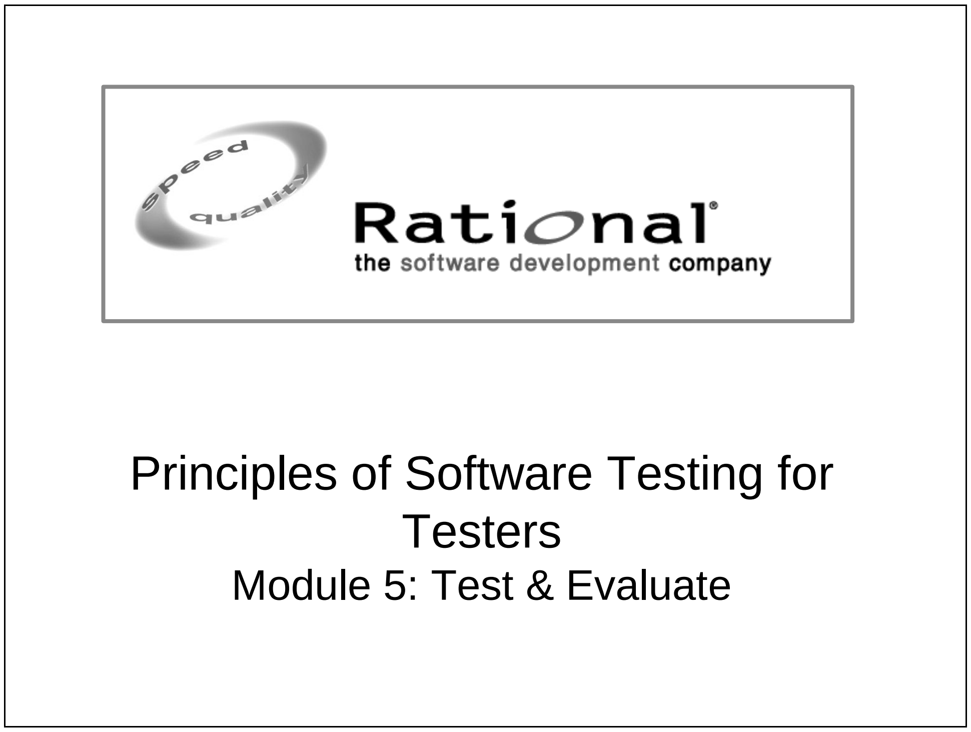
Module 5
Test and Evaluate



Principles of Software Testing for Testers

Module 5: Test & Evaluate

Topics

Agenda 5-2

Test and Evaluate Workflow: Test...5-4

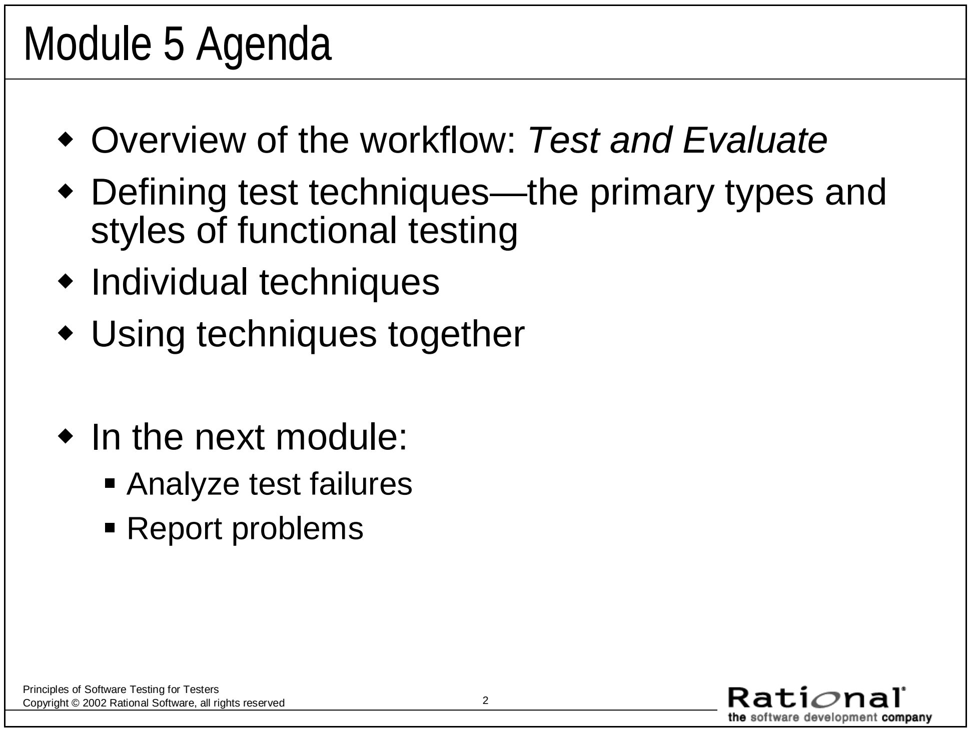
Defining Test Techniques..5-6

Individual Techniques...5-13

Using Techniques Together 5-64

How To Adopt New Techniques/Review 5-68

Agenda



Module 5 Agenda

- Overview of the workflow: *Test and Evaluate*
- Defining test techniques—the primary types and styles of functional testing
- Individual techniques
