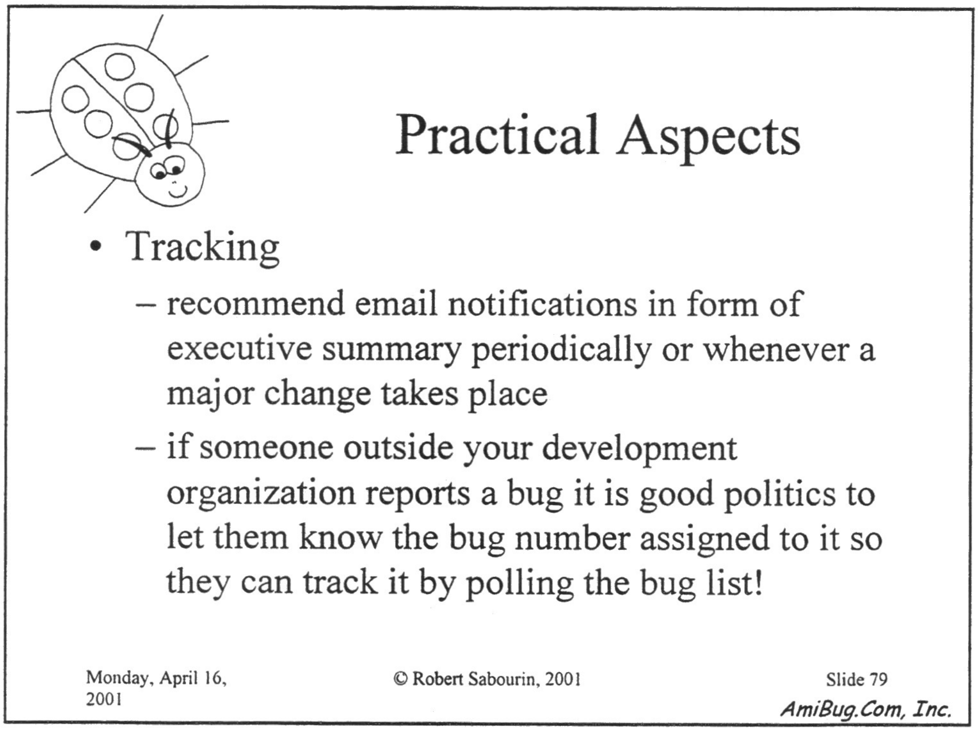


Practical Aspects

- Tracking
  - make bug list available to anyone who may have an interest in the project (READ ONLY)
  - provide training seminars about how to read a bug list without any panic
    - if it is in the list then we know about it and therefore we can make a rational decision about what to do about it

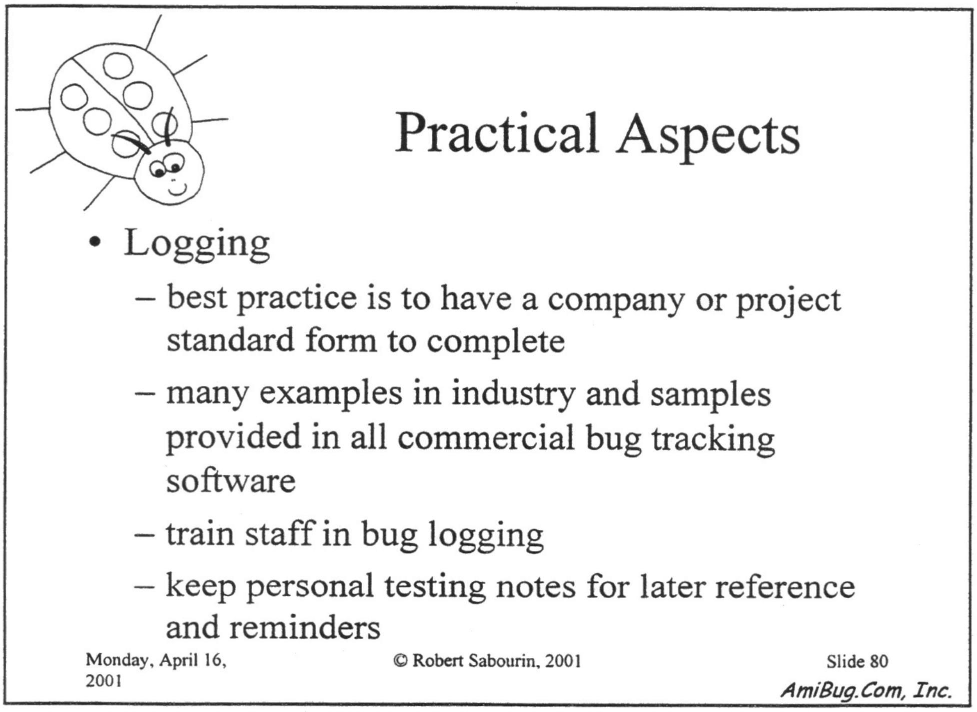
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 78

AmiBug.Com, Inc.



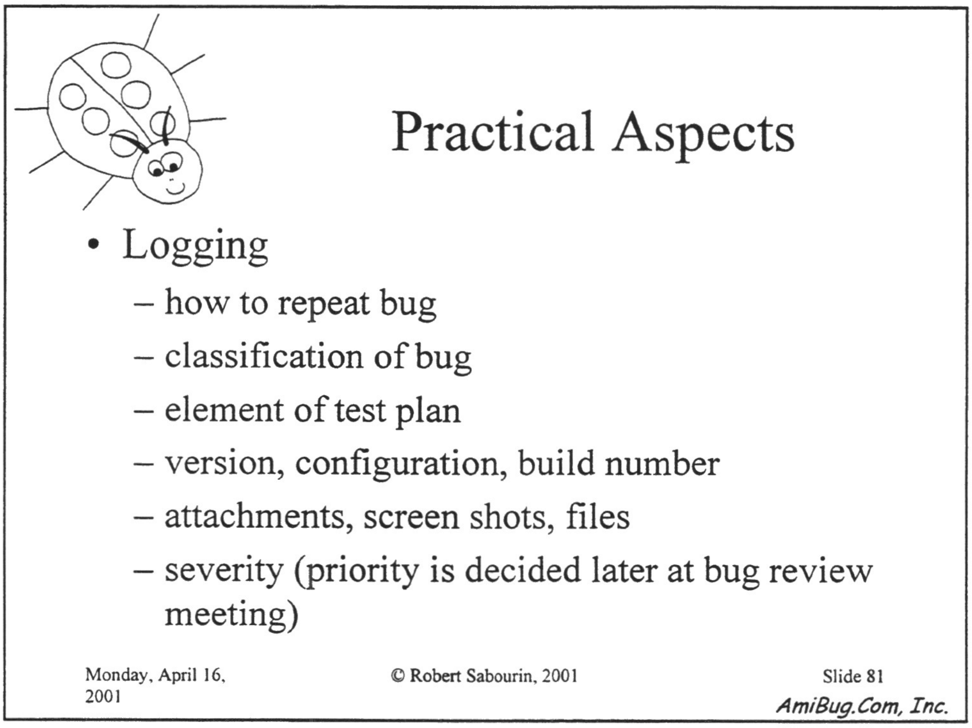
Practical Aspects

- Tracking
  - recommend email notifications in form of executive summary periodically or whenever a major change takes place
  - if someone outside your development organization reports a bug it is good politics to let them know the bug number assigned to it so they can track it by polling the bug list!



Practical Aspects

- Logging
  - best practice is to have a company or project standard form to complete
  - many examples in industry and samples provided in all commercial bug tracking software
  - train staff in bug logging
  - keep personal testing notes for later reference and reminders

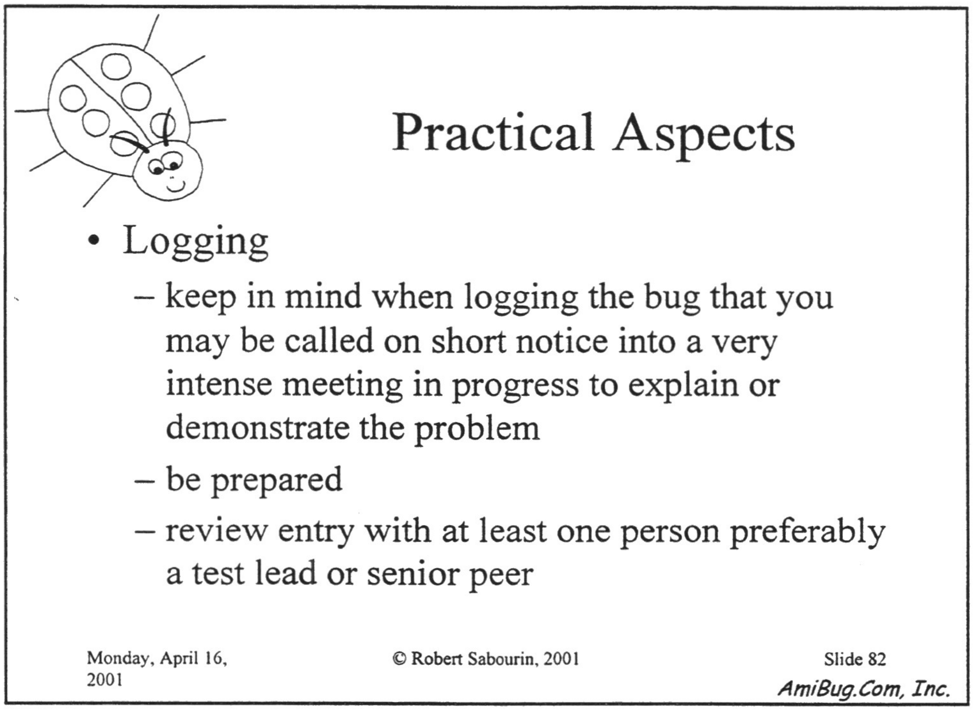


Practical Aspects

- Logging
  - – how to repeat bug
  - – classification of bug
  - – element of test plan
  - – version, configuration, build number
  - – attachments, screen shots, files
  - – severity (priority is decided later at bug review meeting)

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 81

AmiBug.Com, Inc.

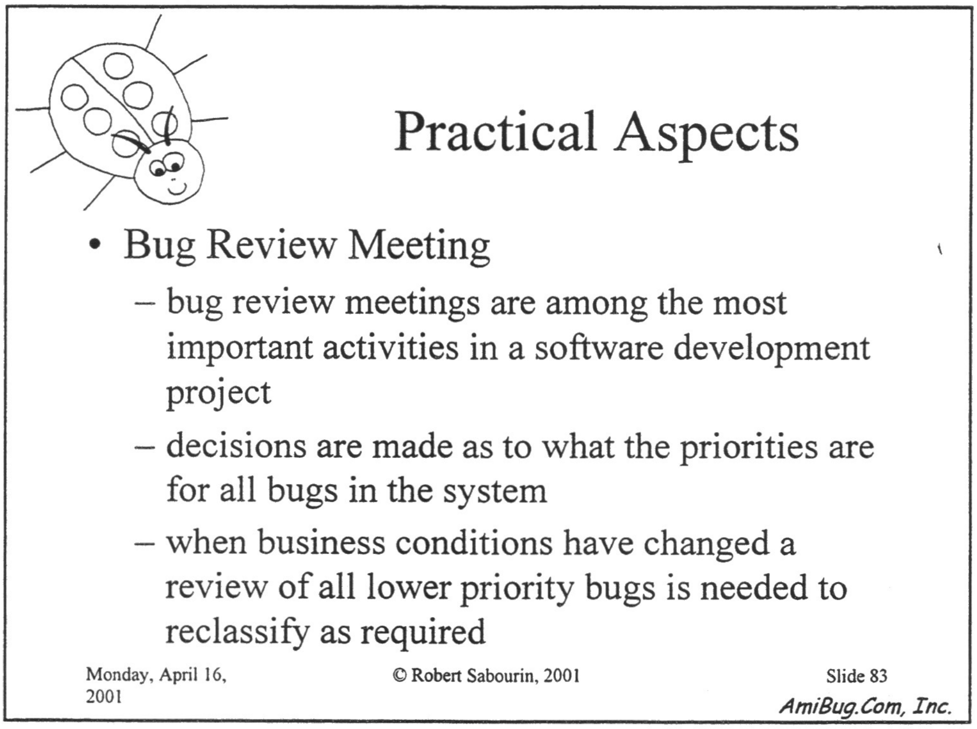


Practical Aspects

- Logging
  - – keep in mind when logging the bug that you may be called on short notice into a very intense meeting in progress to explain or demonstrate the problem
  - – be prepared
  - – review entry with at least one person preferably a test lead or senior peer

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 82

AmiBug.Com, Inc.



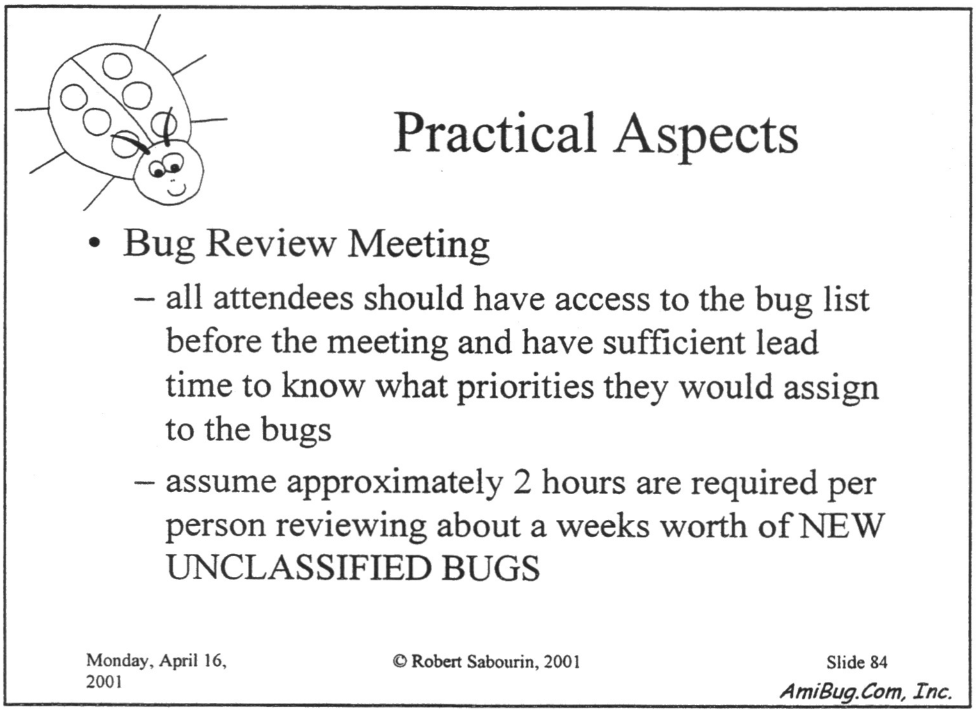
Practical Aspects

- Bug Review Meeting
  - bug review meetings are among the most important activities in a software development project
  - decisions are made as to what the priorities are for all bugs in the system
  - when business conditions have changed a review of all lower priority bugs is needed to reclassify as required

Monday, April 16, 2001

Slide 83

AmiBug.Com, Inc.



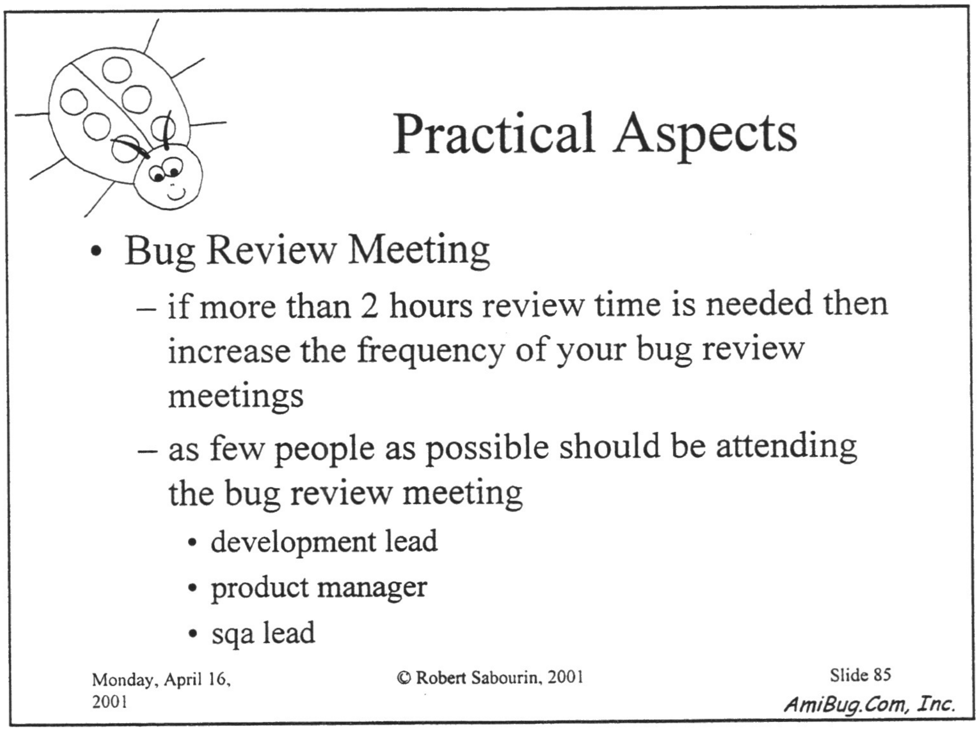
Practical Aspects

- Bug Review Meeting
  - all attendees should have access to the bug list before the meeting and have sufficient lead time to know what priorities they would assign to the bugs
  - assume approximately 2 hours are required per person reviewing about a weeks worth of NEW UNCLASSIFIED BUGS

Monday, April 16, 2001

Slide 84

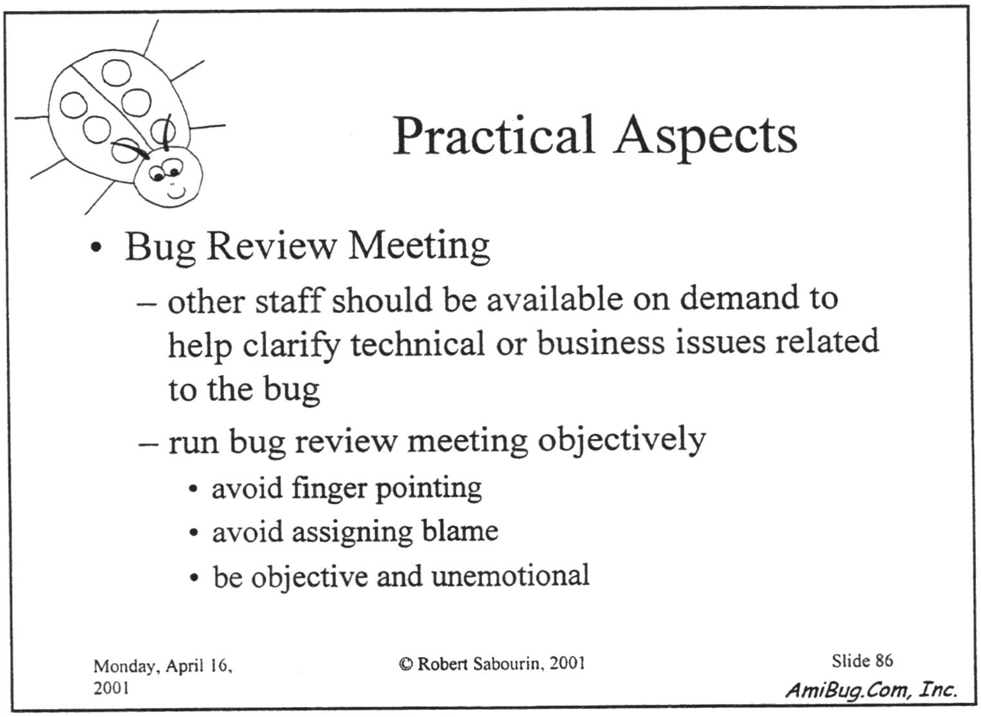
AmiBug.Com, Inc.



Practical Aspects

- Bug Review Meeting
  - if more than 2 hours review time is needed then increase the frequency of your bug review meetings
  - as few people as possible should be attending the bug review meeting
    - development lead
    - product manager
    - sqa lead

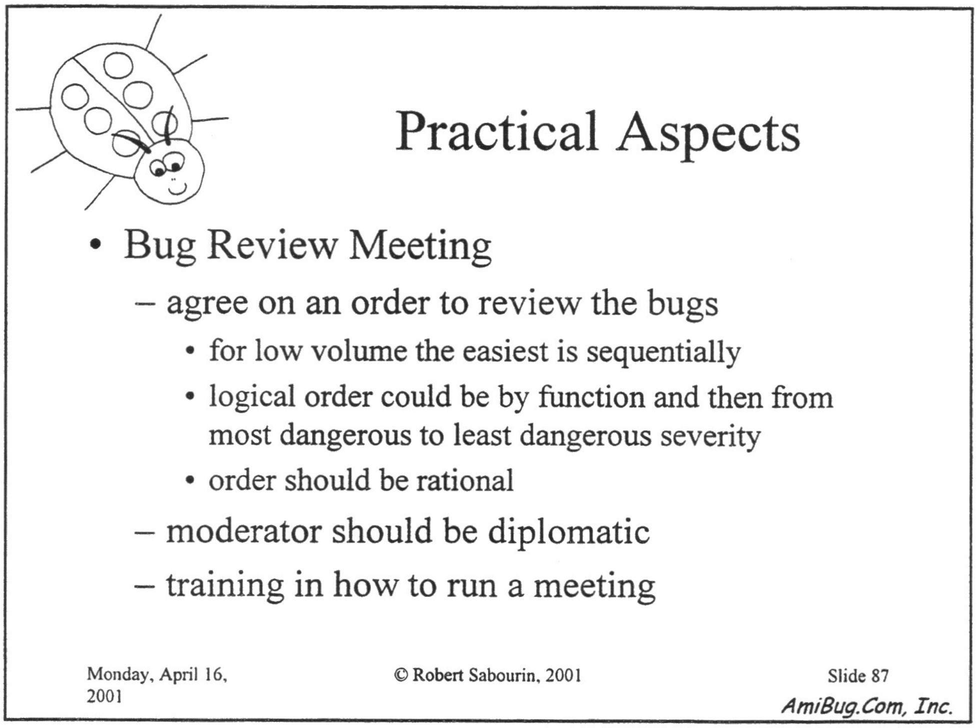
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 85
AmiBug.Com, Inc.



Practical Aspects

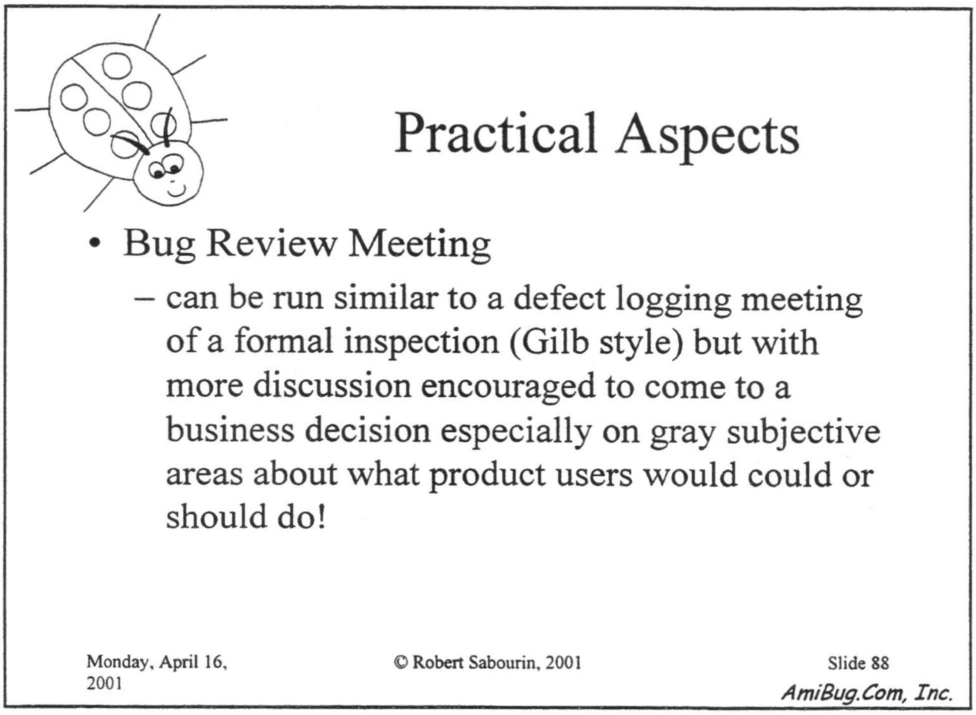
- Bug Review Meeting
  - other staff should be available on demand to help clarify technical or business issues related to the bug
  - run bug review meeting objectively
    - avoid finger pointing
    - avoid assigning blame
    - be objective and unemotional

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 86
AmiBug.Com, Inc.