- Using techniques together

- In the next module:
  - Analyze test failures
  - Report problems

Principles of Software Testing for Testers

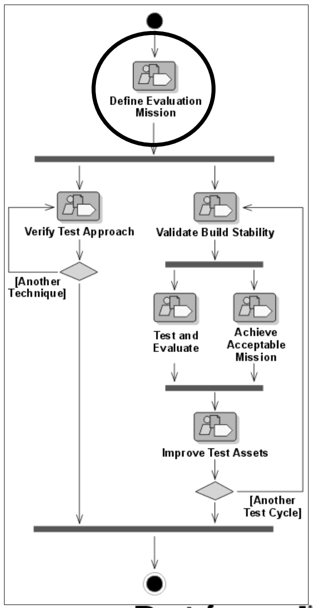
2

Rational
the software development company

Review: Where We've Been

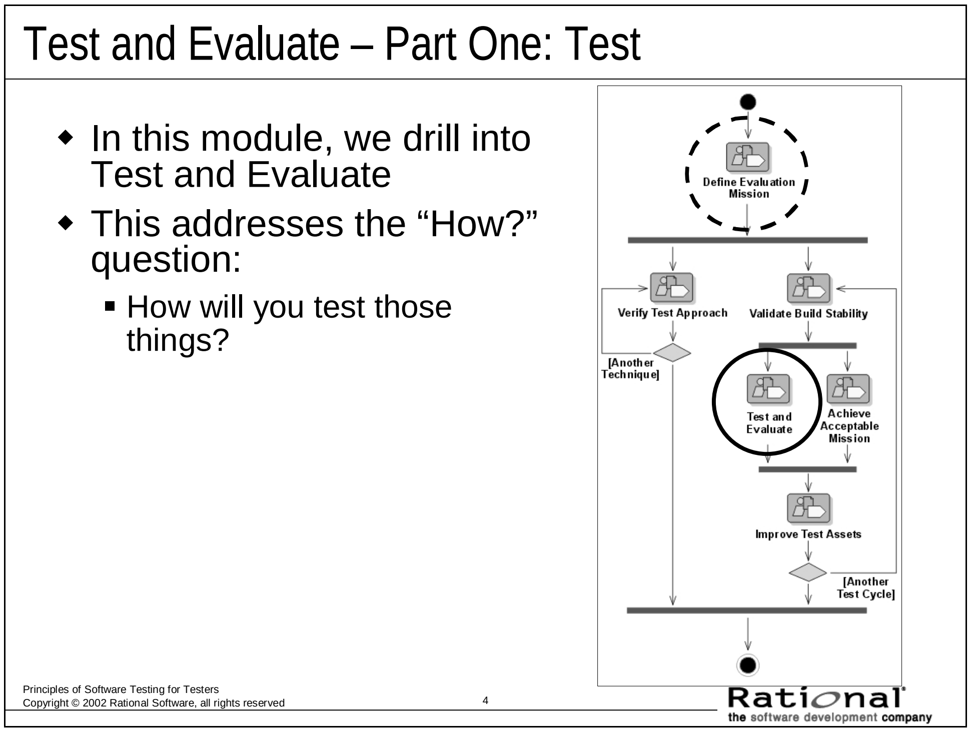
Review: Where We've Been

- In the last module, we covered the workflow detail Define Evaluation Mission
- The Mission focuses on the high-level objectives of the test team for the current iteration
  - What things should motivate us to test?
  - Why these things (and not others)?



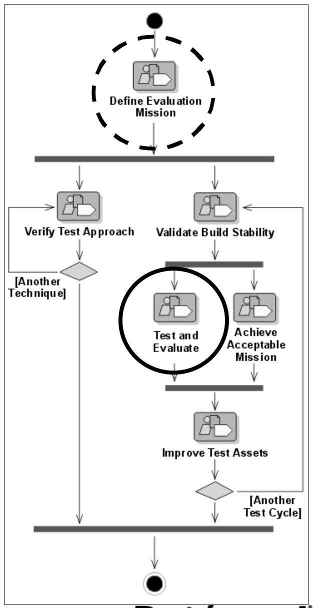
3
Rational
the software development company

Test and Evaluate Workflow: Test



Test and Evaluate – Part One: Test

- In this module, we drill into Test and Evaluate
- This addresses the "How?" question:
  - How will you test those things?



Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 4

Rational the software development company

The purpose of this workflow detail is to achieve appropriate breadth and depth of the test effort to enable a sufficient evaluation of the Target Test Items — where sufficient evaluation is governed by the Test Motivators and Evaluation Mission.

For each test cycle, this work is focused mainly on:

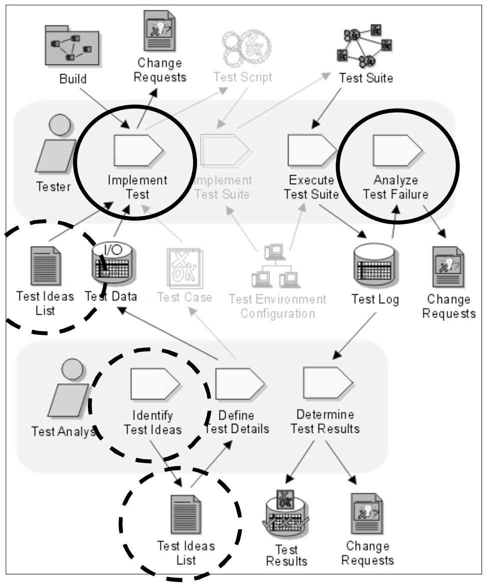
- Achieving suitable breadth and depth in the test and evaluation work

This is the heart of the test cycle, doing the testing itself and analyzing the results.

Test and Evaluate – Part One: Test

Test and Evaluate – Part One: Test

- This module focuses on the activity *Implement Test*
- Earlier, we covered *Test-Idea Lists*, which are input here
- In the next module, we'll cover *Analyze Test Failures*, the second half of *Test and Evaluate*



5
Rational the software development company

Here are the roles, activities and artifacts RUP focuses on in this work.