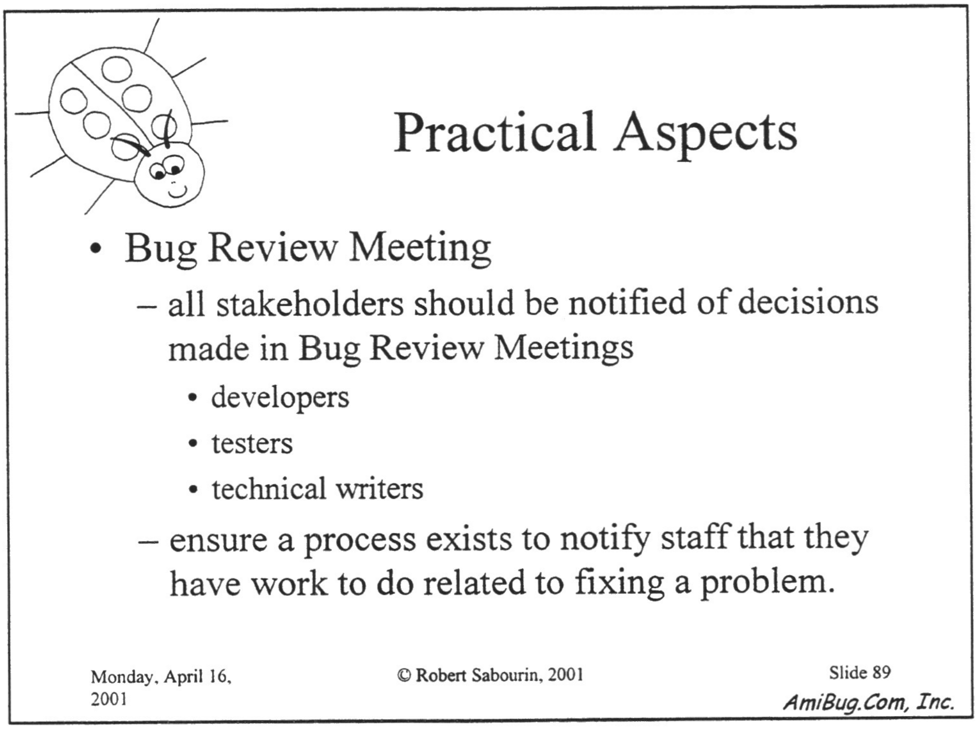
Practical Aspects

- Bug Review Meeting
  - agree on an order to review the bugs
    - for low volume the easiest is sequentially
    - logical order could be by function and then from most dangerous to least dangerous severity
    - order should be rational
  - moderator should be diplomatic
  - training in how to run a meeting



Practical Aspects

- Bug Review Meeting
  - can be run similar to a defect logging meeting of a formal inspection (Gilb style) but with more discussion encouraged to come to a business decision especially on gray subjective areas about what product users would could or should do!

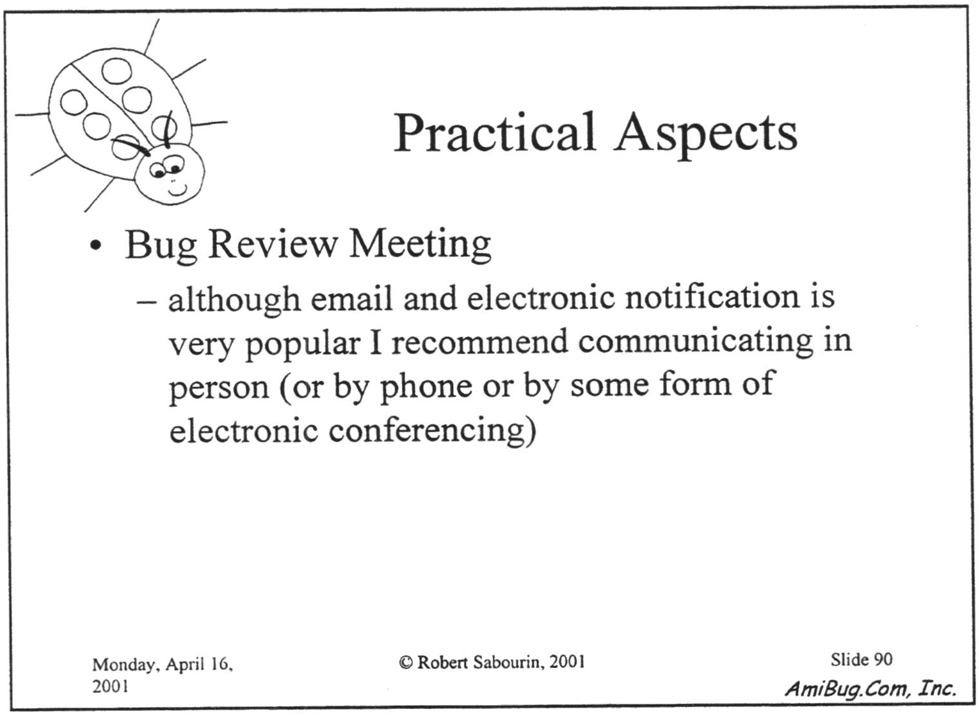


Practical Aspects

- Bug Review Meeting
  - all stakeholders should be notified of decisions made in Bug Review Meetings
    - developers
    - testers
    - technical writers
  - ensure a process exists to notify staff that they have work to do related to fixing a problem.

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 89

AmiBug.Com, Inc.

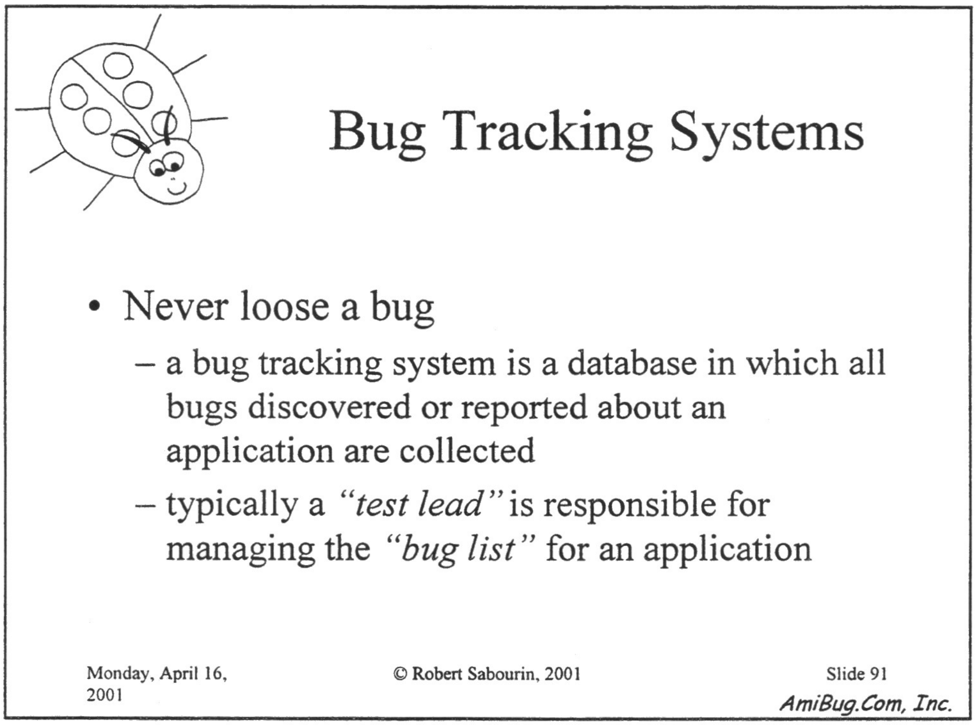


Practical Aspects

- Bug Review Meeting
  - although email and electronic notification is very popular I recommend communicating in person (or by phone or by some form of electronic conferencing)

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 90

AmiBug.Com, Inc.

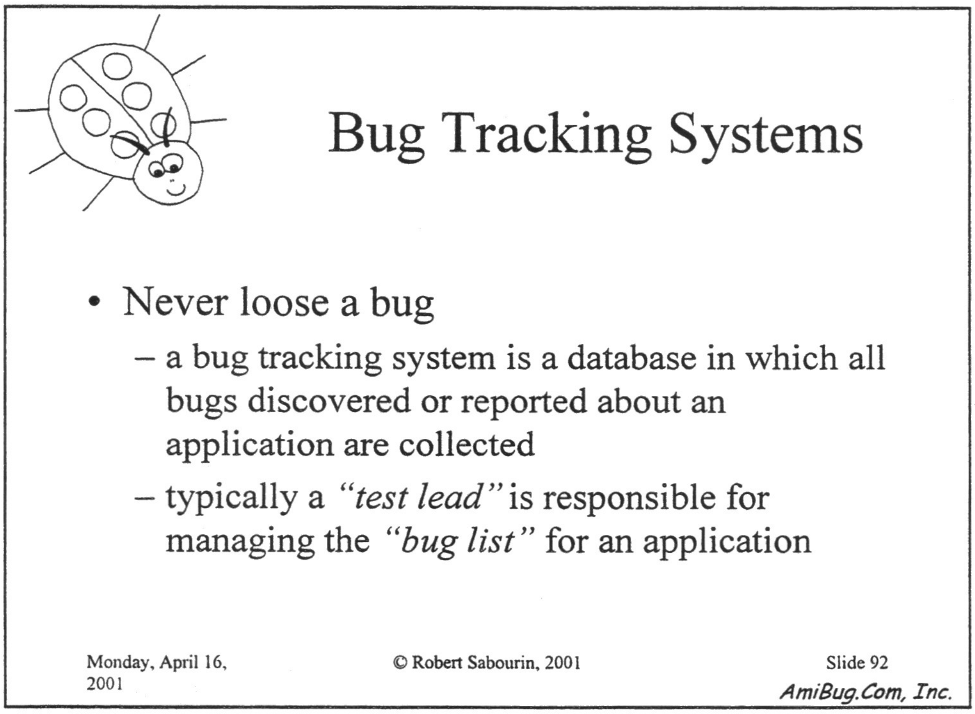


Bug Tracking Systems

- Never loose a bug
  - a bug tracking system is a database in which all bugs discovered or reported about an application are collected
  - typically a *"test lead"* is responsible for managing the *"bug list"* for an application

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 91

AmiBug.Com, Inc.

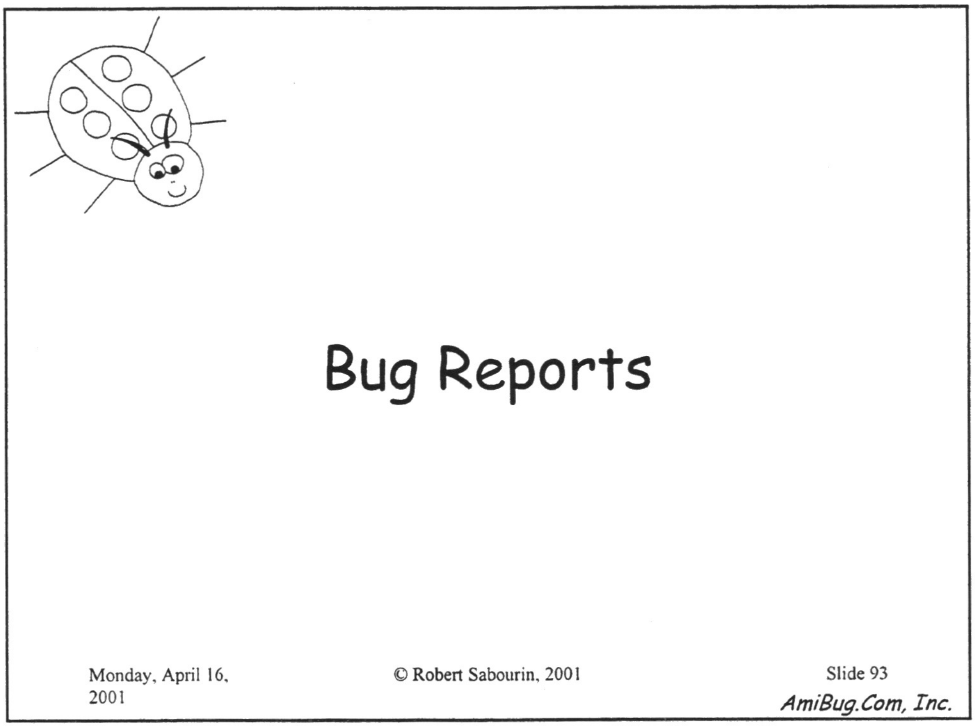


Bug Tracking Systems

- Never loose a bug
  - a bug tracking system is a database in which all bugs discovered or reported about an application are collected
  - typically a *"test lead"* is responsible for managing the *"bug list"* for an application

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 92

AmiBug.Com, Inc.



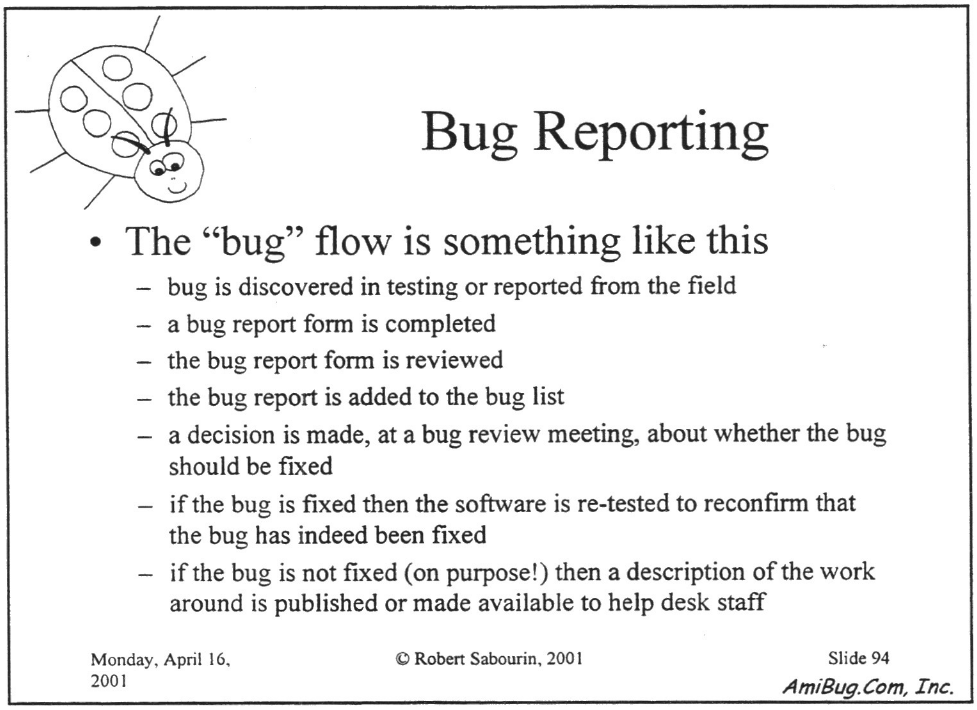
Bug Reports

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 93

AmiBug.Com, Inc.



Bug Reporting

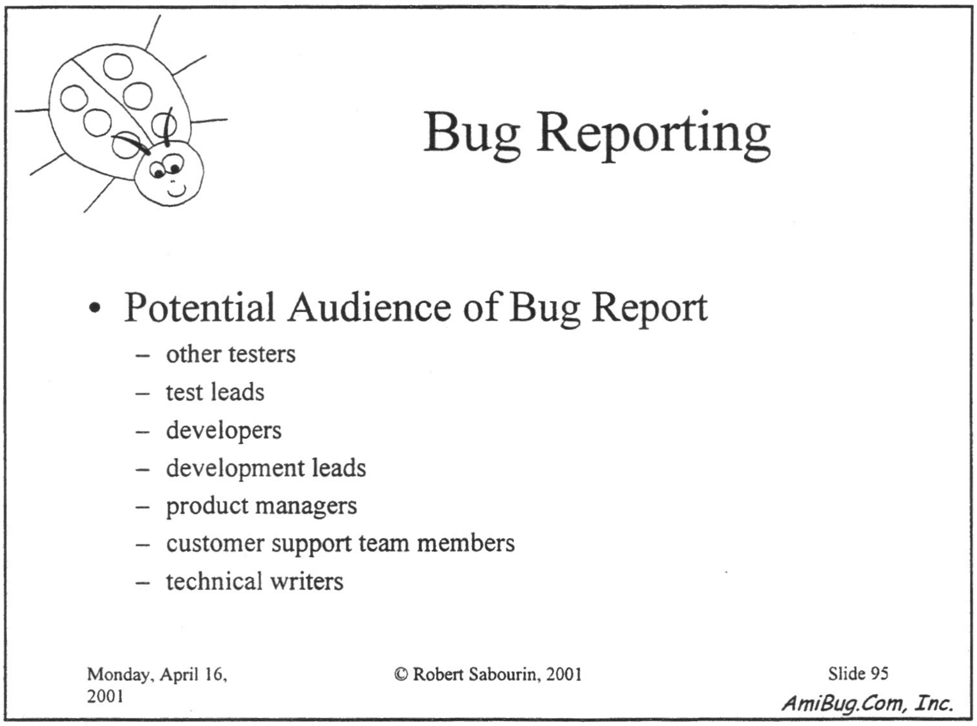
- The "bug" flow is something like this
  - bug is discovered in testing or reported from the field
  - a bug report form is completed
  - the bug report form is reviewed
  - the bug report is added to the bug list
  - a decision is made, at a bug review meeting, about whether the bug should be fixed
  - if the bug is fixed then the software is re-tested to reconfirm that the bug has indeed been fixed
  - if the bug is not fixed (on purpose!) then a description of the work around is published or made available to help desk staff

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 94

AmiBug.Com, Inc.



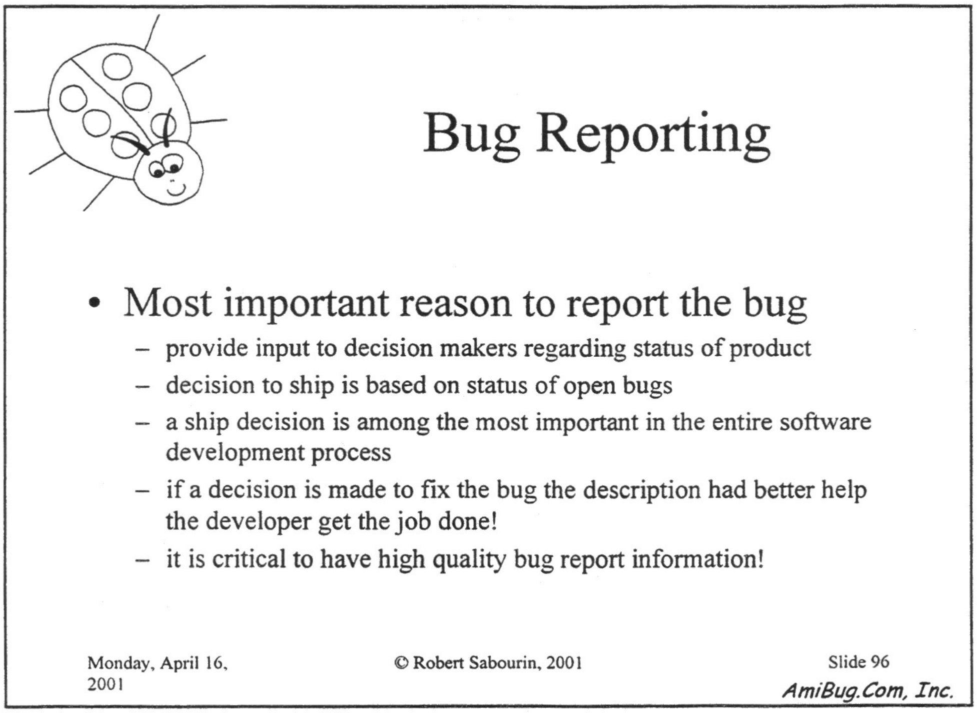
Bug Reporting

- Potential Audience of Bug Report
  - other testers
  - test leads
  - developers
  - development leads
  - product managers
  - customer support team members
  - technical writers

Monday, April 16, 2001

Slide 95

AmiBug.Com, Inc.



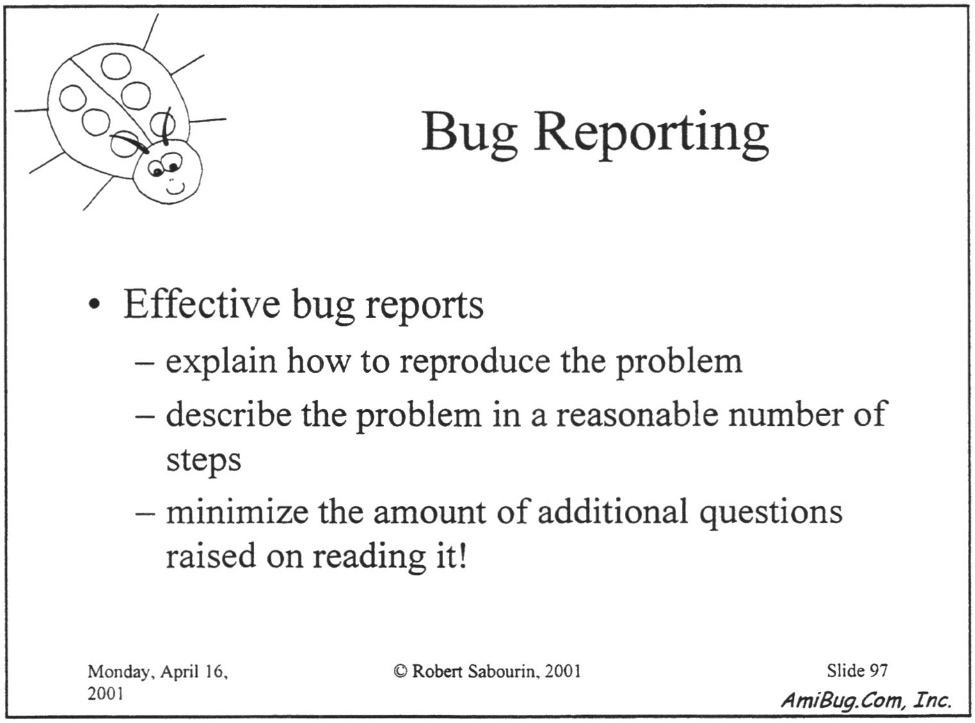
Bug Reporting

- Most important reason to report the bug
  - provide input to decision makers regarding status of product
  - decision to ship is based on status of open bugs
  - a ship decision is among the most important in the entire software development process
  - if a decision is made to fix the bug the description had better help the developer get the job done!
  - it is critical to have high quality bug report information!

Monday, April 16, 2001

Slide 96

AmiBug.Com, Inc.

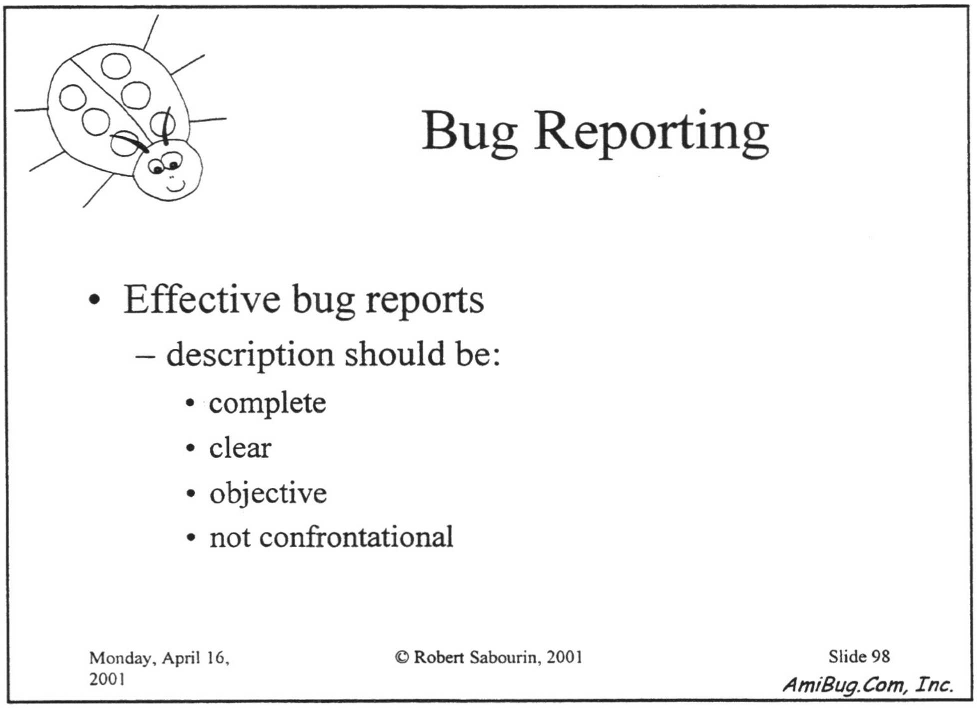


Bug Reporting

- Effective bug reports
  - explain how to reproduce the problem
  - describe the problem in a reasonable number of steps
  - minimize the amount of additional questions raised on reading it!

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 97

AmiBug.Com, Inc.

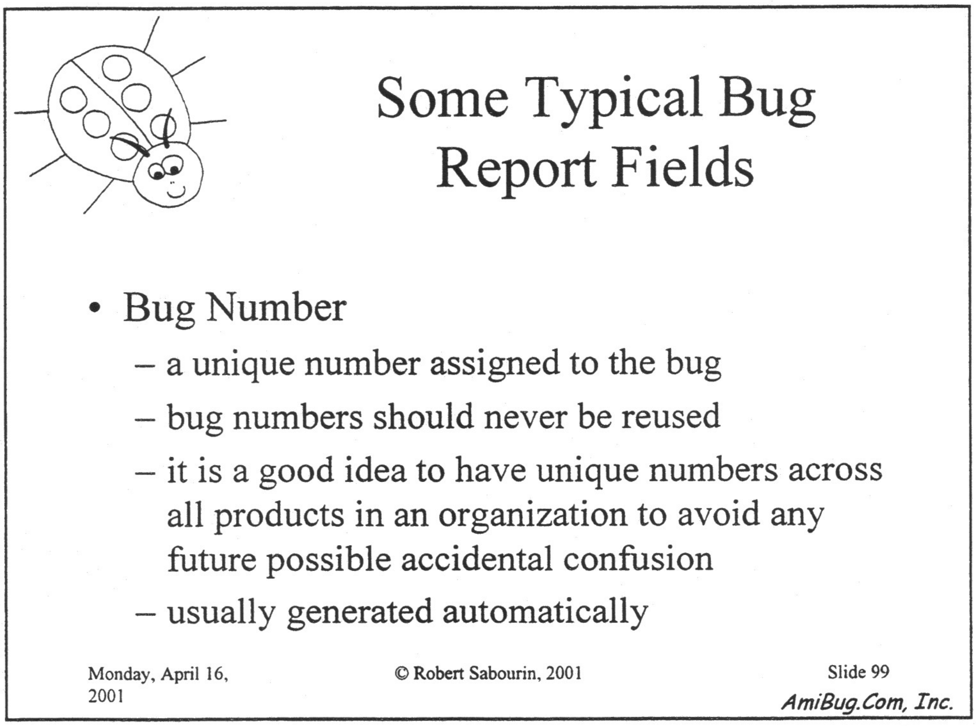


Bug Reporting

- Effective bug reports
  - description should be:
    - complete
    - clear
    - objective
    - not confrontational

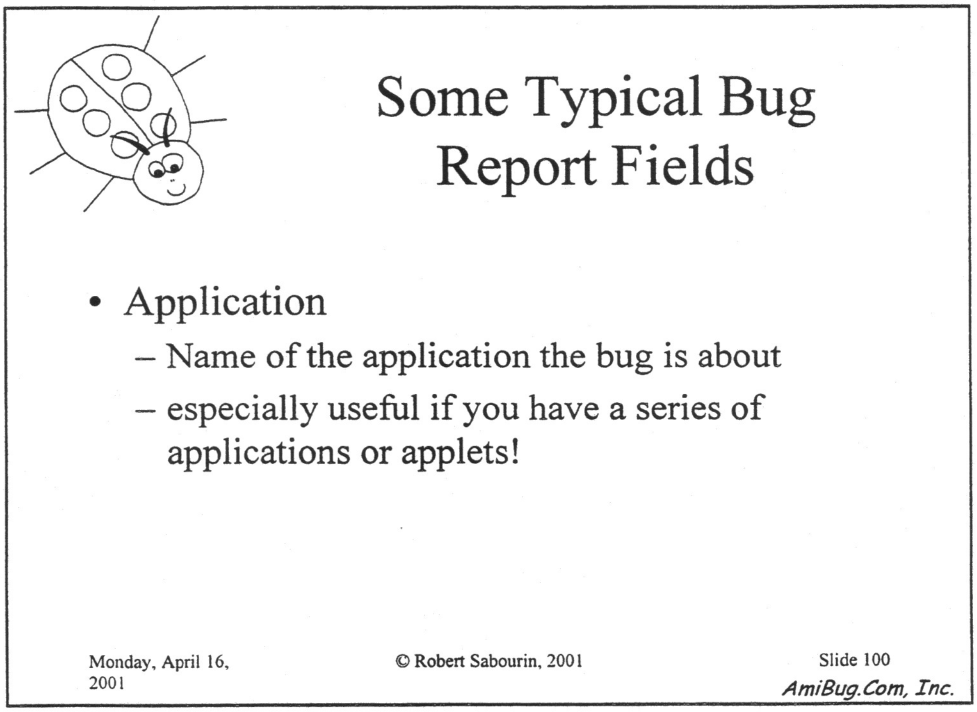
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 98

AmiBug.Com, Inc.



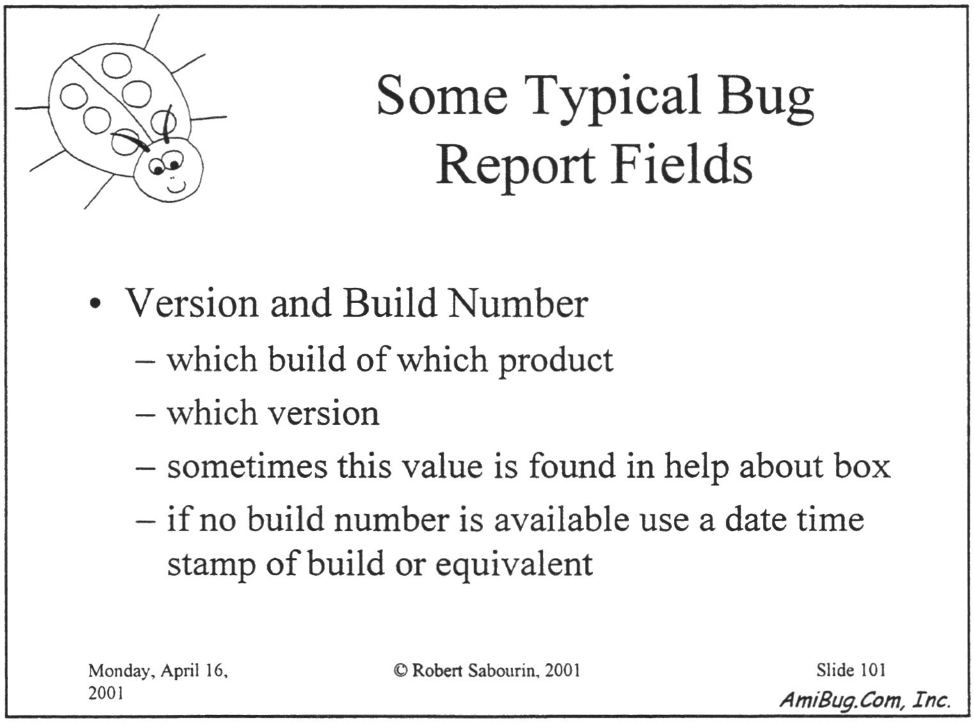
Some Typical Bug Report Fields

- Bug Number
  - a unique number assigned to the bug
  - bug numbers should never be reused
  - it is a good idea to have unique numbers across all products in an organization to avoid any future possible accidental confusion
  - usually generated automatically



Some Typical Bug Report Fields

- Application
  - Name of the application the bug is about
  - especially useful if you have a series of applications or applets!

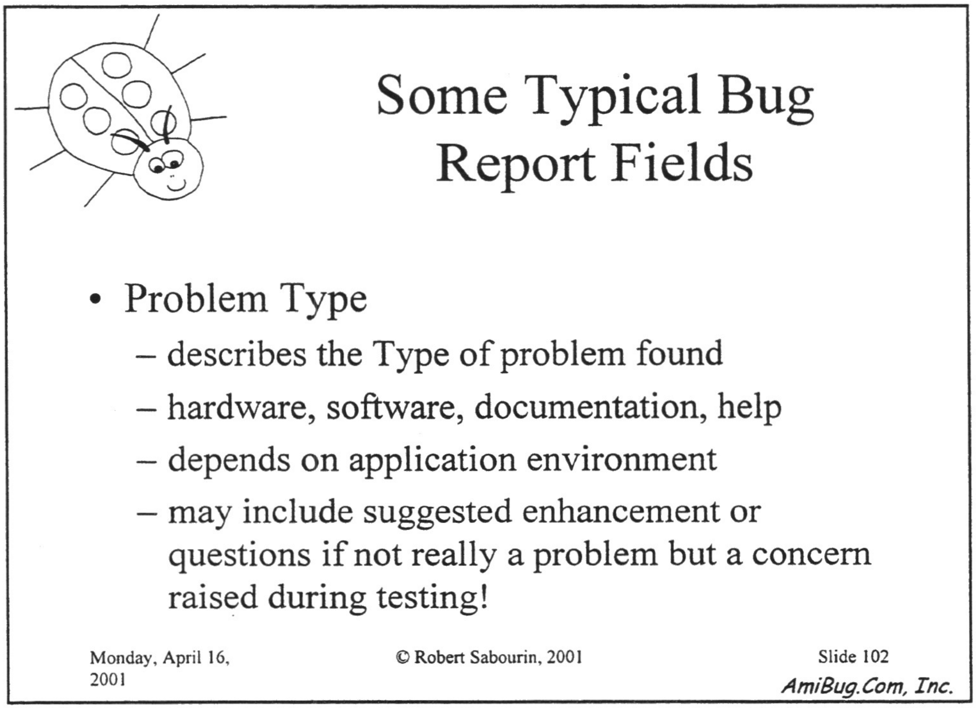


Some Typical Bug Report Fields

- Version and Build Number
  - which build of which product
  - which version
  - sometimes this value is found in help about box
  - if no build number is available use a date time stamp of build or equivalent

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 101

AmiBug.Com, Inc.

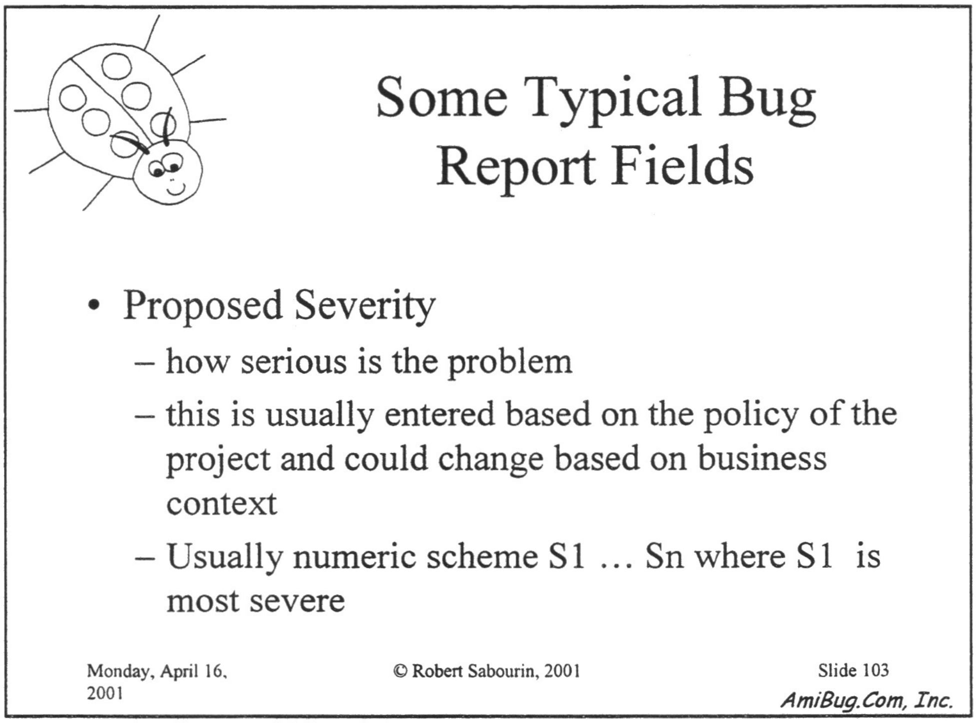


Some Typical Bug Report Fields

- Problem Type
  - describes the Type of problem found
  - hardware, software, documentation, help
  - depends on application environment
  - may include suggested enhancement or questions if not really a problem but a concern raised during testing!

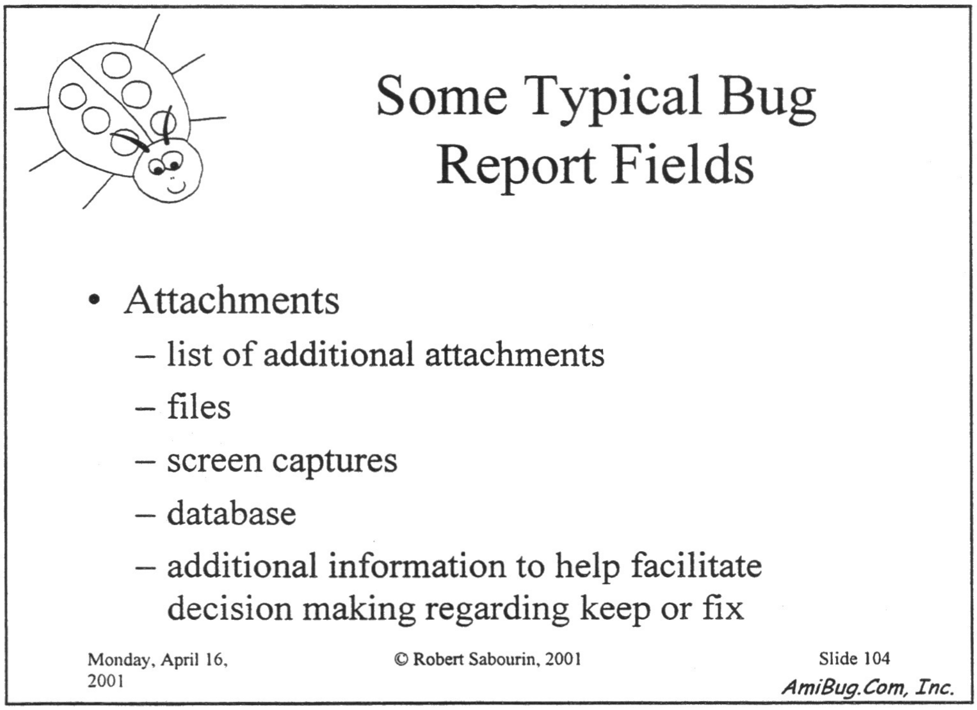
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 102

AmiBug.Com, Inc.



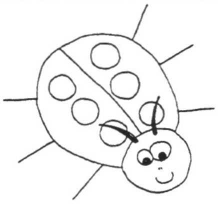
Some Typical Bug Report Fields

- Proposed Severity
  - how serious is the problem
  - this is usually entered based on the policy of the project and could change based on business context
  - Usually numeric scheme S1 … Sn where S1 is most severe



Some Typical Bug Report Fields

- Attachments
  - list of additional attachments
  - files
  - screen captures
  - database
  - additional information to help facilitate decision making regarding keep or fix

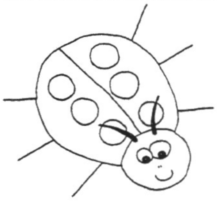


Some Typical Bug Report Fields

- Problem Summary
  - short clear description of the problem
  - usually used in executive summary of bug status
  - *"program crashes when saving using an invalid file name"* for example

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 105

AmiBug.Com, Inc.

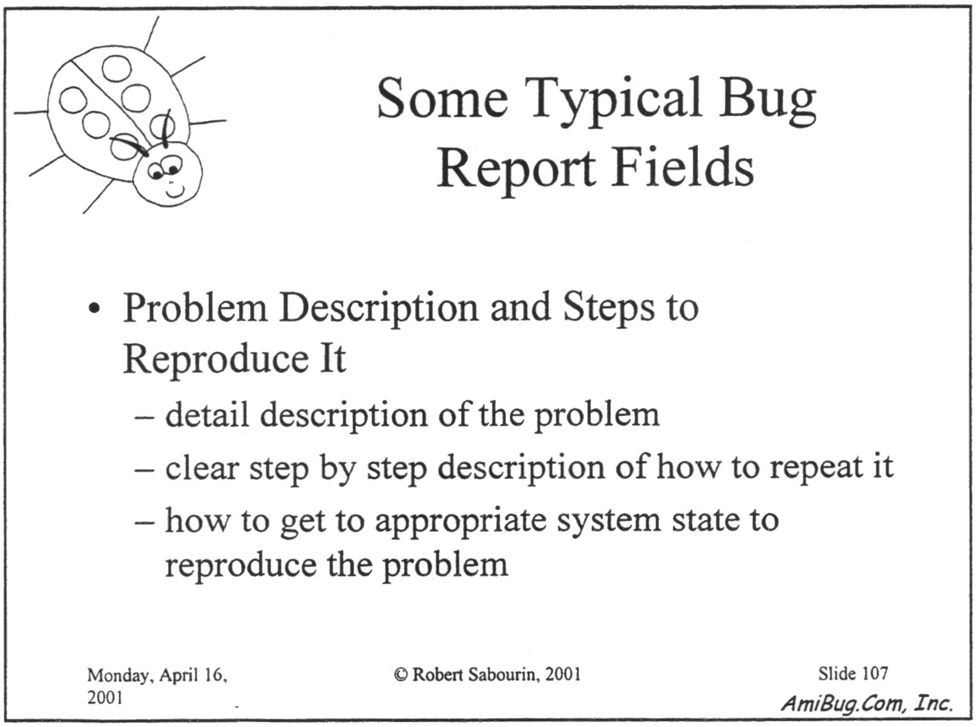


Some Typical Bug Report Fields

- Can Problem be reproduced
  - Yes, No or Sometimes
  - especially useful for the case of field reported problems
  - generally testing team should have a "yes" here!

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 106

AmiBug.Com, Inc.



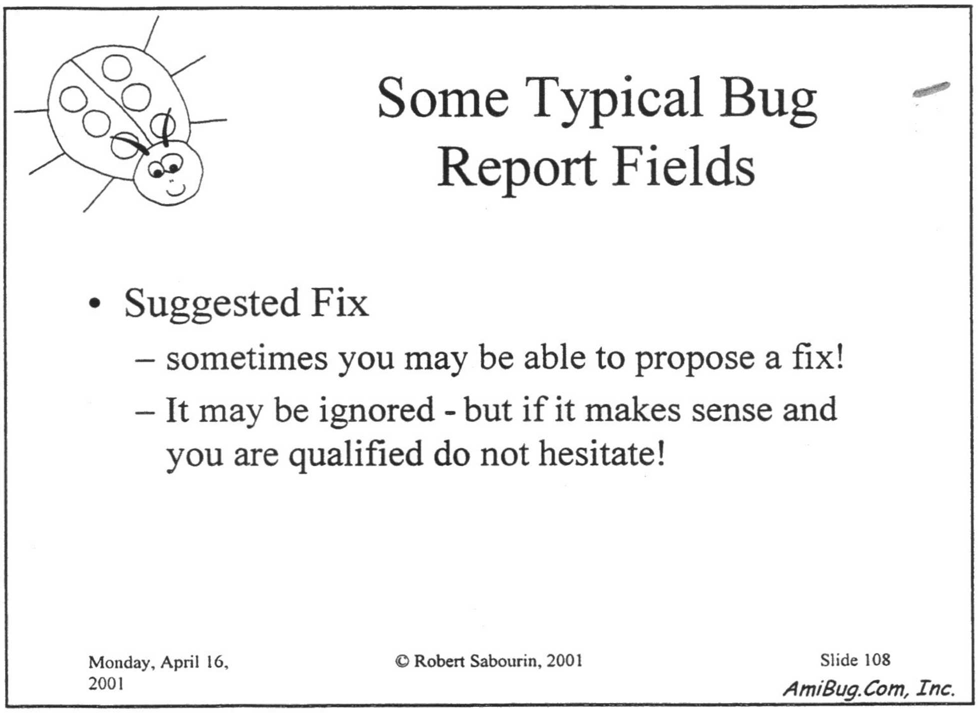
Some Typical Bug Report Fields

- Problem Description and Steps to Reproduce It
  - detail description of the problem
  - clear step by step description of how to repeat it
  - how to get to appropriate system state to reproduce the problem

Monday, April 16, 2001

Slide 107

AmiBug.Com, Inc.



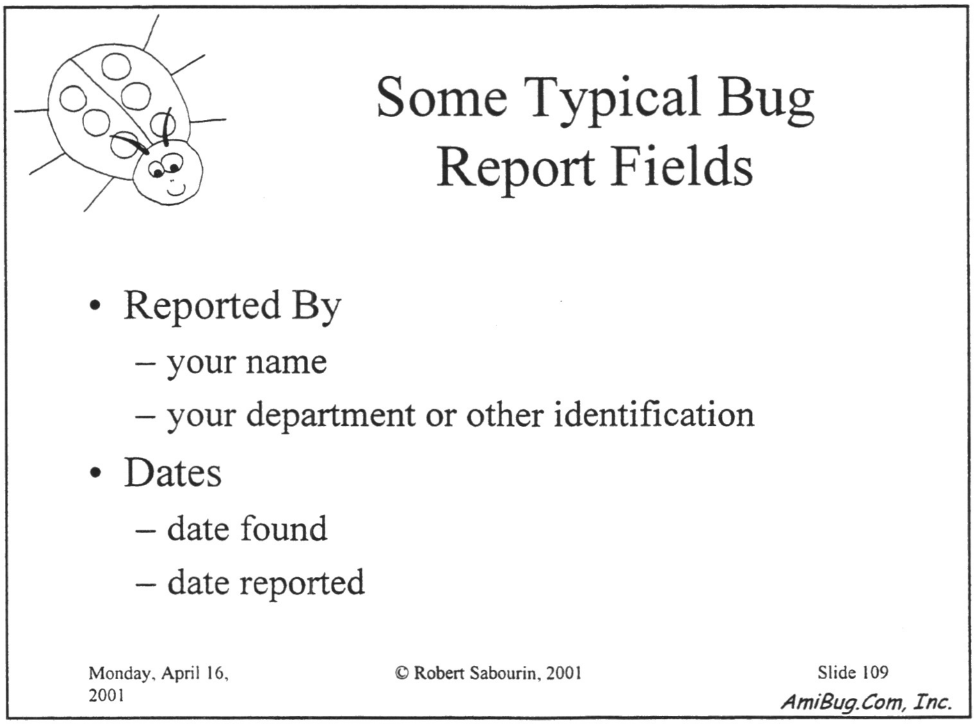
Some Typical Bug Report Fields

- Suggested Fix
  - sometimes you may be able to propose a fix!
  - It may be ignored - but if it makes sense and you are qualified do not hesitate!