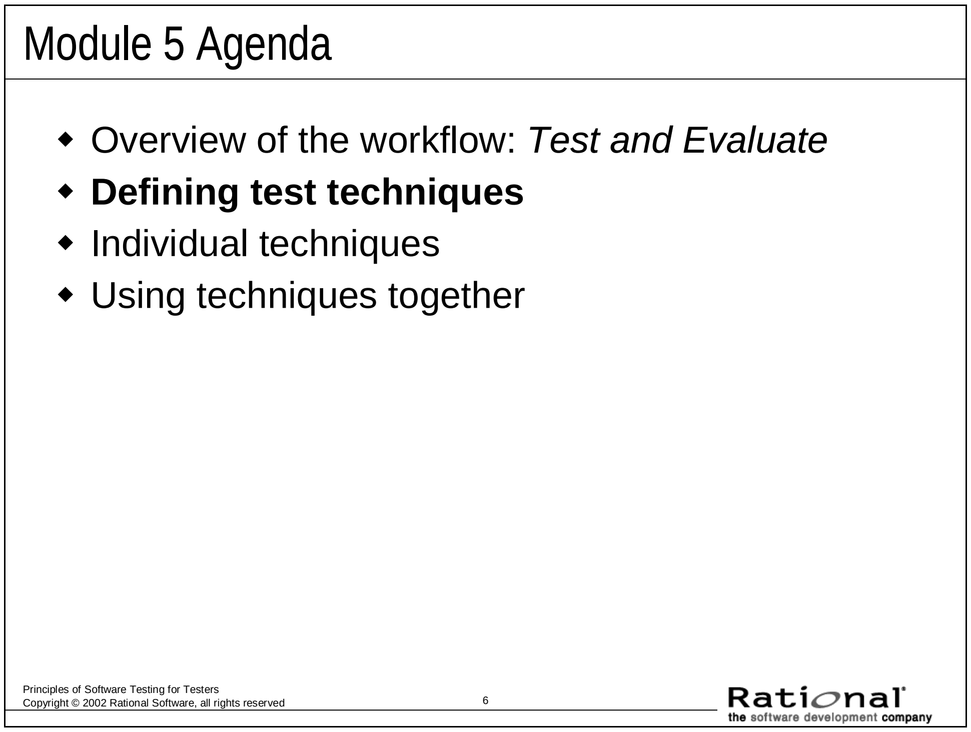
In earlier modules, we discussed how identifying test ideas is a useful way to reason about tests early in the lifecycle without needing to completely define each specific test.

In this module we'll look at a selection of techniques that can be used to apply those test ideas.

In the next module, we'll talk more about evaluating the output of the tests that have been run.

Note that diagram shows some grayed-out elements: these are additional testing elements that RUP provides guidance for which not covered directly in this course. You can found out more about these elements by consulting RUP directly.

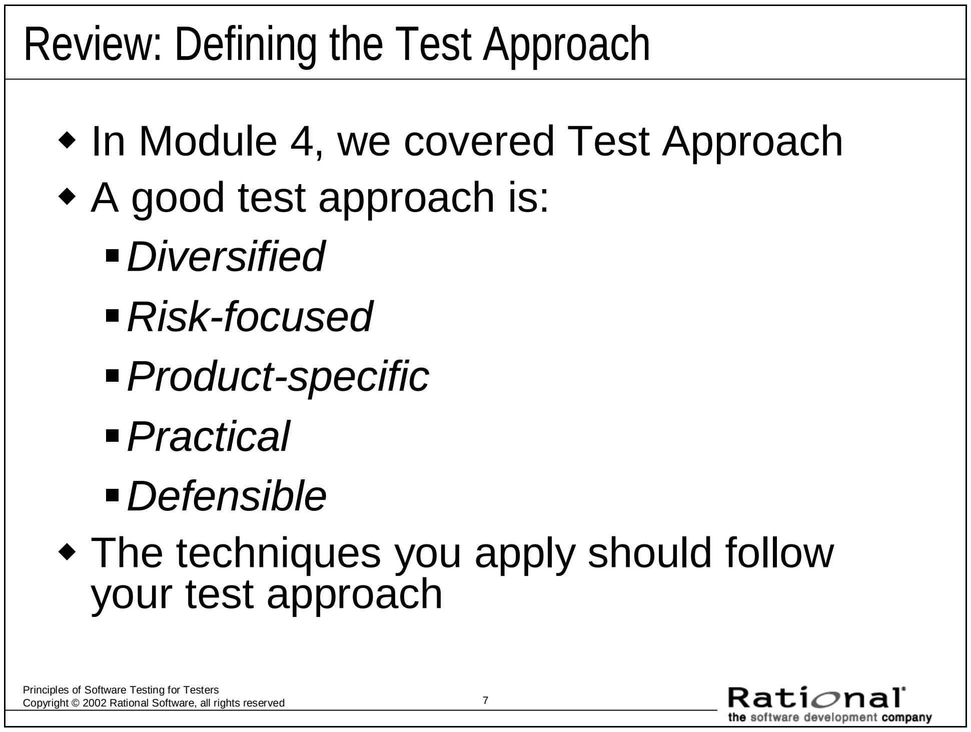
Defining Test Techniques



Module 5 Agenda

- Overview of the workflow: *Test and Evaluate*
- **Defining test techniques**
- Individual techniques
- Using techniques together