Monday, April 16, 2001

Slide 108

AmiBug.Com, Inc.

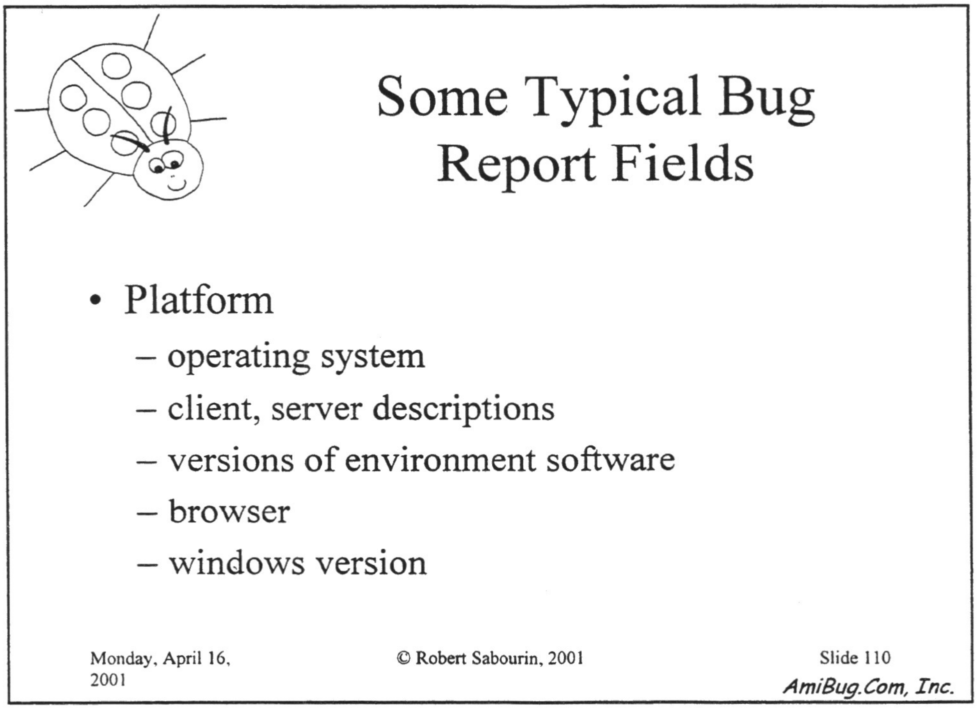


Some Typical Bug Report Fields

- Reported By
  - your name
  - your department or other identification
- Dates
  - date found
  - date reported

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 109

AmiBug.Com, Inc.

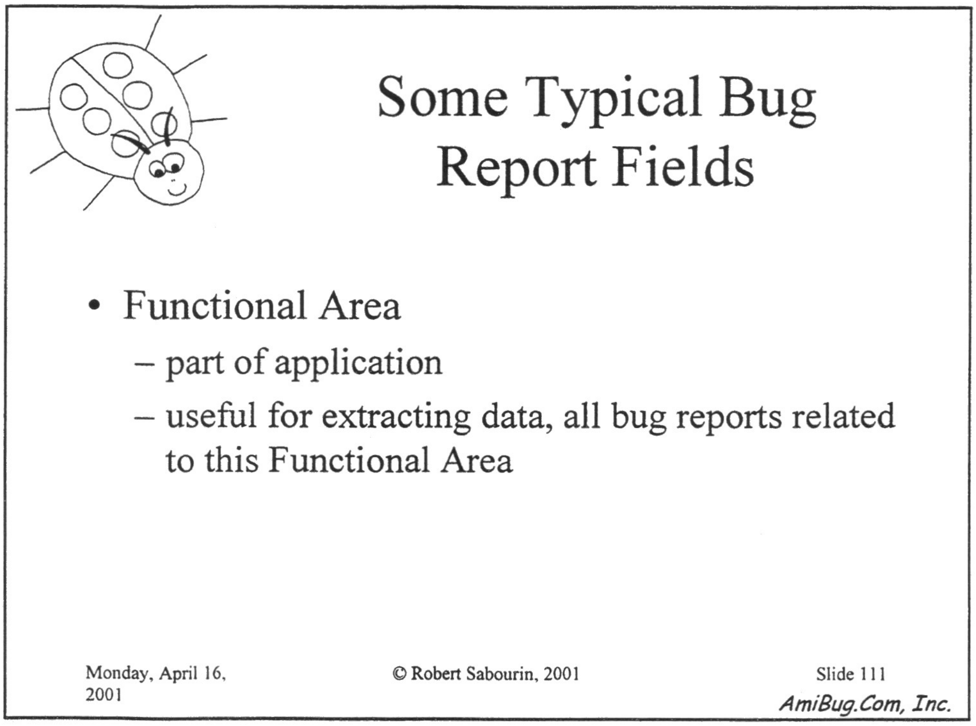


Some Typical Bug Report Fields

- Platform
  - operating system
  - client, server descriptions
  - versions of environment software
  - browser
  - windows version

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 110

AmiBug.Com, Inc.



Some Typical Bug Report Fields

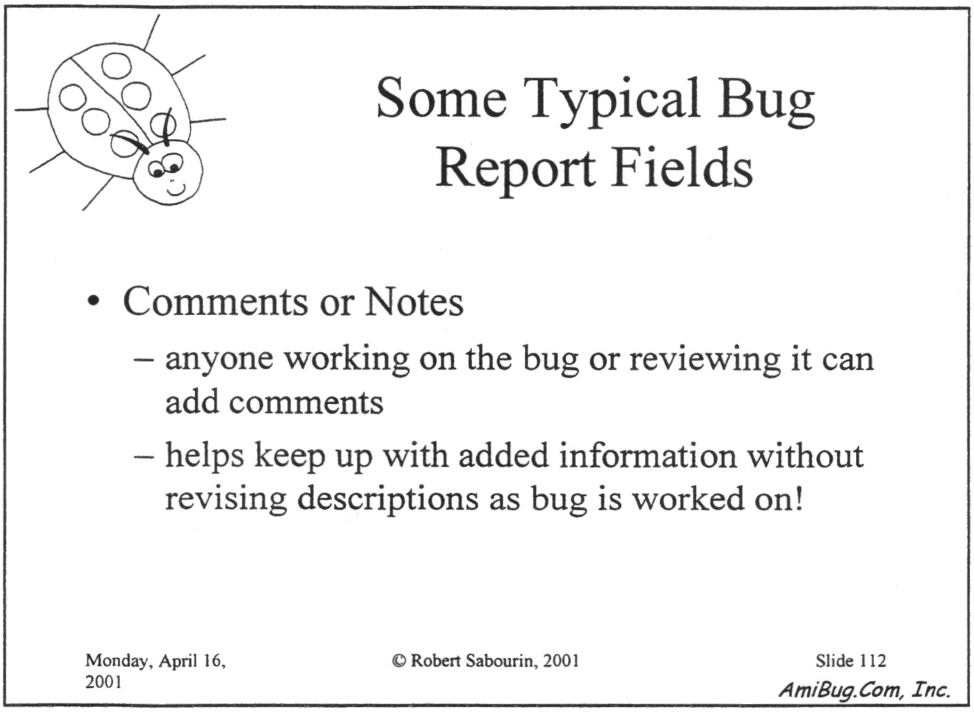
- Functional Area
  - part of application
  - useful for extracting data, all bug reports related to this Functional Area

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 111

AmiBug.Com, Inc.



Some Typical Bug Report Fields

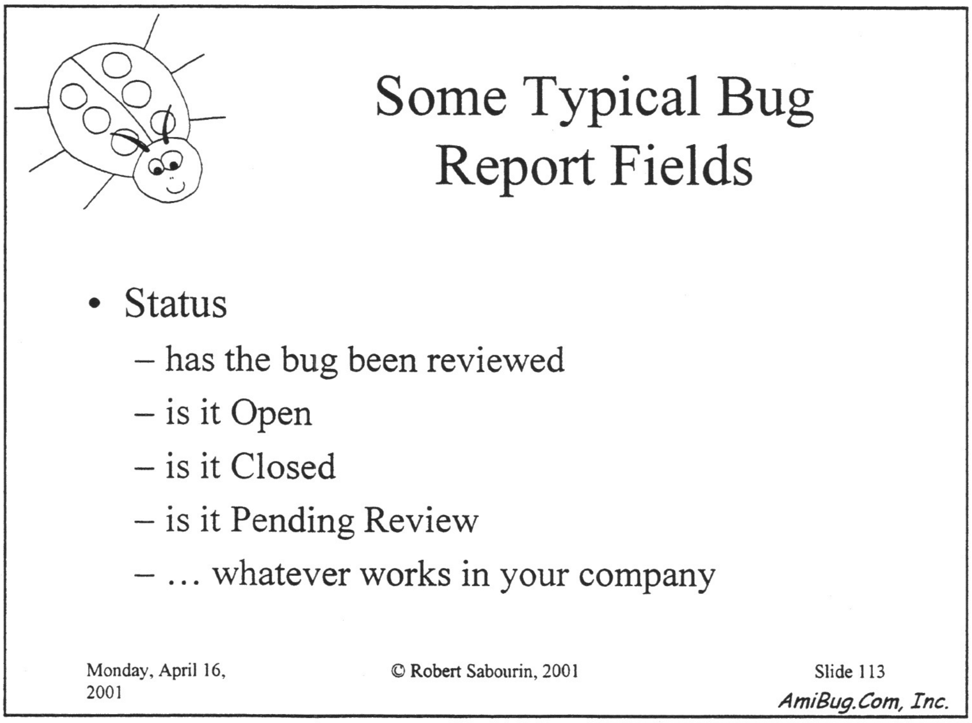
- Comments or Notes
  - anyone working on the bug or reviewing it can add comments
  - helps keep up with added information without revising descriptions as bug is worked on!

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 112

AmiBug.Com, Inc.

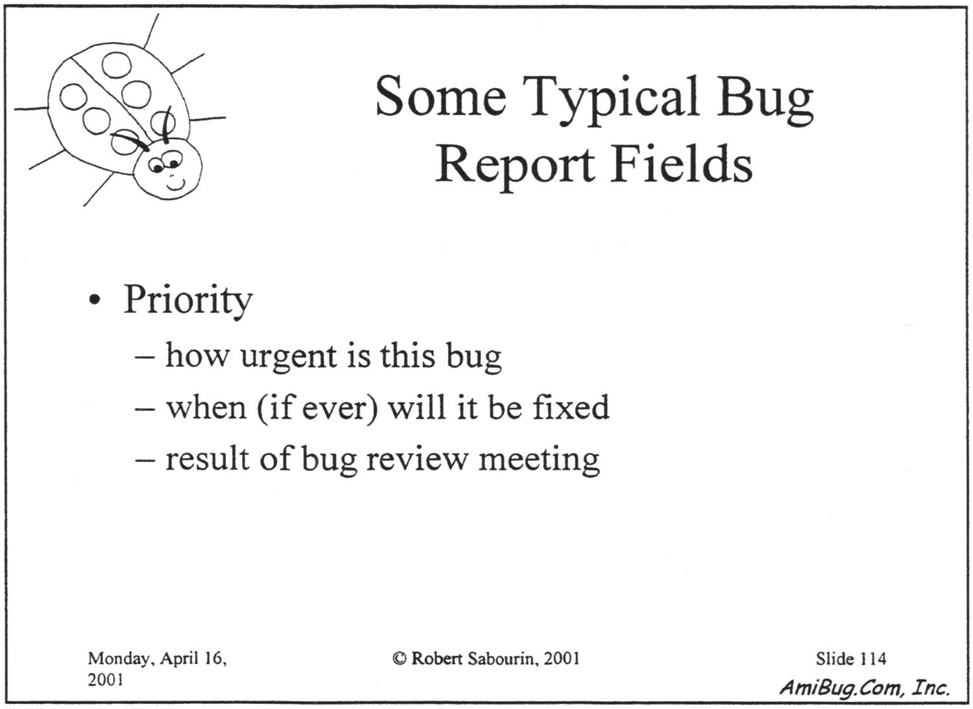


Some Typical Bug Report Fields

- Status
  - – has the bug been reviewed
  - – is it Open
  - – is it Closed
  - – is it Pending Review
  - – … whatever works in your company

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 113

AmiBug.Com, Inc.

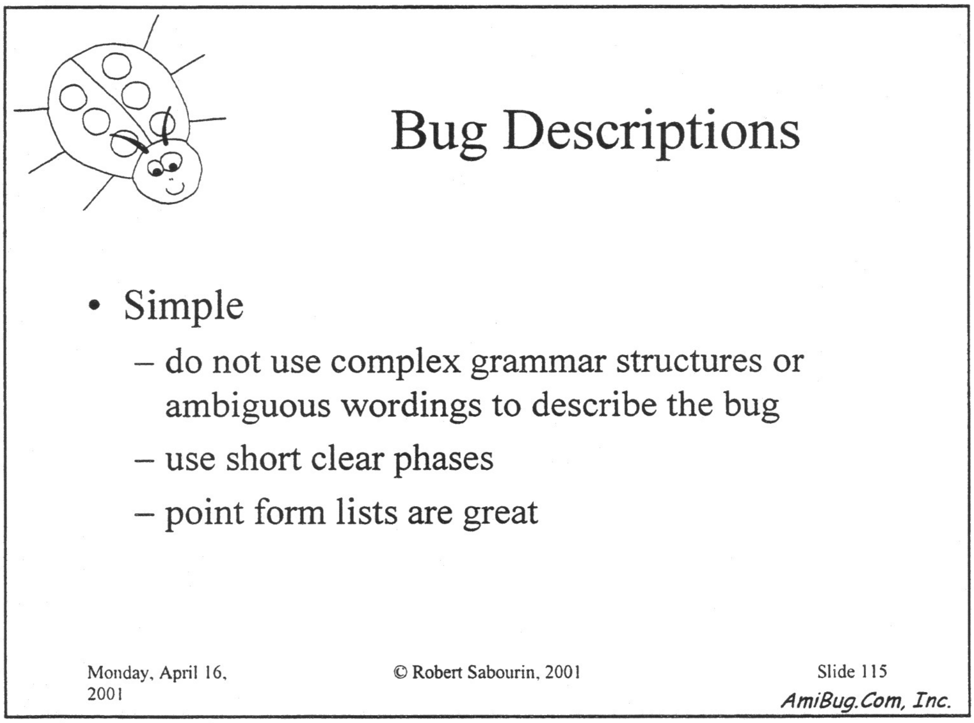


Some Typical Bug Report Fields

- Priority
  - – how urgent is this bug
  - – when (if ever) will it be fixed
  - – result of bug review meeting

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 114

AmiBug.Com, Inc.



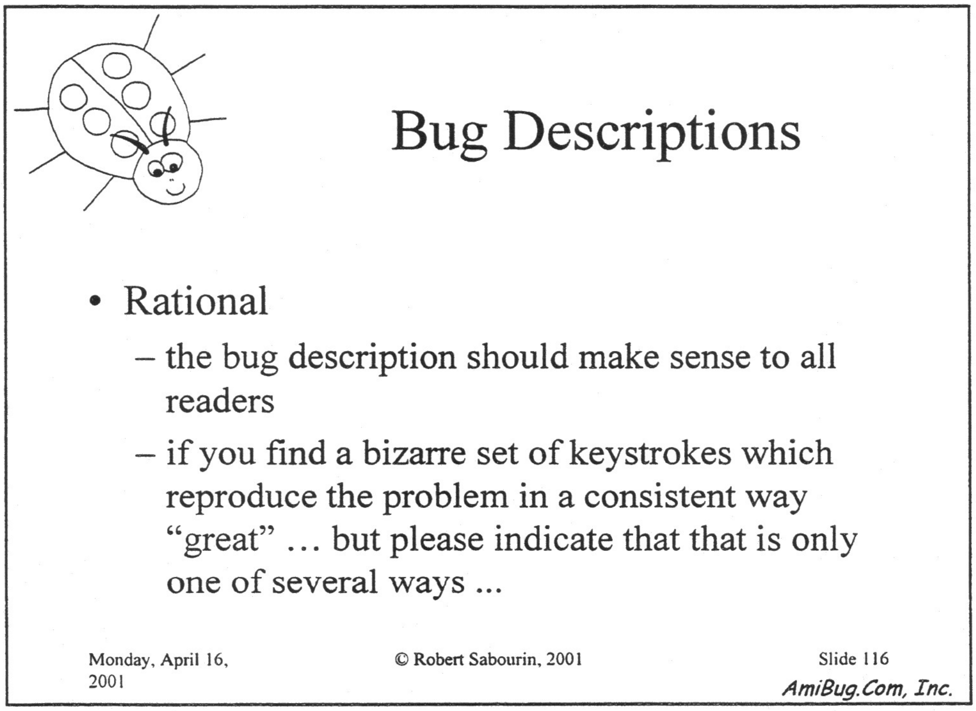
Bug Descriptions

- Simple
  - do not use complex grammar structures or ambiguous wordings to describe the bug
  - use short clear phases
  - point form lists are great

Monday, April 16, 2001

Slide 115

AmiBug.Com, Inc.



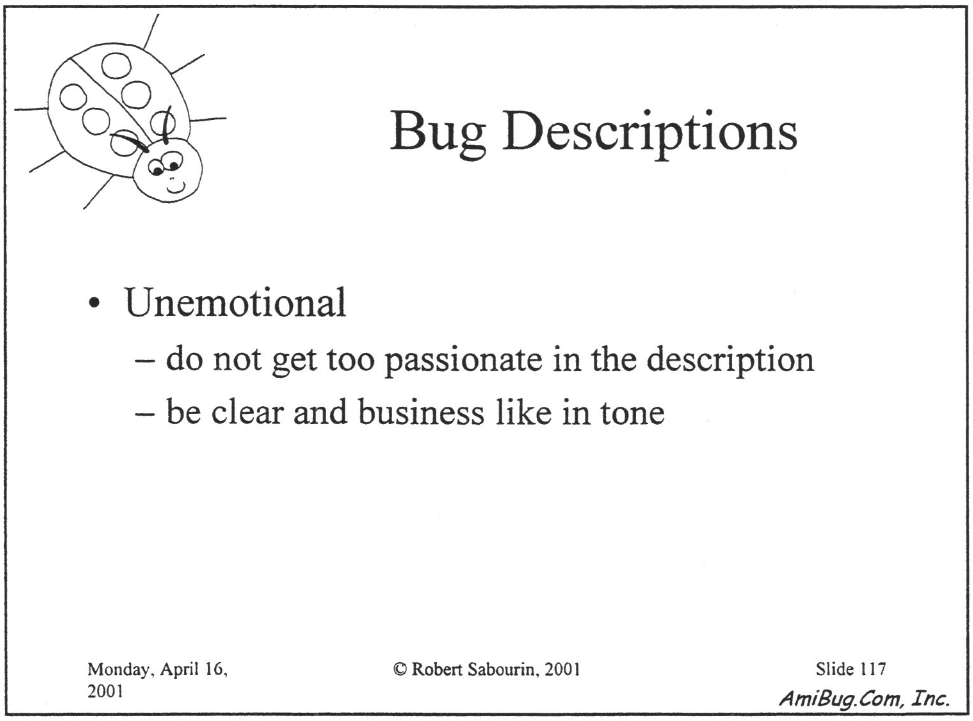
Bug Descriptions

- Rational
  - the bug description should make sense to all readers
  - if you find a bizarre set of keystrokes which reproduce the problem in a consistent way "great" … but please indicate that that is only one of several ways ...

Monday, April 16, 2001

Slide 116

AmiBug.Com, Inc.

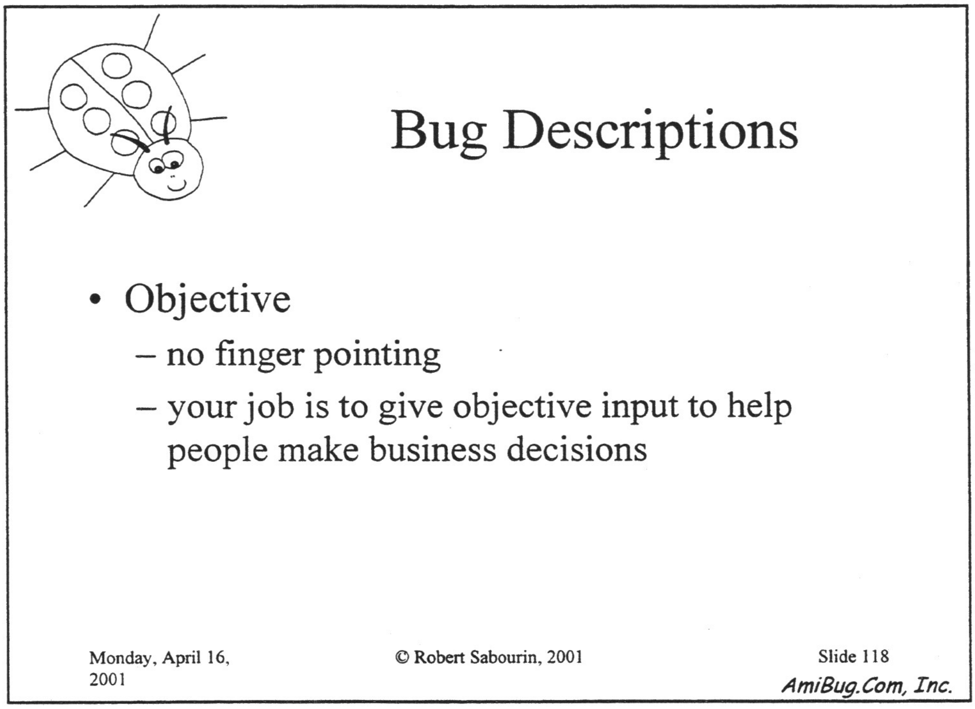


Bug Descriptions

- Unemotional
  - do not get too passionate in the description
  - be clear and business like in tone

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 117

AmiBug.Com, Inc.

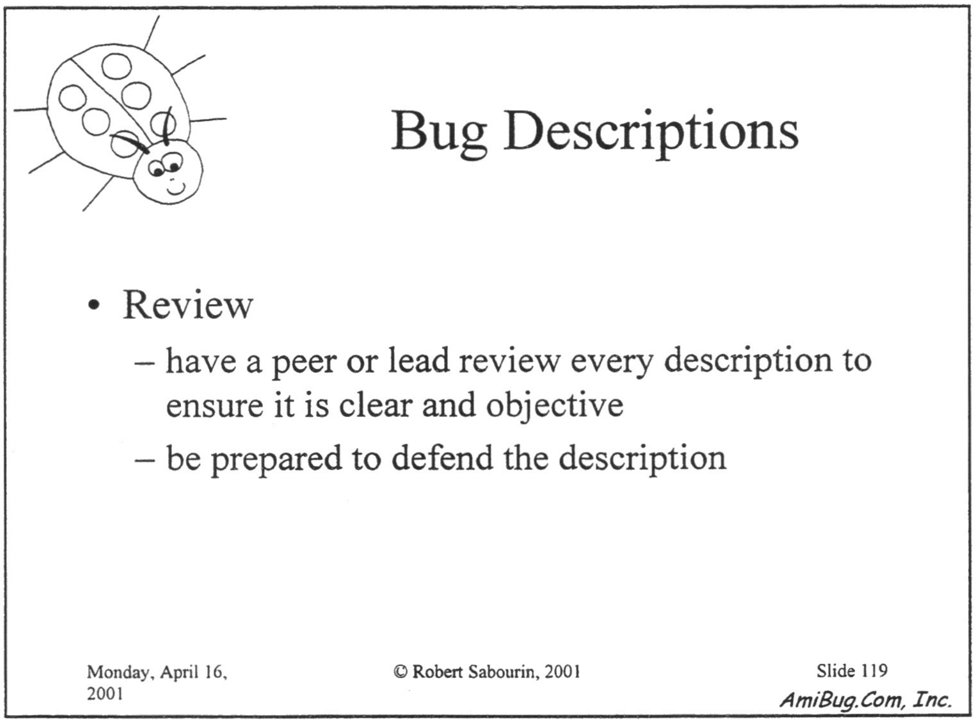


Bug Descriptions

- Objective
  - no finger pointing
  - your job is to give objective input to help people make business decisions

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 118

AmiBug.Com, Inc.



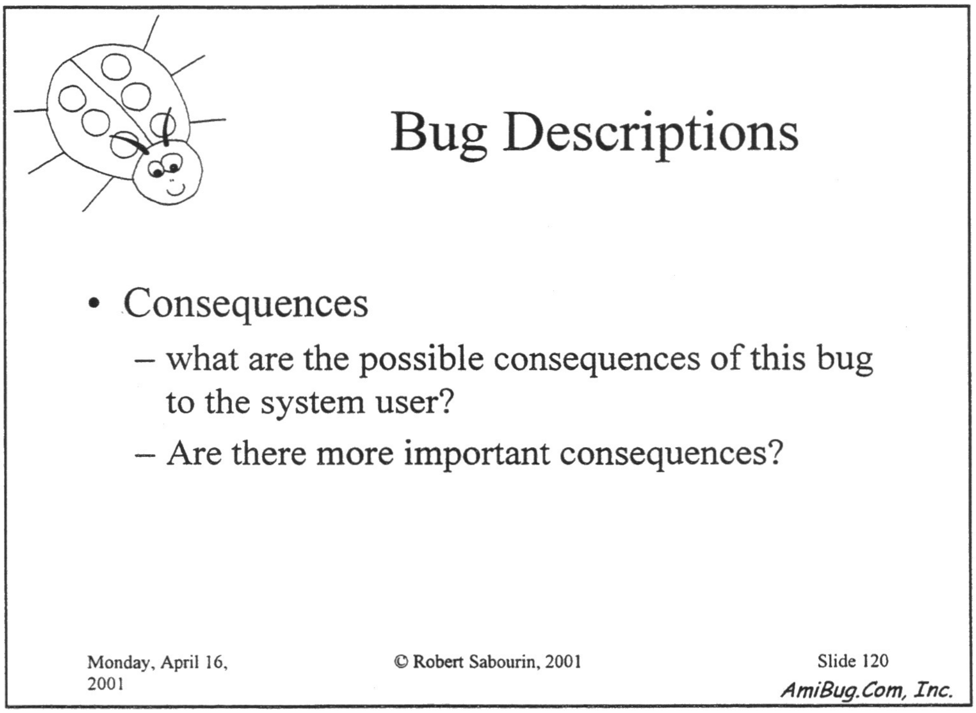
Bug Descriptions

- Review
  - have a peer or lead review every description to ensure it is clear and objective
  - be prepared to defend the description

Monday, April 16, 2001

Slide 119

AmiBug.Com, Inc.



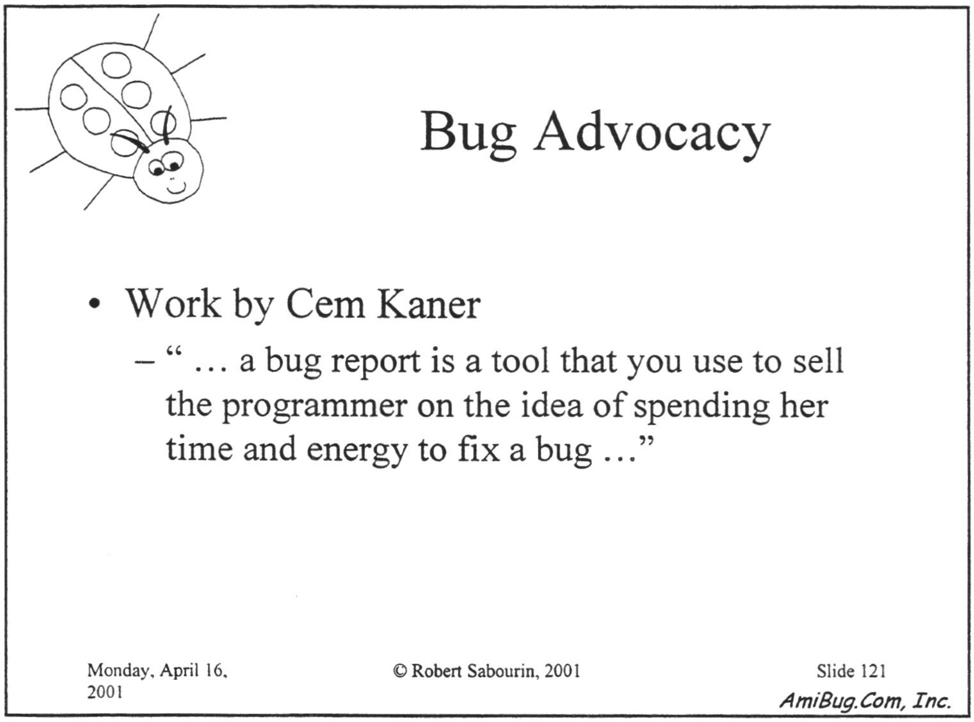
Bug Descriptions

- Consequences
  - what are the possible consequences of this bug to the system user?
  - Are there more important consequences?

Monday, April 16, 2001

Slide 120

AmiBug.Com, Inc.

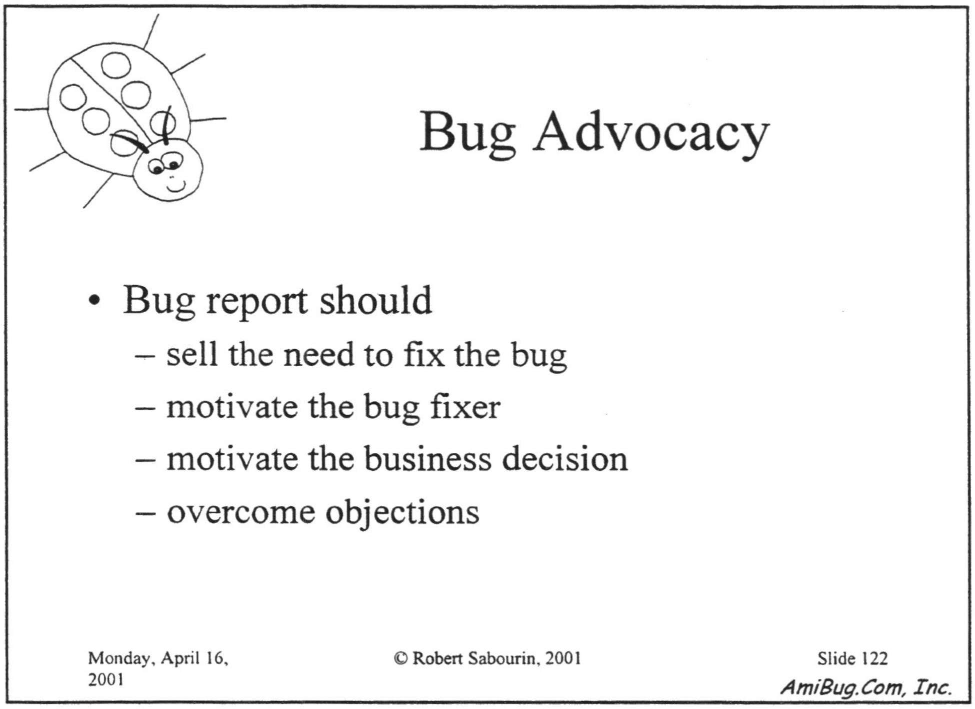


Bug Advocacy

- Work by Cem Kaner
  - " … a bug report is a tool that you use to sell the programmer on the idea of spending her time and energy to fix a bug …"

Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 121

AmiBug.Com, Inc.

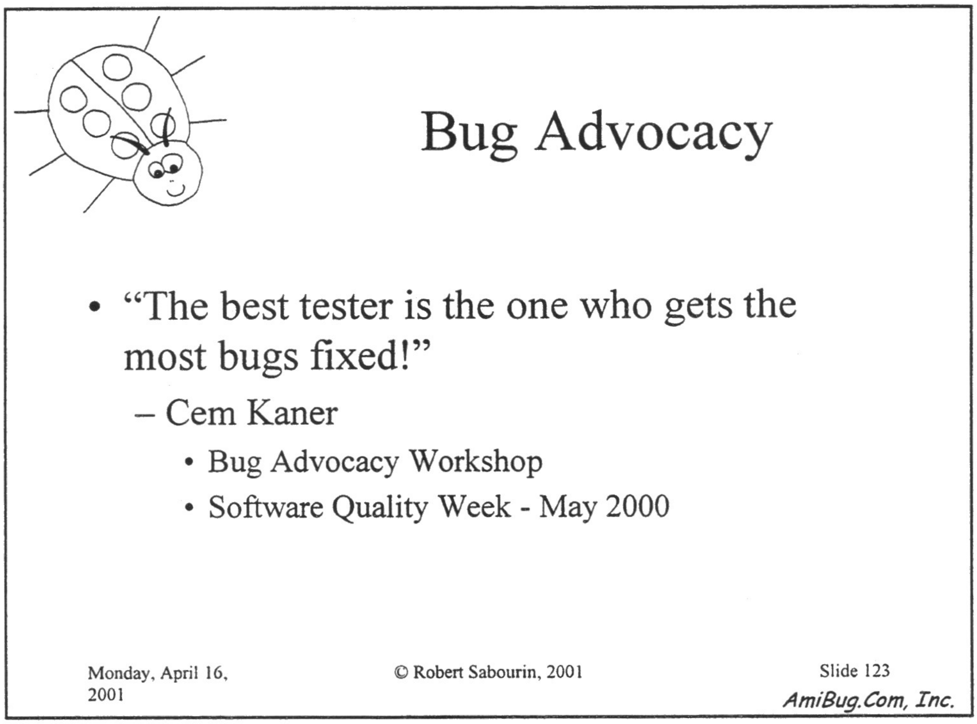


Bug Advocacy

- Bug report should
  - sell the need to fix the bug
  - motivate the bug fixer
  - motivate the business decision
  - overcome objections

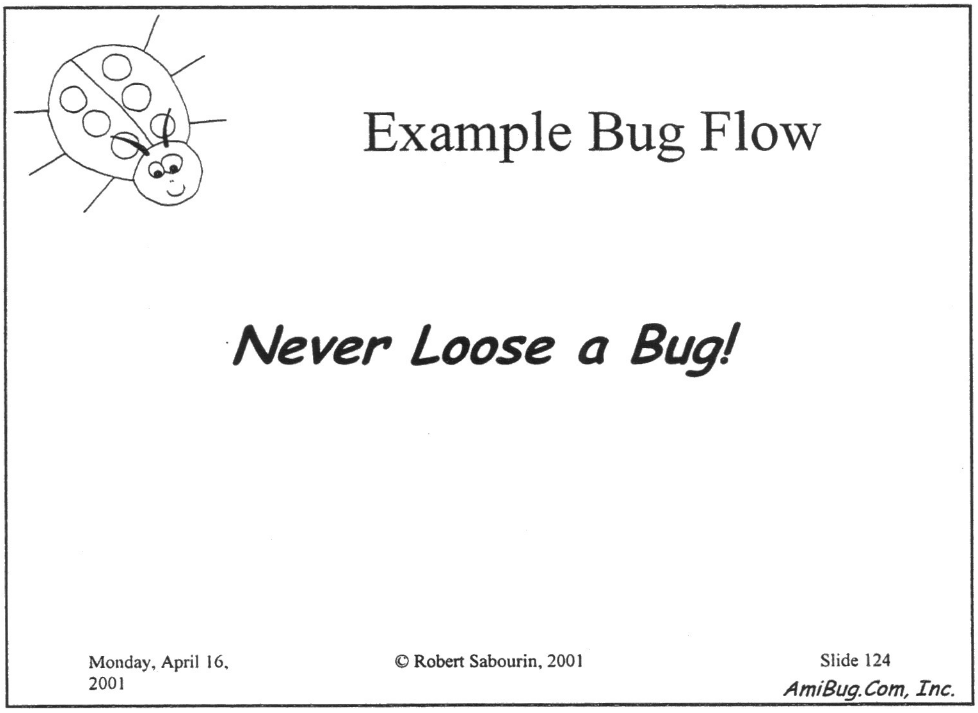
Monday, April 16, 2001 © Robert Sabourin, 2001 Slide 122

AmiBug.Com, Inc.



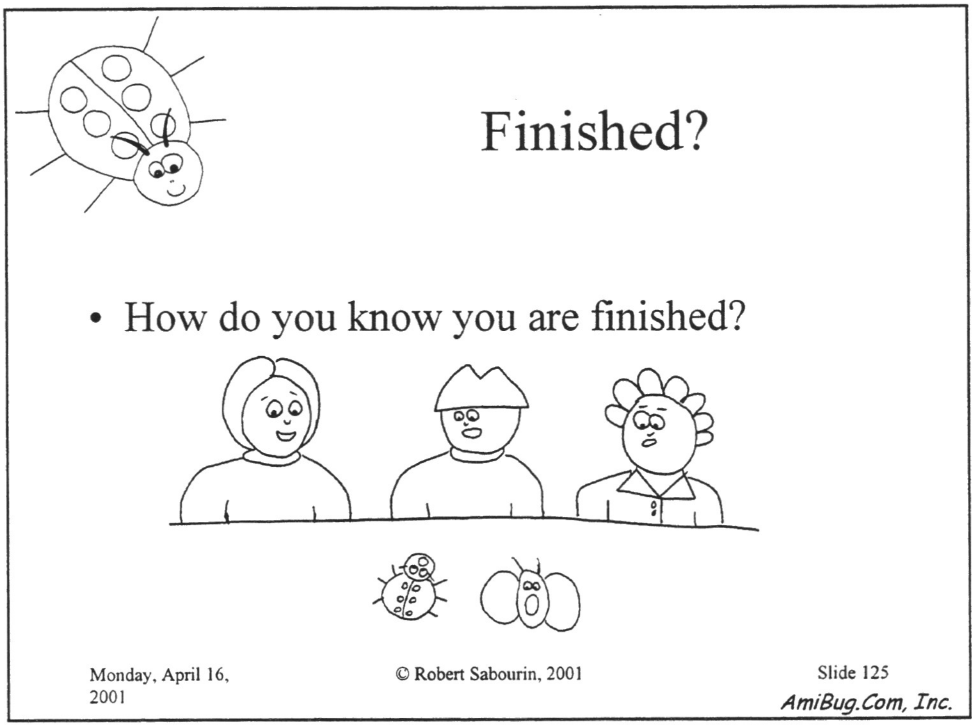
Bug Advocacy

- “The best tester is the one who gets the most bugs fixed!”
  - – Cem Kaner
    - Bug Advocacy Workshop
    - Software Quality Week - May 2000



Example Bug Flow

Never Loose a Bug!



Finished?

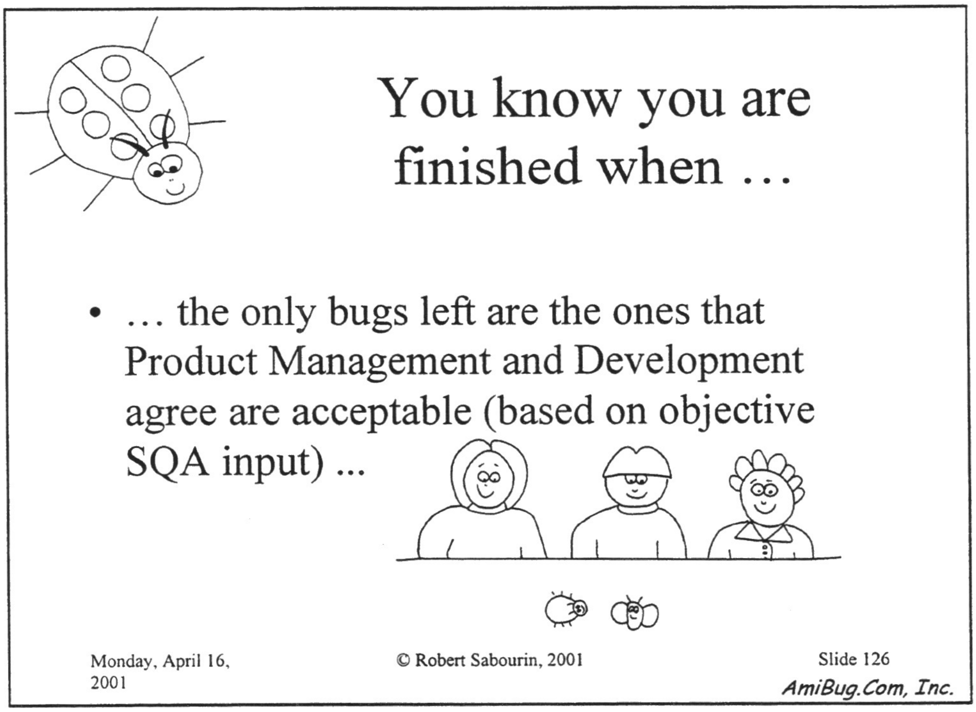
- How do you know you are finished?

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 125

AmiBug.Com, Inc.



You know you are finished when …

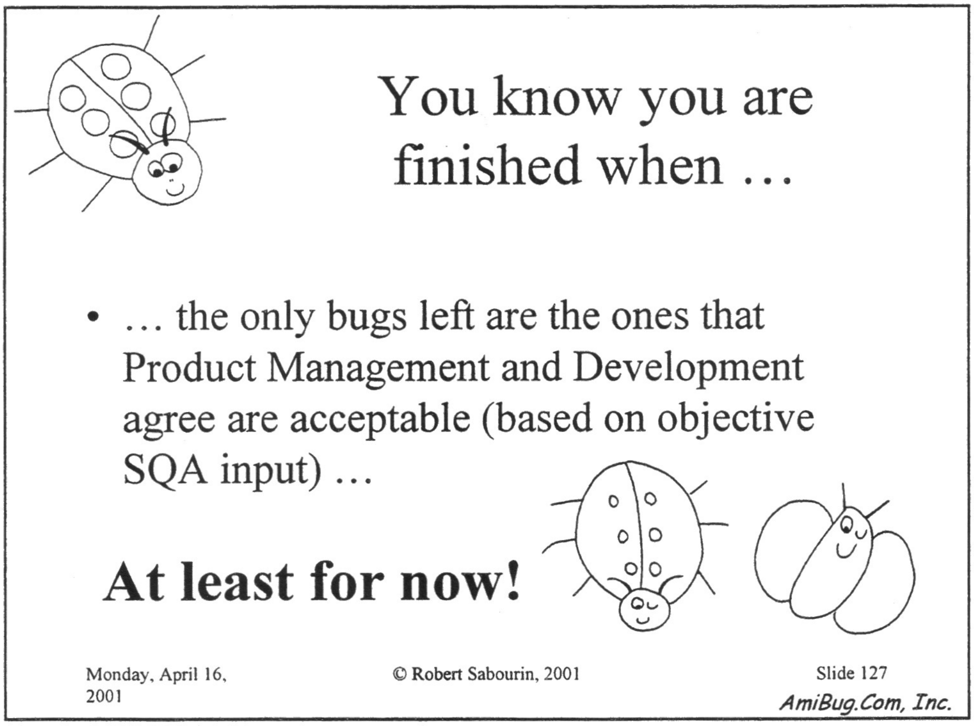
- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) ...

Monday, April 16, 2001

© Robert Sabourin, 2001

Slide 126

AmiBug.Com, Inc.



You know you are finished when …

- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) …

At least for now!

Monday, April 16, 2001

Slide 127

AmiBug.Com, Inc.

QW2001 Workshop W4

Ms. Johanna Rothman & Ms. Elizabeth Hendrickson
(The Rothman Consulting Group)

Grace Under Pressure: Handling Sticky Situations in Testing

Key Points

- How to make sure they heard what you said and not what they wanted you to say.
- How to handle difficult situations and obstinate people.
- What to do when you don't feel like anyone is listening.
- How to say "no" and make it stick.

Presentation Abstract

In this workshop Elisabeth Hendrickson and Johanna Rothman examine a series of difficult interactions between testers, test leads, developers, and managers, demonstrating proven techniques for presenting bad news, saying "no," and influencing others' behavior when you have no authority over them.

About the Author

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

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

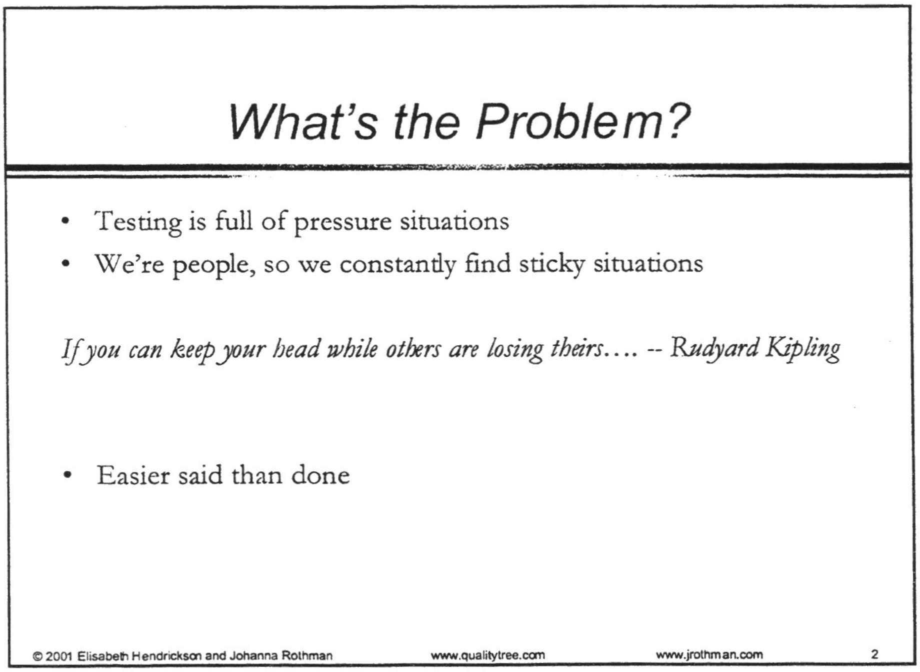
Johanna is the founder and principal of Rothman Consulting Group, Inc., and is a member of the clinical faculty of The Gordon Institute at Tufts University, a practical management degree program for engineers.

Elisabeth Hendrickson is the Director of Quality Engineering at Aveo Inc., an Application Service Provider. Aveo Inc. offers Attune, a pre-emptive technical support service whose mission is to help companies communicate the right information to the right customer at the right time. Prior to joining Aveo, Elisabeth was the founder of Quality Tree Consulting, a software quality assurance consulting firm. As a consultant, Elisabeth provided services to Application Service Providers as well as more traditional independent software vendors.