Review: Defining the Test Approach



Review: Defining the Test Approach

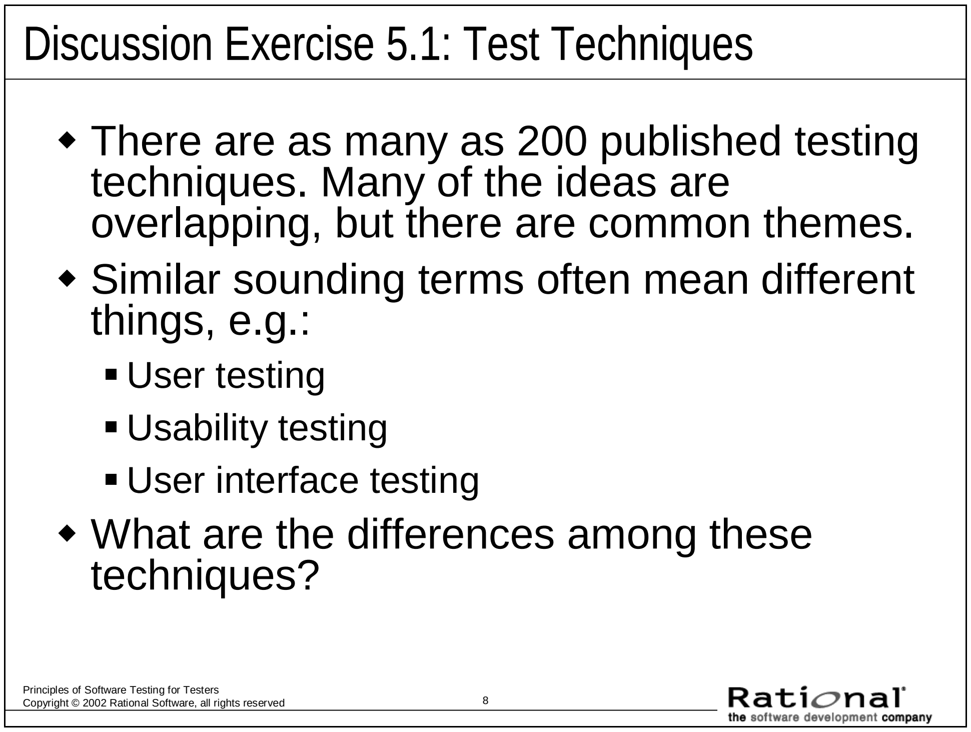
- In Module 4, we covered Test Approach
- A good test approach is:
  - *Diversified*
  - *Risk-focused*
  - *Product-specific*
  - *Practical*
  - *Defensible*
- The techniques you apply should follow your test approach

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 7

Rational the software development company

In Module 4, we discussed Test Approach and mentioned techniques. Here we'll drill into the techniques that you might use.