Grace Under Pressure: Handling Sticky Situations in Testing

Elisabeth Hendrickson
Quality Tree Software, Inc.
925-426-9726
esh@qualitytree.com
www.qualitytree.com

Johanna Rothman
Rothman Consulting Group, Inc.
781-641-4046
jr@jrothman.com
www.jrothman.com

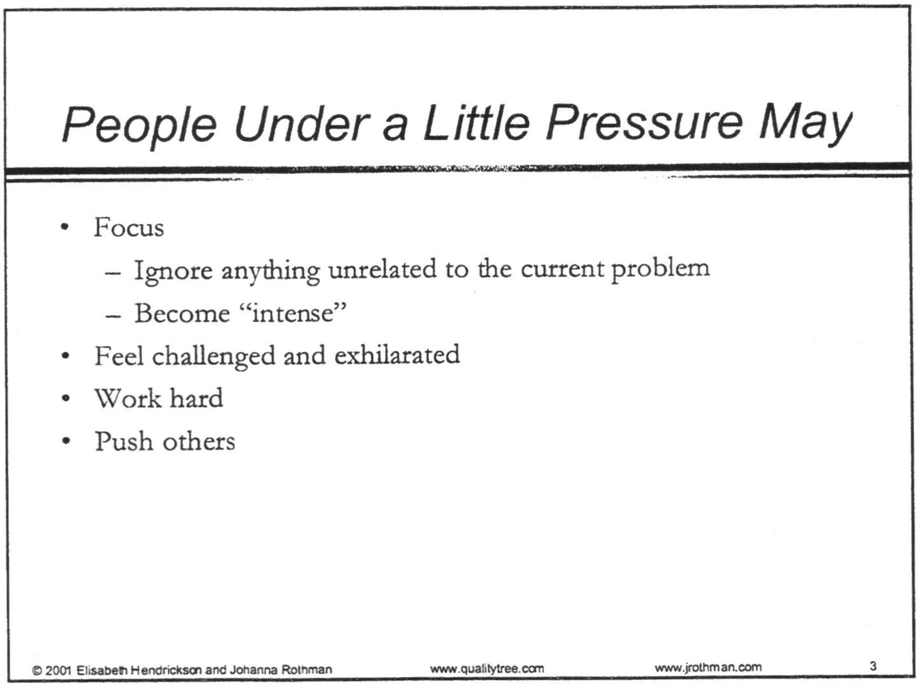


What's the Problem?

- Testing is full of pressure situations
- We're people, so we constantly find sticky situations

If you can keep your head while others are losing theirs…. -- *Rudyard Kipling*

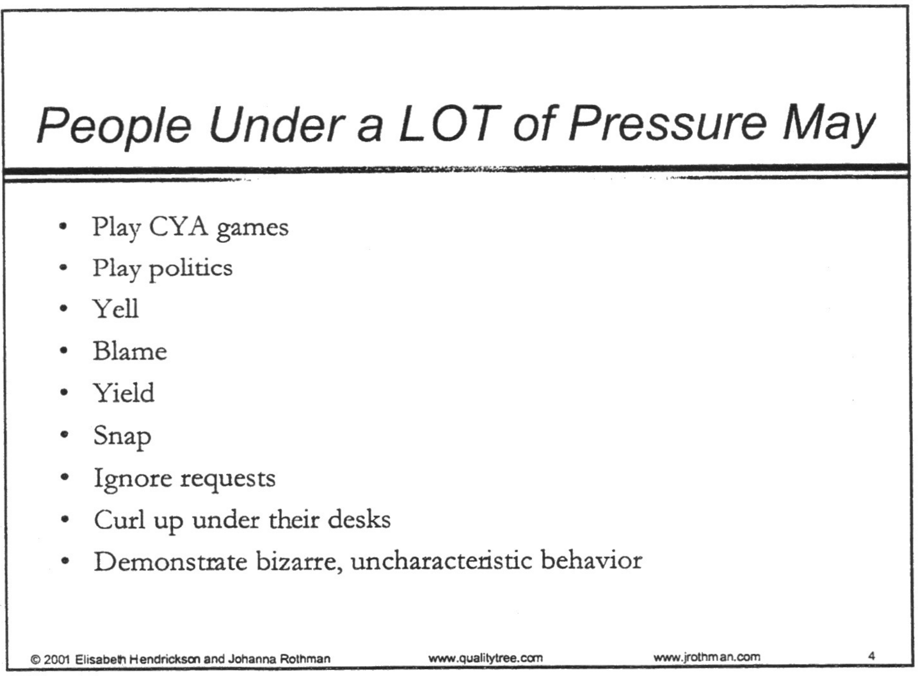
- Easier said than done



People Under a Little Pressure May

- Focus
  - Ignore anything unrelated to the current problem
  - Become “intense”
- Feel challenged and exhilarated
- Work hard
- Push others

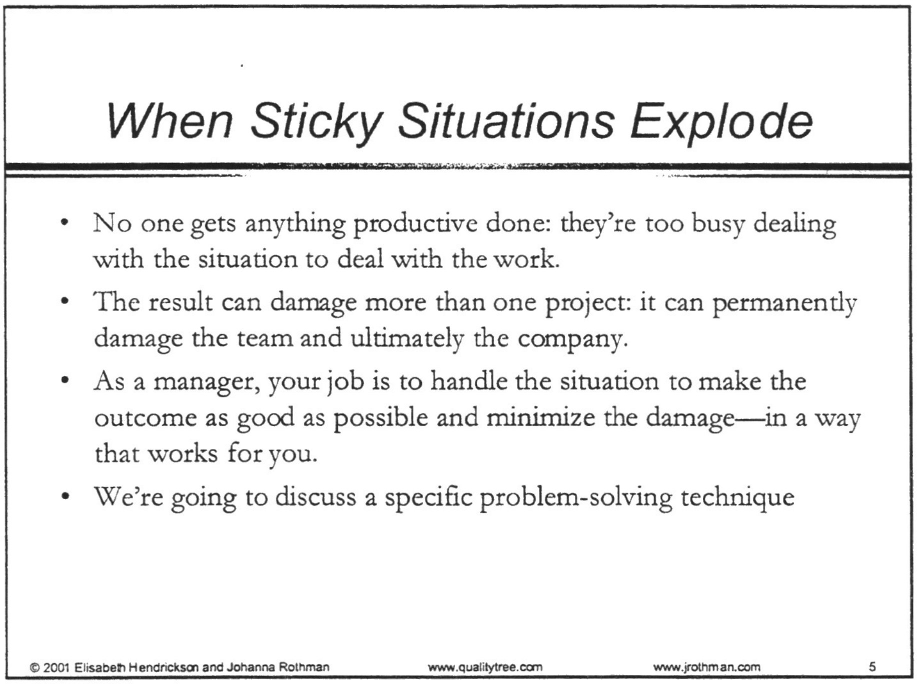
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 3



People Under a LOT of Pressure May

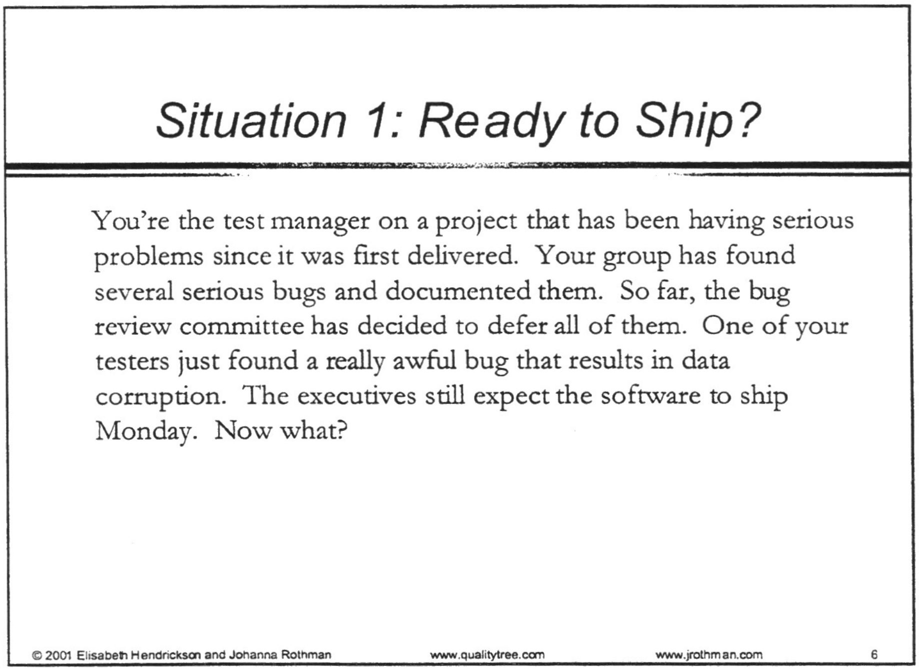
- Play CYA games
- Play politics
- Yell
- Blame
- Yield
- Snap
- Ignore requests
- Curl up under their desks
- Demonstrate bizarre, uncharacteristic behavior

© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 4



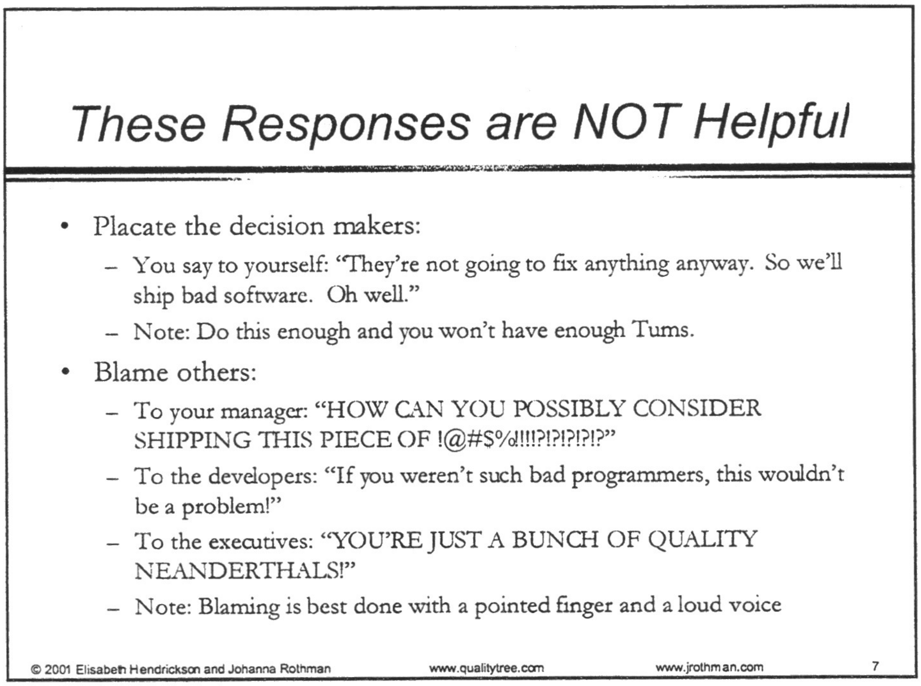
When Sticky Situations Explode

- No one gets anything productive done: they're too busy dealing with the situation to deal with the work.
- The result can damage more than one project: it can permanently damage the team and ultimately the company.
- As a manager, your job is to handle the situation to make the outcome as good as possible and minimize the damage—in a way that works for you.
- We're going to discuss a specific problem-solving technique



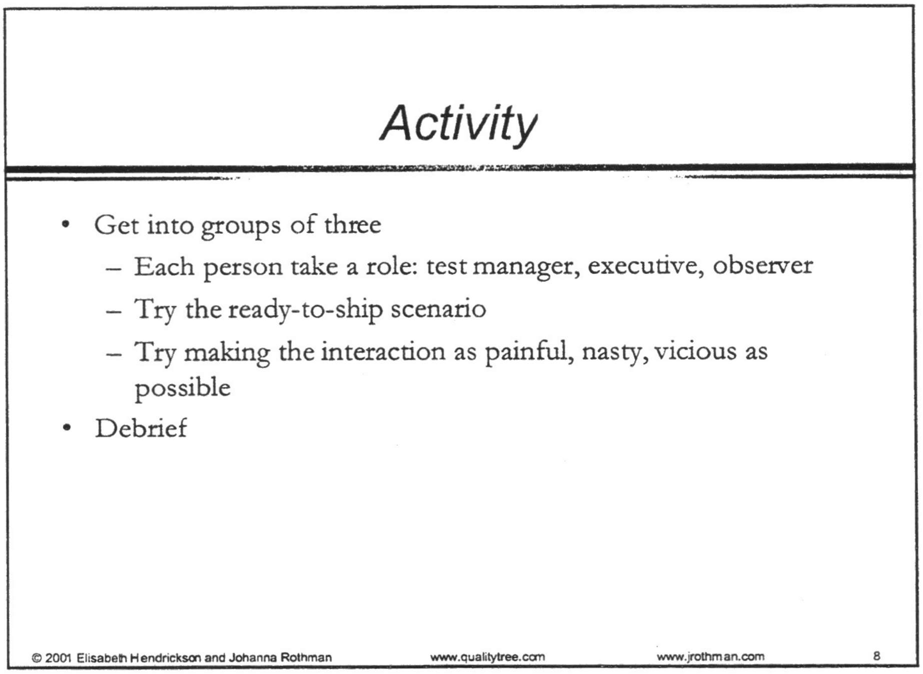
Situation 1: Ready to Ship?

You're the test manager on a project that has been having serious problems since it was first delivered. Your group has found several serious bugs and documented them. So far, the bug review committee has decided to defer all of them. One of your testers just found a really awful bug that results in data corruption. The executives still expect the software to ship Monday. Now what?



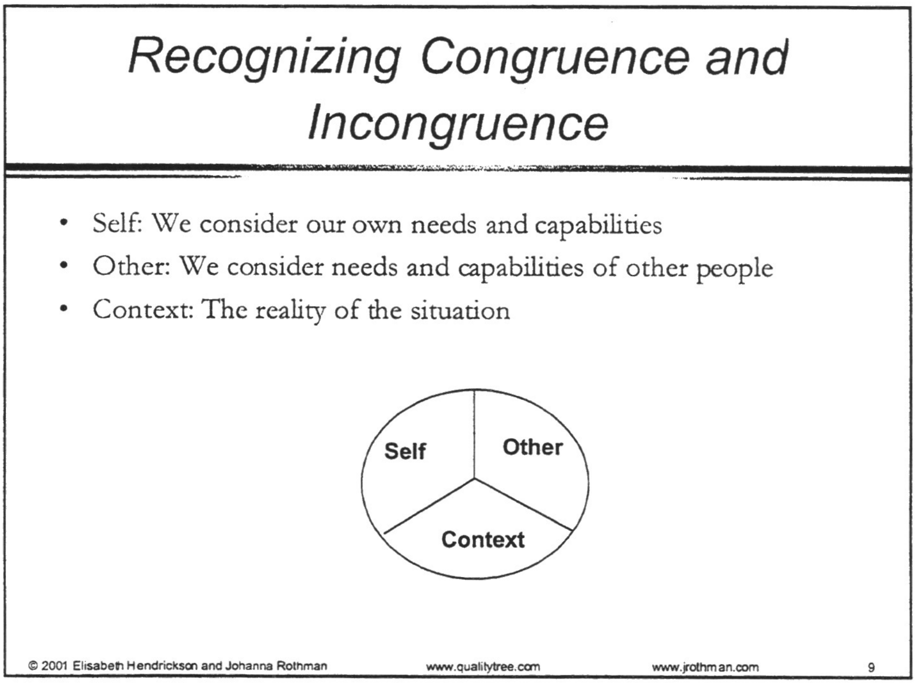
These Responses are NOT Helpful

- Placate the decision makers:
  - You say to yourself: "They're not going to fix anything anyway. So we'll ship bad software. Oh well."
  - Note: Do this enough and you won't have enough Tums.
- Blame others:
  - To your manager: "HOW CAN YOU POSSIBLY CONSIDER SHIPPING THIS PIECE OF !@#$%d!!!?!?!?!?!?"
  - To the developers: "If you weren't such bad programmers, this wouldn't be a problem!"
  - To the executives: "YOU'RE JUST A BUNCH OF QUALITY NEANDERTHALS!"
  - Note: Blaming is best done with a pointed finger and a loud voice



Activity

- Get into groups of three
  - Each person take a role: test manager, executive, observer
  - Try the ready-to-ship scenario
  - Try making the interaction as painful, nasty, vicious as possible
- Debrief



Recognizing Congruence and Incongruence

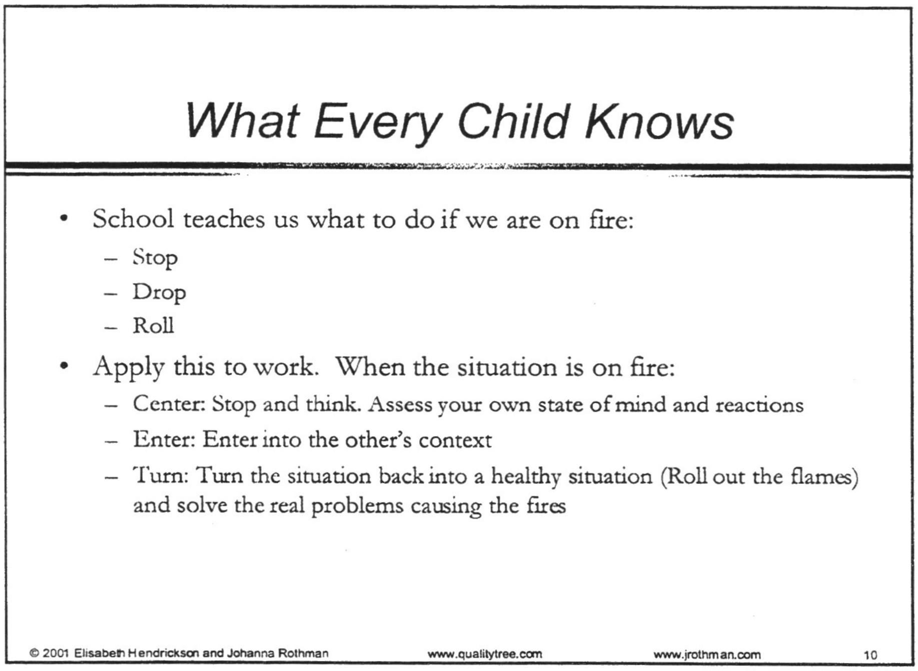
- Self: We consider our own needs and capabilities
- Other: We consider needs and capabilities of other people
- Context: The reality of the situation

Self

Other

Context

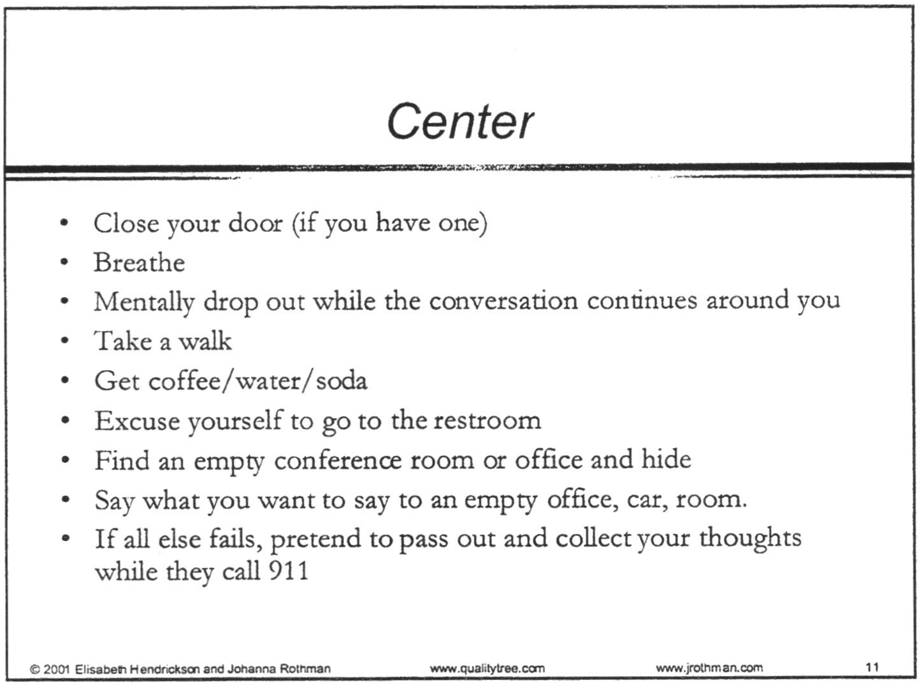
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 9



What Every Child Knows

- School teaches us what to do if we are on fire:
  - Stop
  - Drop
  - Roll
- Apply this to work. When the situation is on fire:
  - Center: Stop and think. Assess your own state of mind and reactions
  - Enter: Enter into the other's context
  - Turn: Turn the situation back into a healthy situation (Roll out the flames) and solve the real problems causing the fires

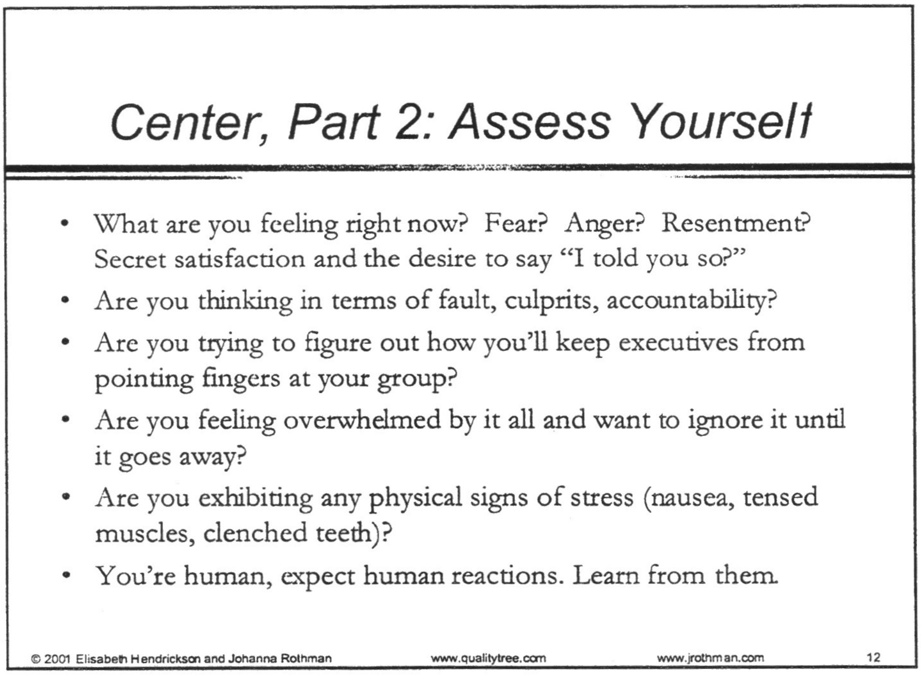
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 10



Center

- Close your door (if you have one)
- Breathe
- Mentally drop out while the conversation continues around you
- Take a walk
- Get coffee/water/soda
- Excuse yourself to go to the restroom
- Find an empty conference room or office and hide
- Say what you want to say to an empty office, car, room.
- If all else fails, pretend to pass out and collect your thoughts while they call 911

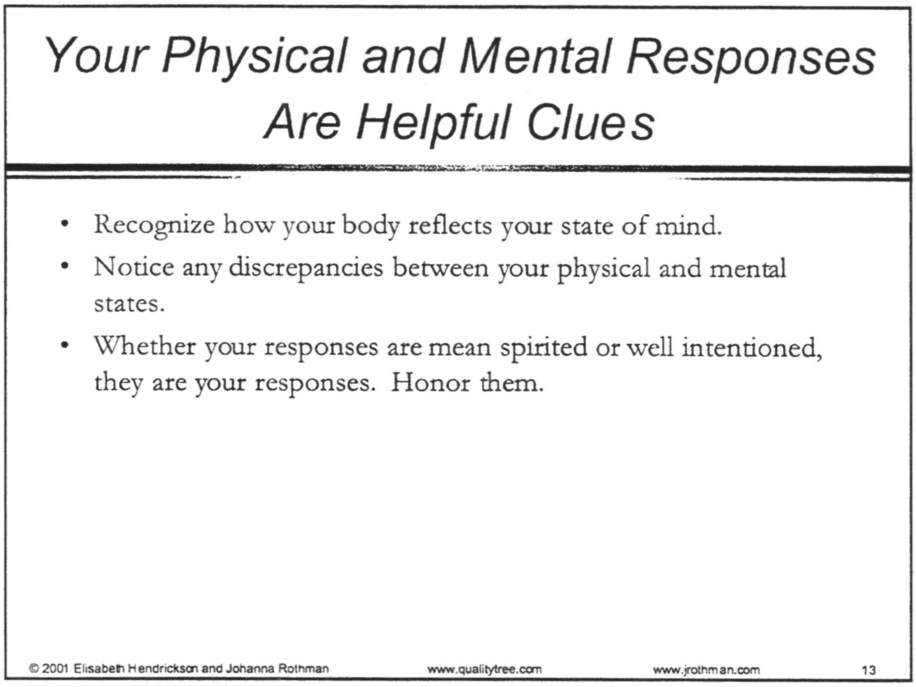
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 11



Center, Part 2: Assess Yourself

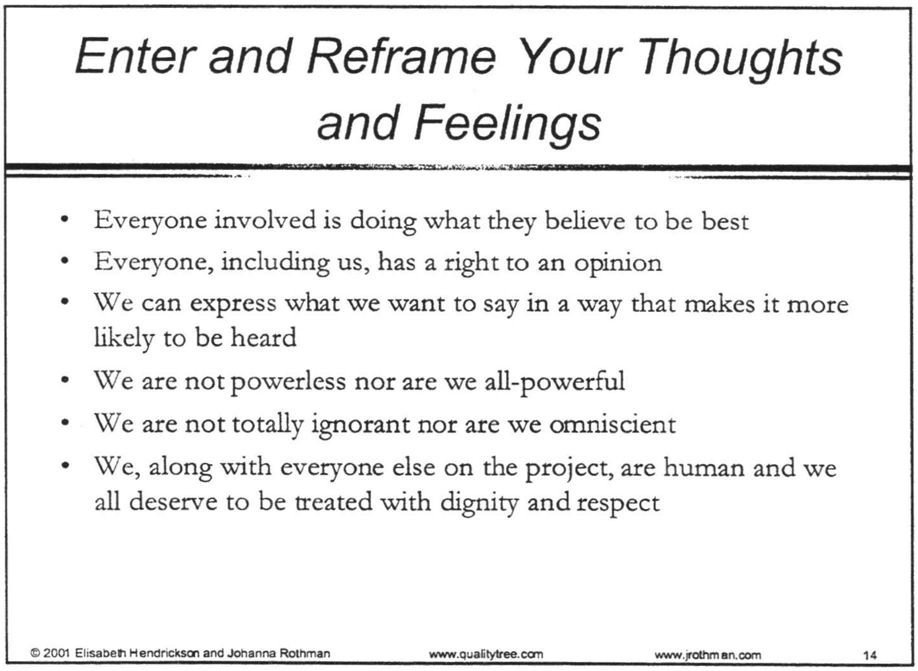
- What are you feeling right now? Fear? Anger? Resentment? Secret satisfaction and the desire to say "I told you so?"
- Are you thinking in terms of fault, culprits, accountability?
- Are you trying to figure out how you'll keep executives from pointing fingers at your group?
- Are you feeling overwhelmed by it all and want to ignore it until it goes away?
- Are you exhibiting any physical signs of stress (nausea, tensed muscles, clenched teeth)?
- You're human, expect human reactions. Learn from them.

© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 12



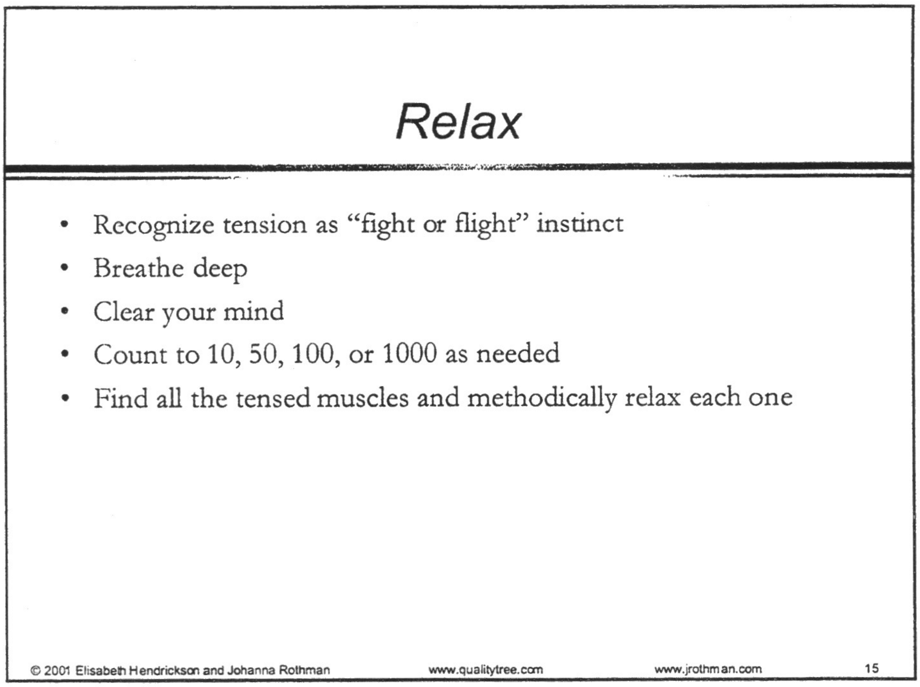
Your Physical and Mental Responses Are Helpful Clues

- Recognize how your body reflects your state of mind.
- Notice any discrepancies between your physical and mental states.
- Whether your responses are mean spirited or well intentioned, they are your responses. Honor them.



Enter and Reframe Your Thoughts and Feelings

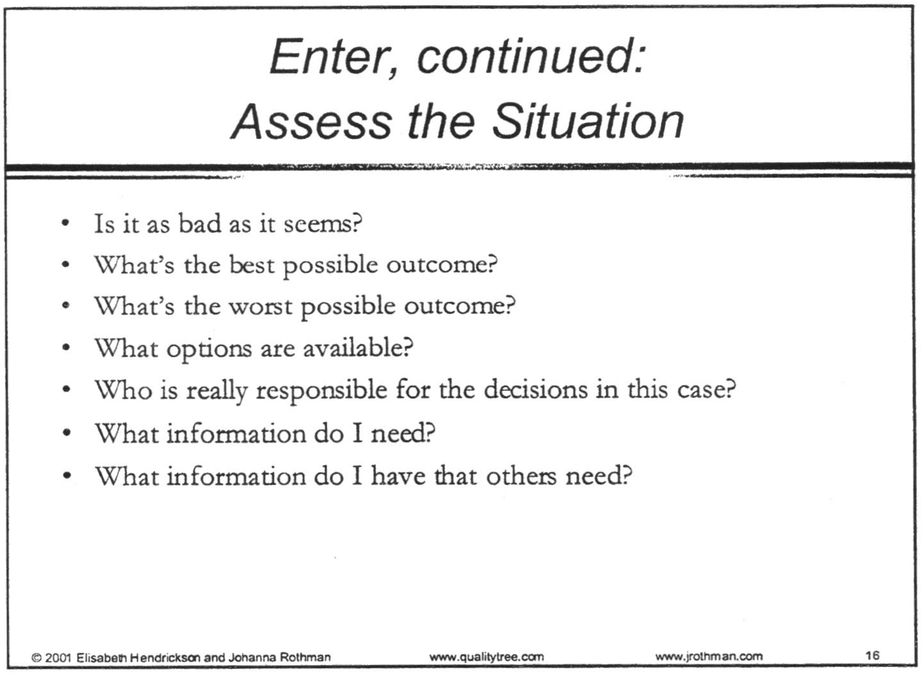
- Everyone involved is doing what they believe to be best
- Everyone, including us, has a right to an opinion
- We can express what we want to say in a way that makes it more likely to be heard
- We are not powerless nor are we all-powerful
- We are not totally ignorant nor are we omniscient
- We, along with everyone else on the project, are human and we all deserve to be treated with dignity and respect



Relax

- Recognize tension as "fight or flight" instinct
- Breathe deep
- Clear your mind
- Count to 10, 50, 100, or 1000 as needed
- Find all the tensed muscles and methodically relax each one

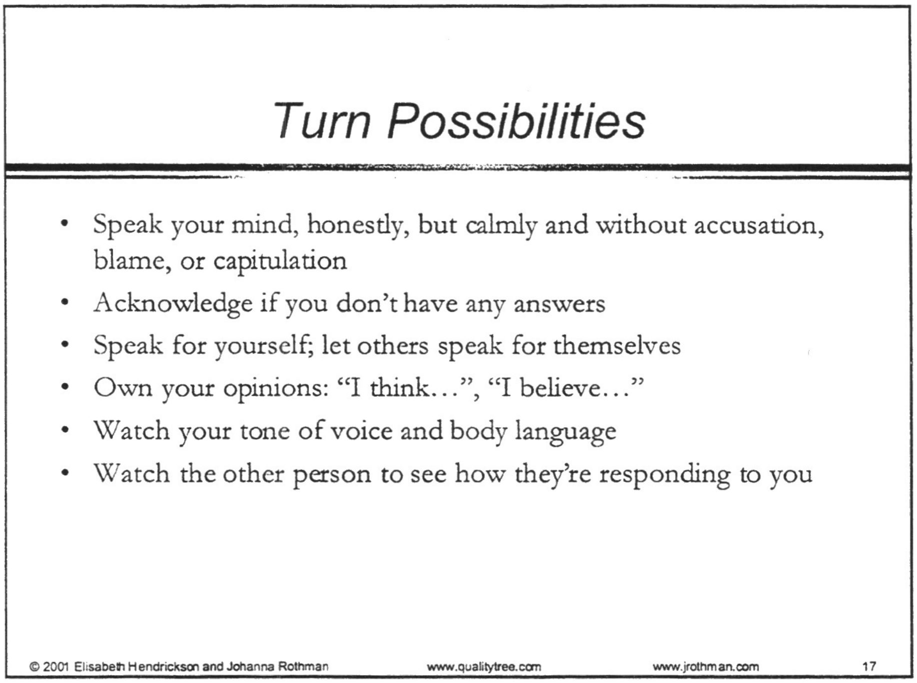
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 15



Enter, continued: Assess the Situation

- Is it as bad as it seems?
- What's the best possible outcome?
- What's the worst possible outcome?
- What options are available?
- Who is really responsible for the decisions in this case?
- What information do I need?
- What information do I have that others need?

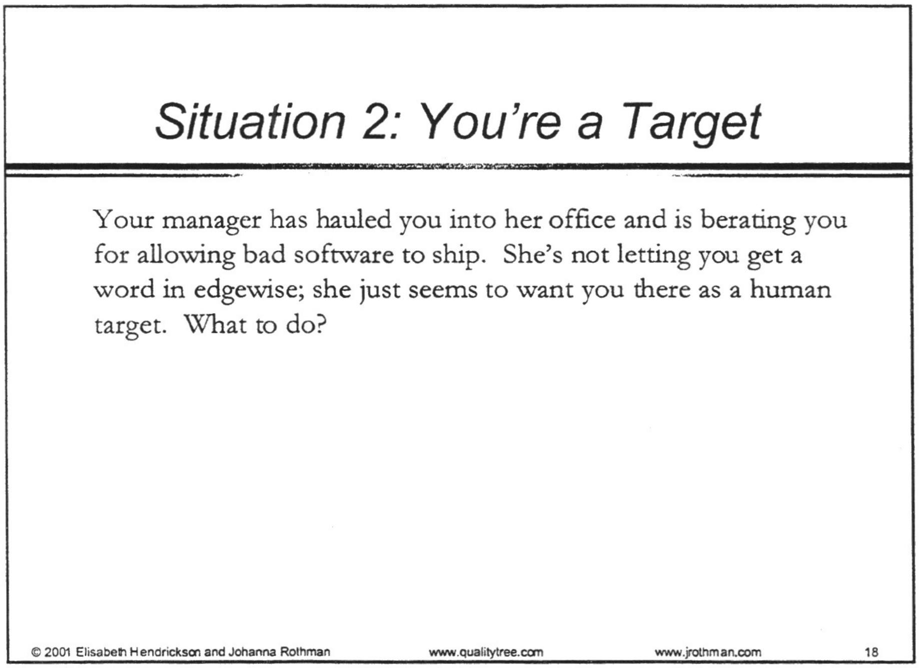
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 16



Turn Possibilities

- Speak your mind, honestly, but calmly and without accusation, blame, or capitulation
- Acknowledge if you don't have any answers
- Speak for yourself; let others speak for themselves
- Own your opinions: "I think…", "I believe…"
- Watch your tone of voice and body language
- Watch the other person to see how they're responding to you

© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 17

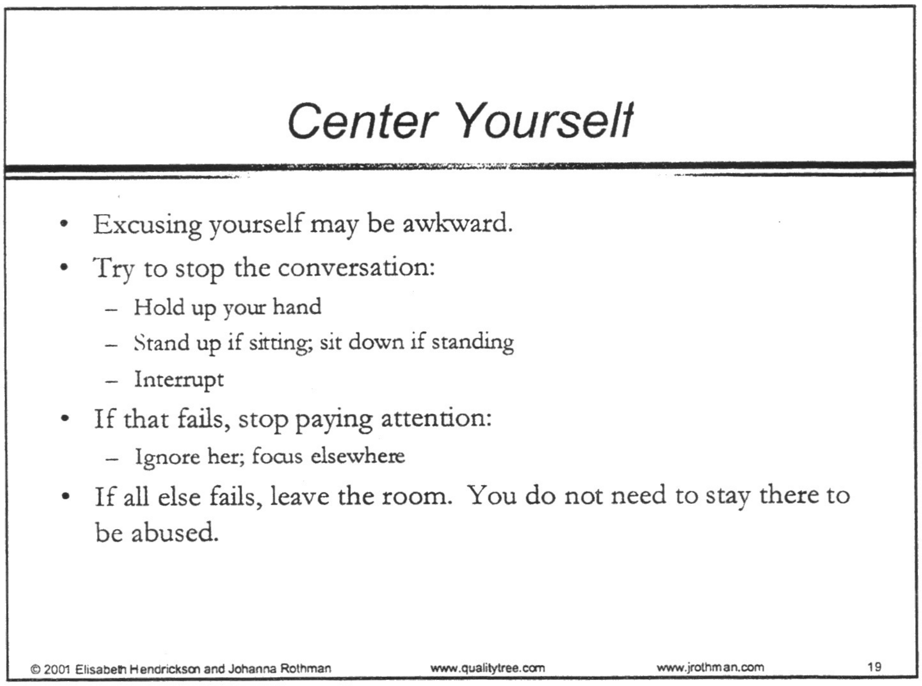


Situation 2: You're a Target

Your manager has hauled you into her office and is berating you for allowing bad software to ship. She's not letting you get a word in edgewise; she just seems to want you there as a human target. What to do?

© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 18

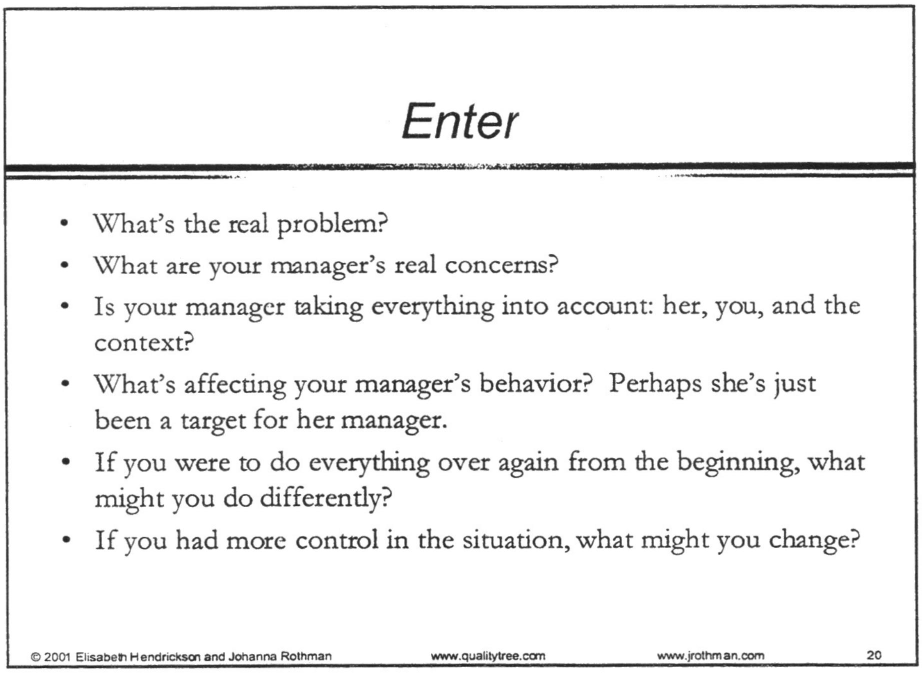
Workshop 4



Center Yourself

- Excusing yourself may be awkward.
- Try to stop the conversation:
  - Hold up your hand
  - Stand up if sitting; sit down if standing
  - Interrupt
- If that fails, stop paying attention:
  - Ignore her; focus elsewhere
- If all else fails, leave the room. You do not need to stay there to be abused.

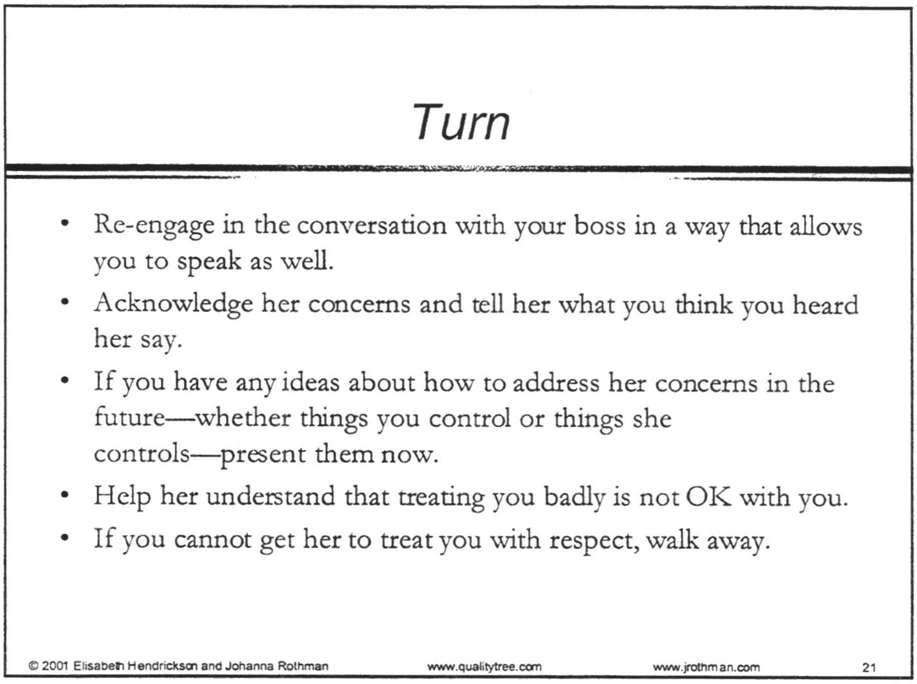
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 19



Enter

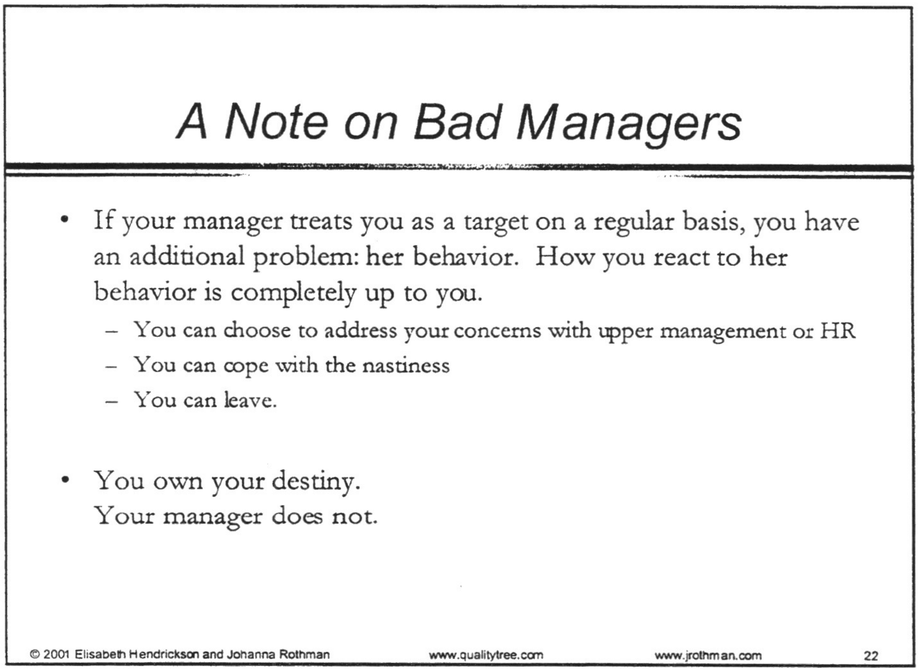
- What's the real problem?
- What are your manager's real concerns?
- Is your manager taking everything into account: her, you, and the context?
- What's affecting your manager's behavior? Perhaps she's just been a target for her manager.
- If you were to do everything over again from the beginning, what might you do differently?
- If you had more control in the situation, what might you change?

© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 20



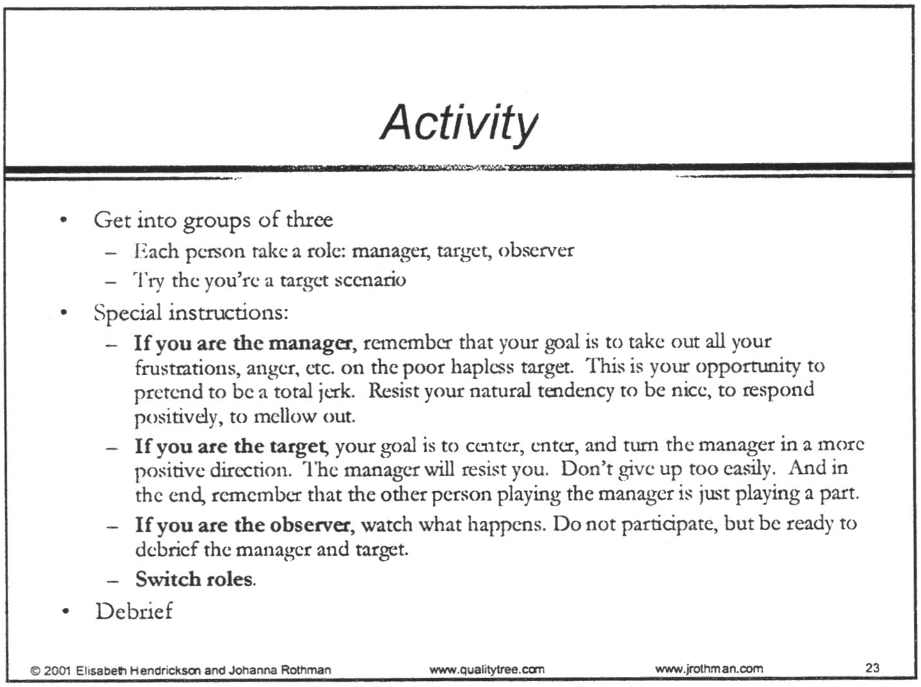
Turn

- Re-engage in the conversation with your boss in a way that allows you to speak as well.
- Acknowledge her concerns and tell her what you think you heard her say.
- If you have any ideas about how to address her concerns in the future—whether things you control or things she controls—present them now.
- Help her understand that treating you badly is not OK with you.
- If you cannot get her to treat you with respect, walk away.