Discussion Exercise 5.1: Test Techniques



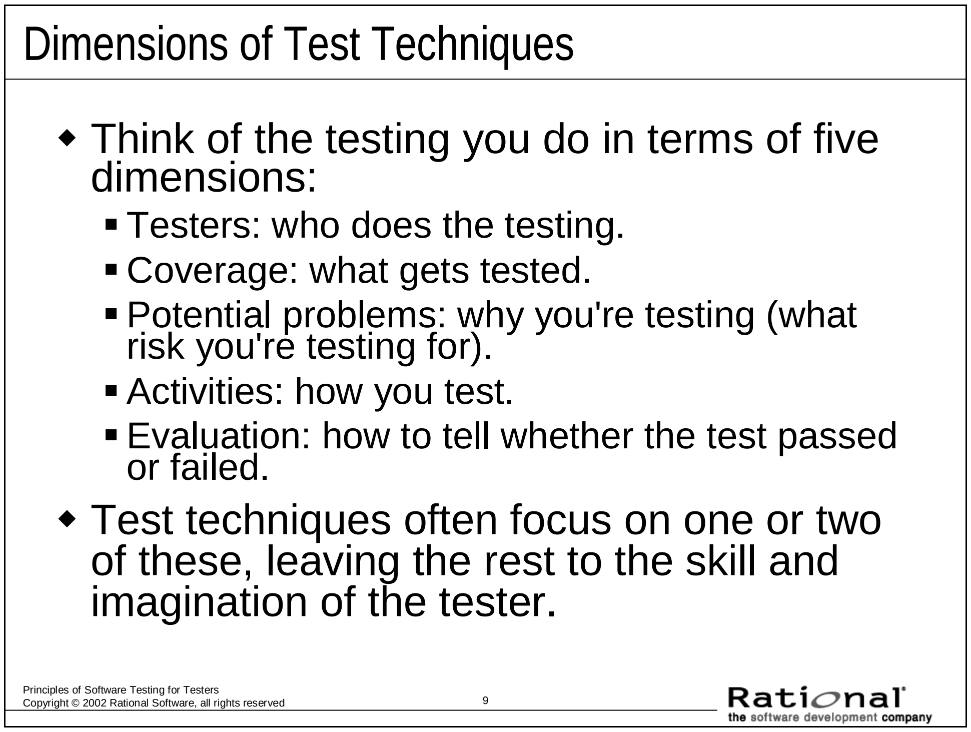
Discussion Exercise 5.1: Test Techniques

- There are as many as 200 published testing techniques. Many of the ideas are overlapping, but there are common themes.
- Similar sounding terms often mean different things, e.g.:
  - User testing
  - Usability testing
  - User interface testing
- What are the differences among these techniques?

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 8

Rational
the software development company

Dimensions of Test Techniques



Dimensions of Test Techniques

- Think of the testing you do in terms of five dimensions:
  - Testers: who does the testing.
  - Coverage: what gets tested.
  - Potential problems: why you're testing (what risk you're testing for).
  - Activities: how you test.
  - Evaluation: how to tell whether the test passed or failed.
- Test techniques often focus on one or two of these, leaving the rest to the skill and imagination of the tester.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 9

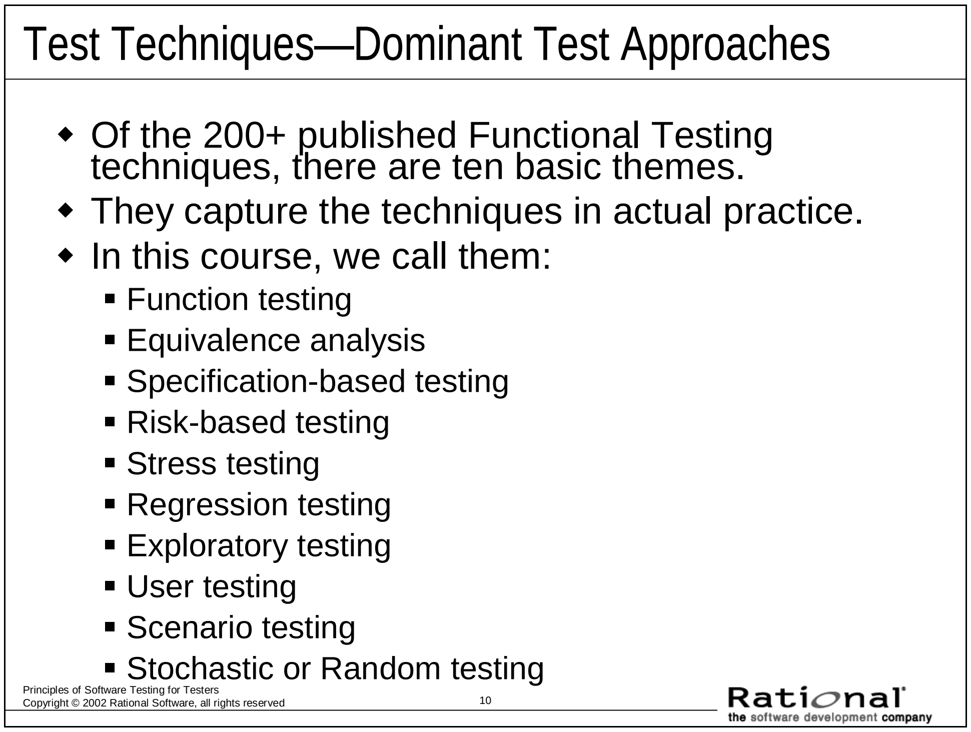
Rational
the software development company