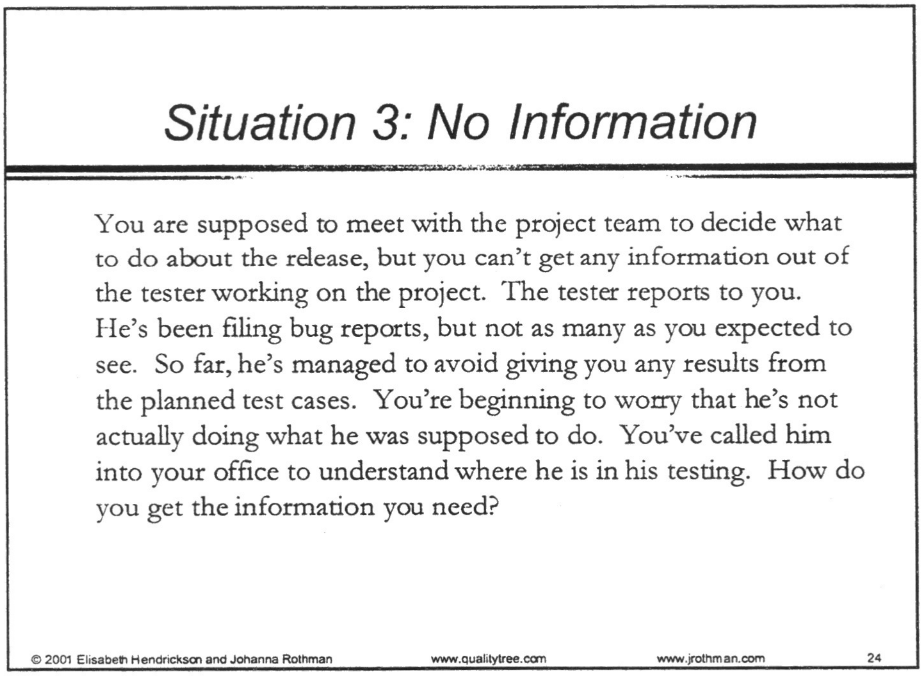
A Note on Bad Managers

- If your manager treats you as a target on a regular basis, you have an additional problem: her behavior. How you react to her behavior is completely up to you.
  - You can choose to address your concerns with upper management or HR
  - You can cope with the nastiness
  - You can leave.
- You own your destiny.
  Your manager does not.



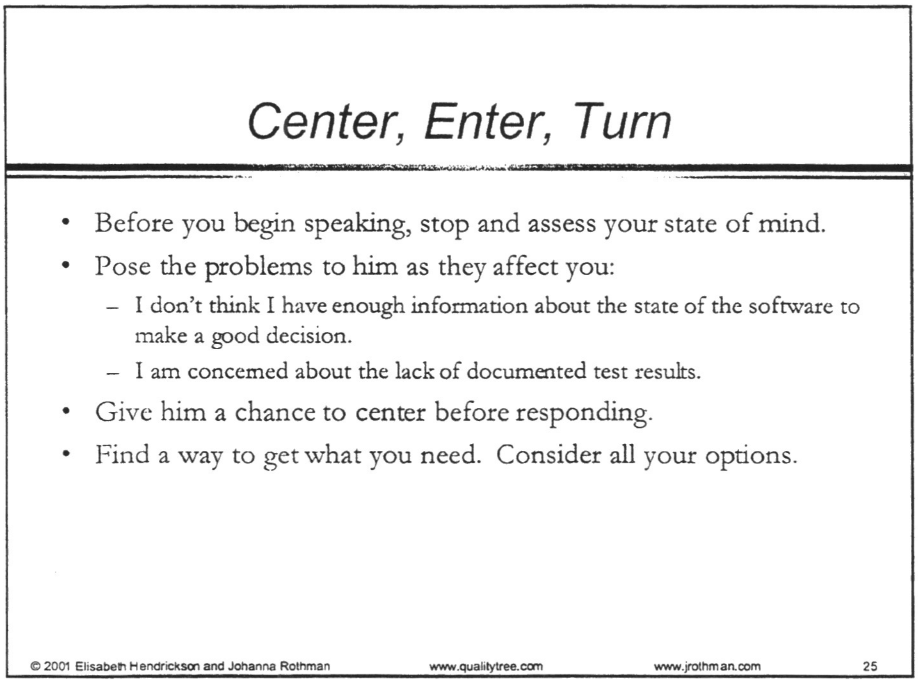
Activity

- Get into groups of three
  - Each person take a role: manager, target, observer
  - Try the you're a target scenario
- Special instructions:
  - **If you are the manager**, remember that your goal is to take out all your frustrations, anger, etc. on the poor hapless target. This is your opportunity to pretend to be a total jerk. Resist your natural tendency to be nice, to respond positively, to mellow out.
  - **If you are the target**, your goal is to center, enter, and turn the manager in a more positive direction. The manager will resist you. Don't give up too easily. And in the end, remember that the other person playing the manager is just playing a part.
  - **If you are the observer**, watch what happens. Do not participate, but be ready to debrief the manager and target.
  - **Switch roles.**
- Debrief



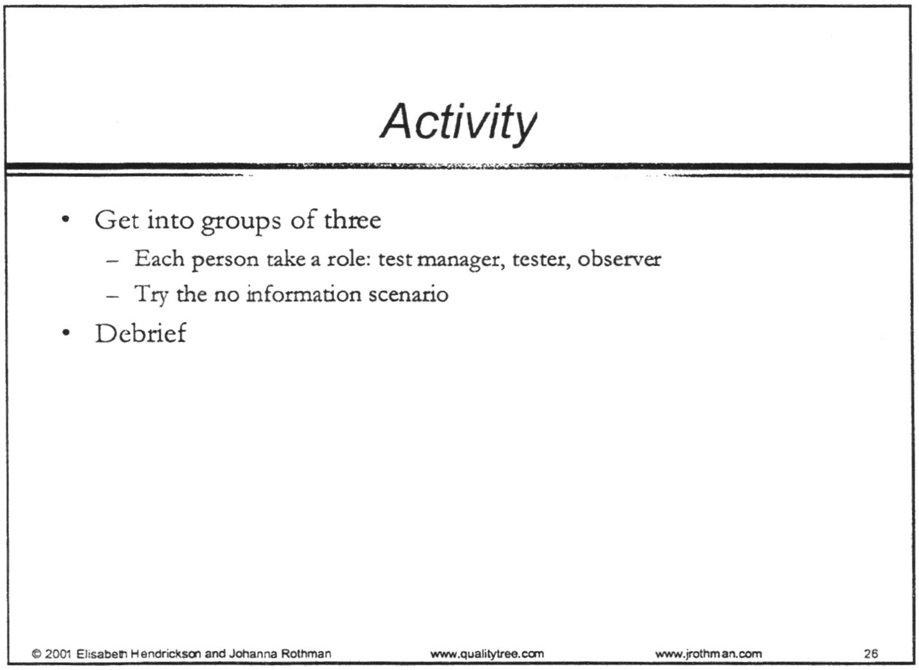
Situation 3: No Information

You are supposed to meet with the project team to decide what to do about the release, but you can't get any information out of the tester working on the project. The tester reports to you. He's been filing bug reports, but not as many as you expected to see. So far, he's managed to avoid giving you any results from the planned test cases. You're beginning to worry that he's not actually doing what he was supposed to do. You've called him into your office to understand where he is in his testing. How do you get the information you need?



Center, Enter, Turn

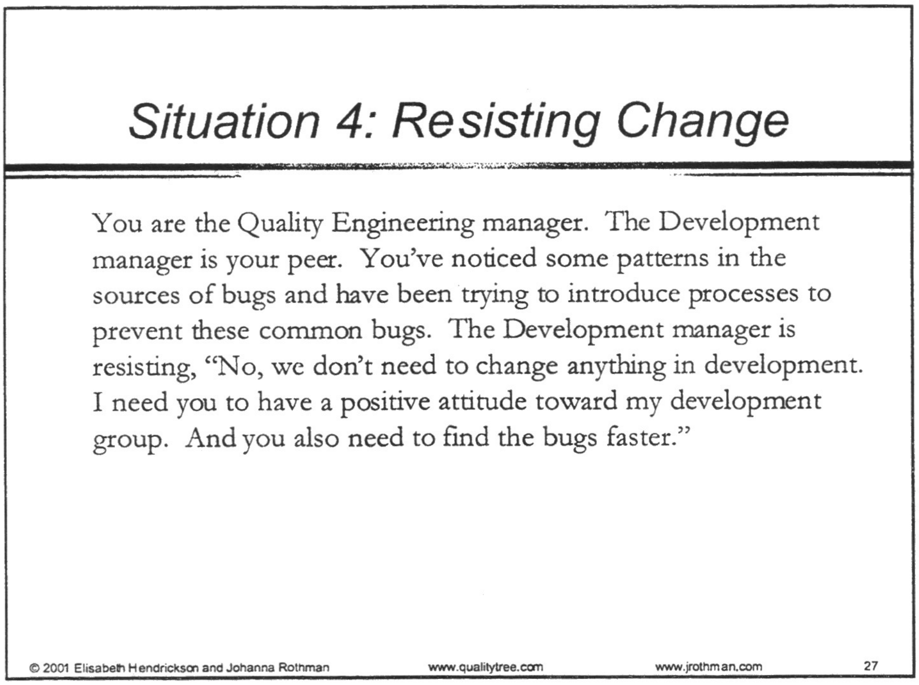
- Before you begin speaking, stop and assess your state of mind.
- Pose the problems to him as they affect you:
  - I don't think I have enough information about the state of the software to make a good decision.
  - I am concemed about the lack of documented test results.
- Give him a chance to center before responding.
- Find a way to get what you need. Consider all your options.



Activity

- Get into groups of three
  - Each person take a role: test manager, tester, observer
  - Try the no information scenario
- Debrief

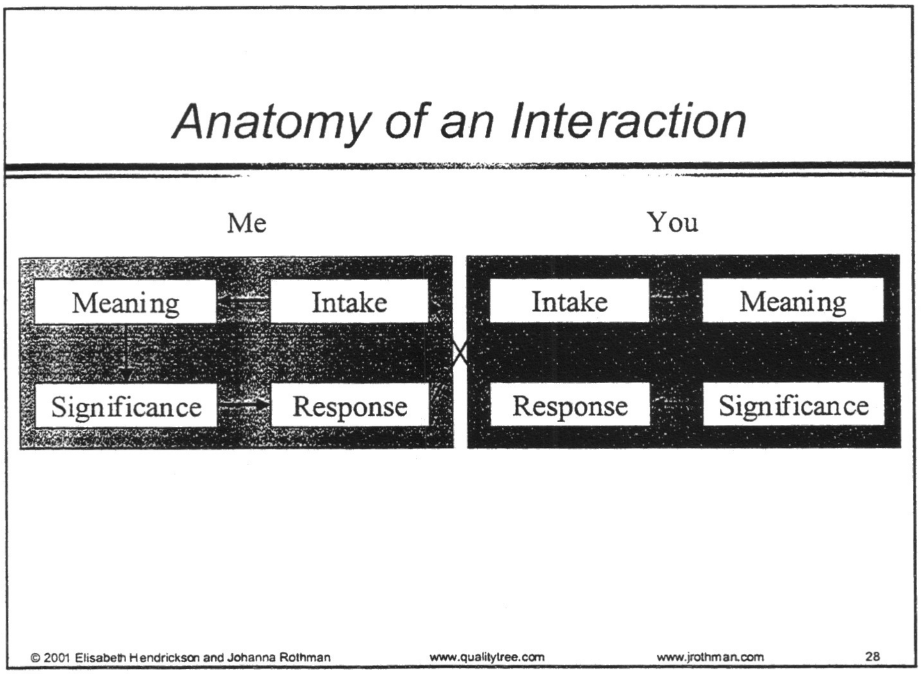
Workshop 4



Situation 4: Resisting Change

You are the Quality Engineering manager. The Development manager is your peer. You've noticed some patterns in the sources of bugs and have been trying to introduce processes to prevent these common bugs. The Development manager is resisting, "No, we don't need to change anything in development. I need you to have a positive attitude toward my development group. And you also need to find the bugs faster."

© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 27



Anatomy of an Interaction

Me

Meaning Intake

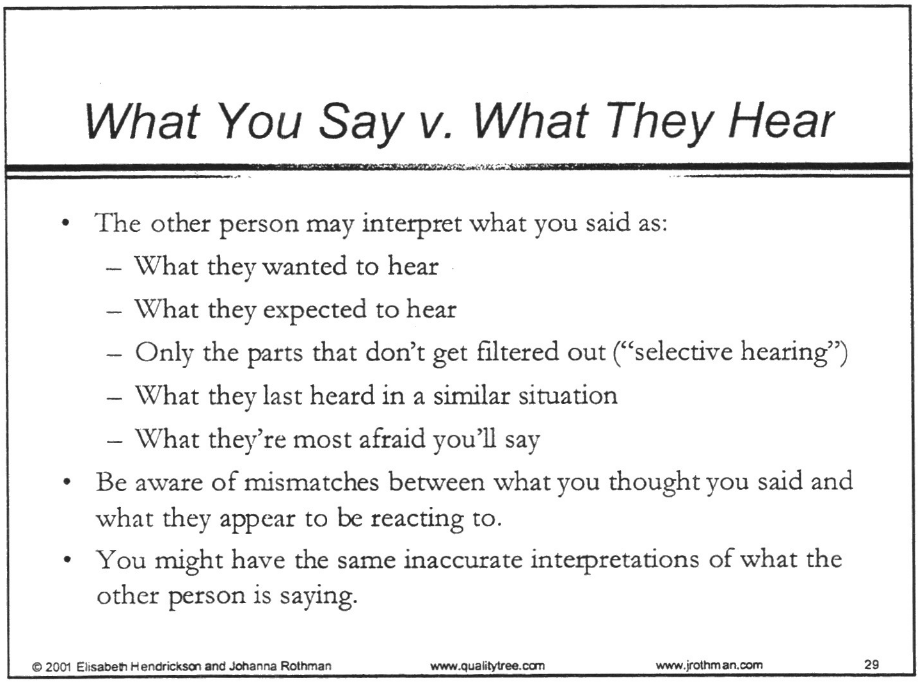
Significance Response

You

Intake Meaning

Response Significance

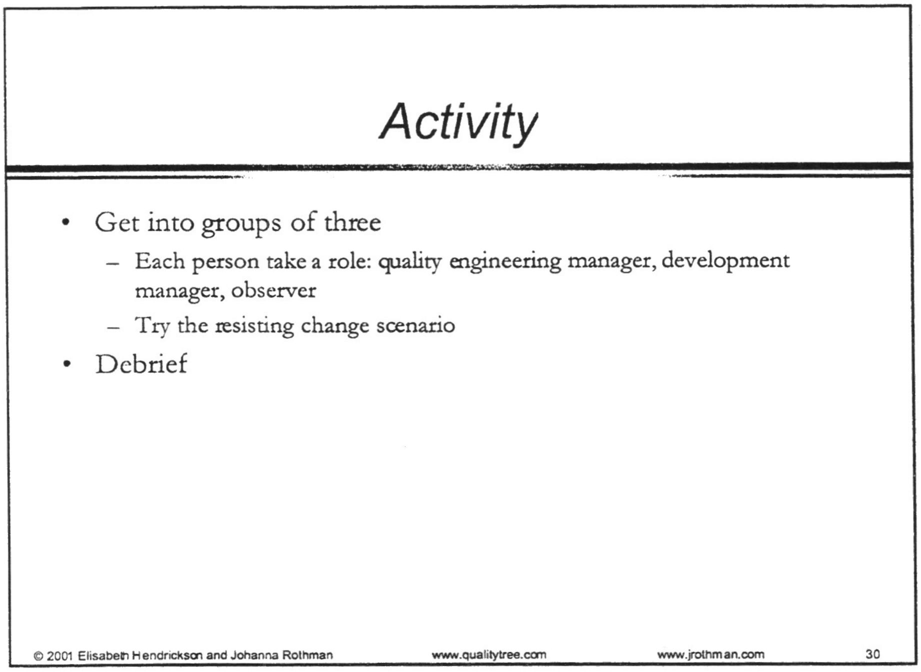
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 28



What You Say v. What They Hear

- The other person may interpret what you said as:
  - What they wanted to hear
  - What they expected to hear
  - Only the parts that don't get filtered out ("selective hearing")
  - What they last heard in a similar situation
  - What they're most afraid you'll say
- Be aware of mismatches between what you thought you said and what they appear to be reacting to.
- You might have the same inaccurate interpretations of what the other person is saying.

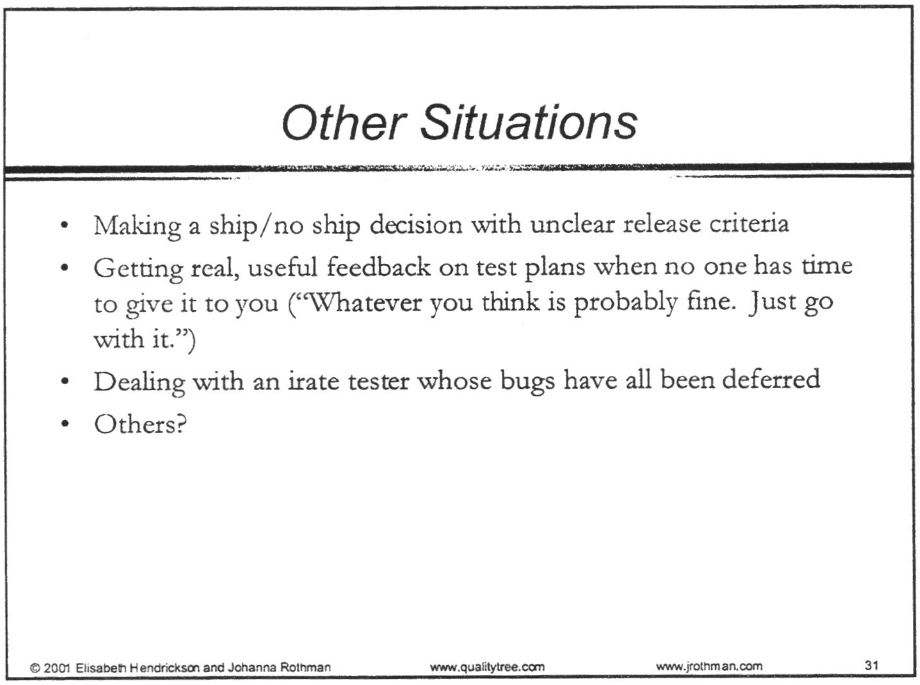
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 29



Activity

- Get into groups of three
  - Each person take a role: quality engineering manager, development manager, observer
  - Try the resisting change scenario
- Debrief

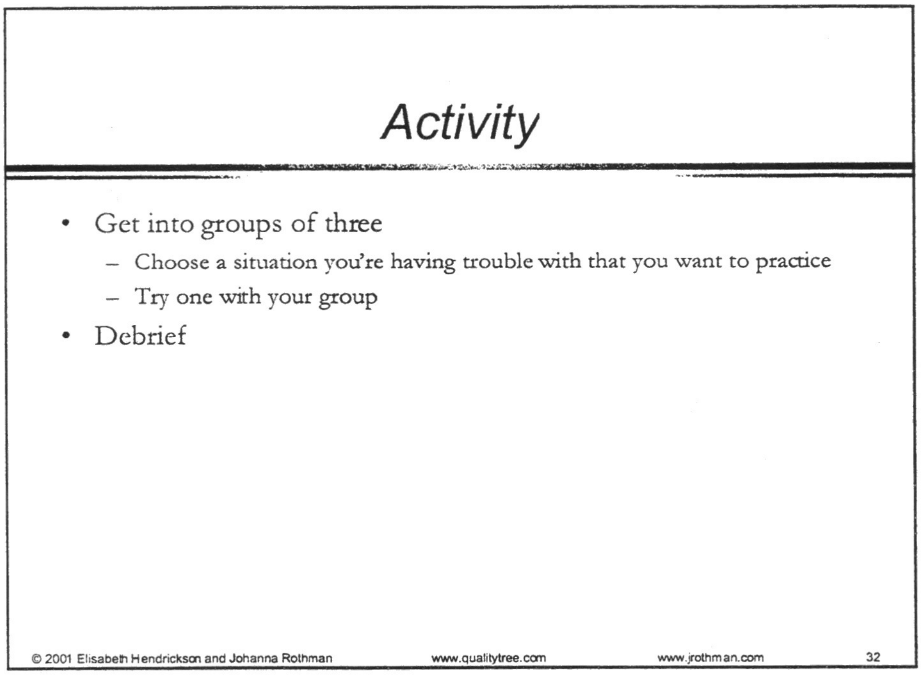
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 30



Other Situations

- Making a ship/no ship decision with unclear release criteria
- Getting real, useful feedback on test plans when no one has time to give it to you ("Whatever you think is probably fine. Just go with it.")
- Dealing with an irate tester whose bugs have all been deferred
- Others?

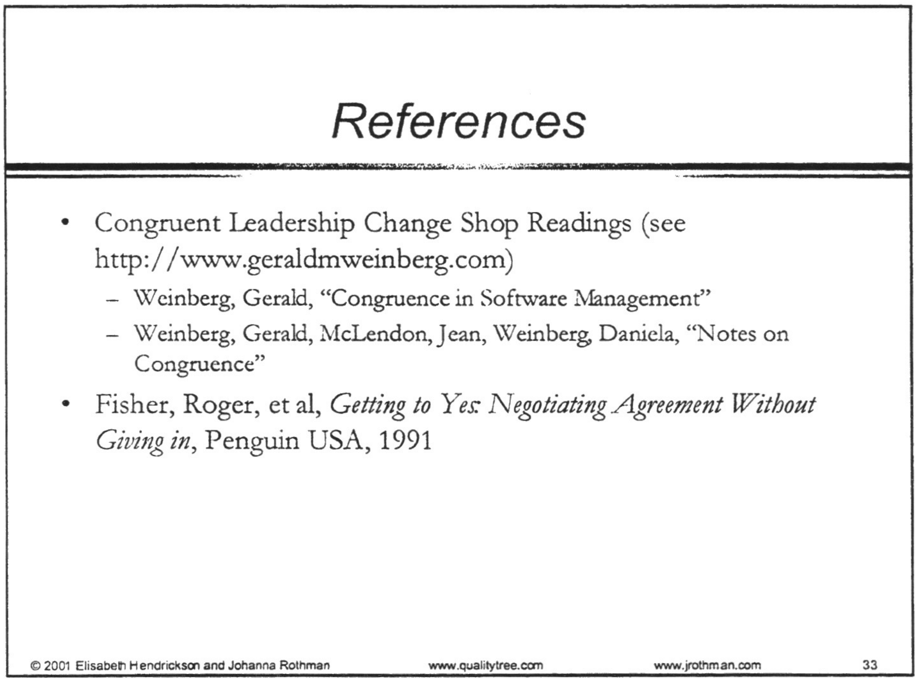
© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 31



Activity

- Get into groups of three
  - Choose a situation you're having trouble with that you want to practice
  - Try one with your group
- Debrief

© 2001 Elisabeth Hendrickson and Johanna Rothman www.qualitytree.com www.jrothman.com 32



References

- Congruent Leadership Change Shop Readings (see http://www.geraldmweinberg.com)
  - Weinberg, Gerald, "Congruence in Software Management"
  - Weinberg, Gerald, McLendon, Jean, Weinberg, Daniela, "Notes on Congruence"
- Fisher, Roger, et al, *Getting to Yes: Negotiating Agreement Without Giving in*, Penguin USA, 1991