Examples of the dimensions:

1. ***Testers***: <u>*User testing*</u> is focused on testing by members of your target market, people who would normally use the product.
2. ***Coverage***: <u>*User interface testing*</u> is focused on the elements of the user interface, such as the menus and other controls. Focusing on this testing involves testing every UI element.
3. ***Potential problems***: Testing for <u>*usability errors*</u> or other problems that would make people abandon the product or be unhappy with it.
4. ***Activities***: Exploratory testing.
5. ***Evaluation***: Comparison to a result provided by a known good program, a test oracle.

Functional testing is roughly synonymous with “behavioral testing” or “black box” testing. The fundamental idea is that your testing is focused on the inputs that you give the program and the responses you get from it. A wide range of techniques fit within this general approach.

Test Techniques—Dominant Test Approaches



Test Techniques—Dominant Test Approaches

- Of the 200+ published Functional Testing techniques, there are ten basic themes.
- They capture the techniques in actual practice.
- In this course, we call them:
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 10

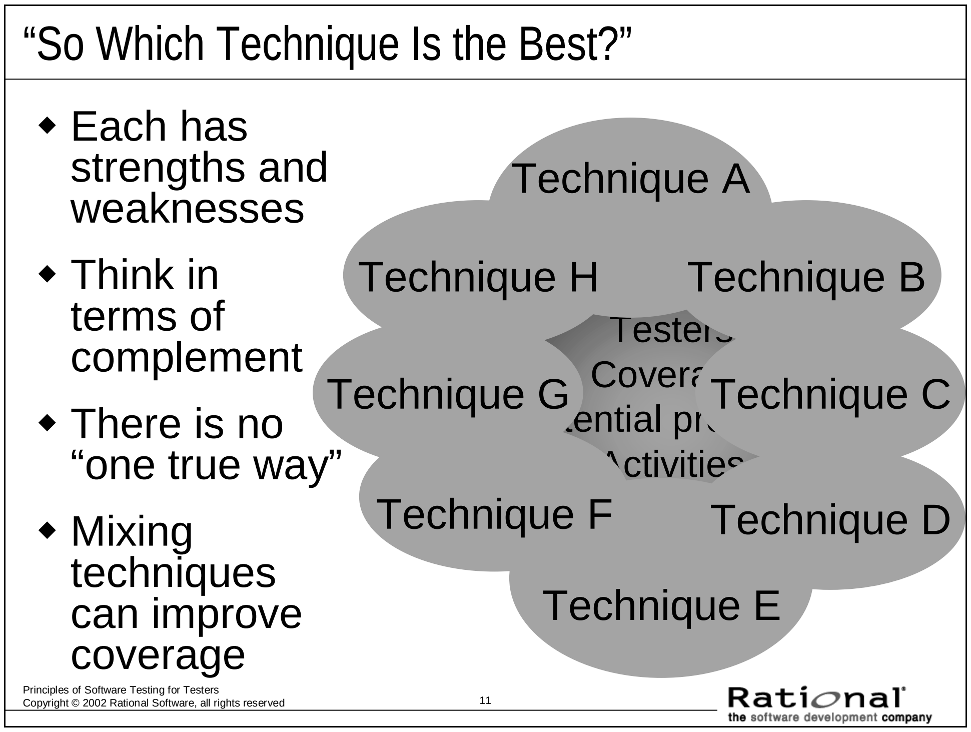
Rational the software development company

No one uses all of these techniques. Some companies focus primarily on one of them (different ones for different companies). This is too narrow—problems that are easy to find under one technique are much harder to find under some others.

We'll walk through a selection of techniques, trying to get a sense of what it's like to analyze a system through the eyes of a tester who focuses on one or another of these techniques.

You might be tempted to try to add several of these approaches to your company's repertoire at the same time. That may not be wise. You might be better off adding one technique, getting good at it, and then adding the next. Many highly effective groups focus on a few of these approaches, perhaps four, rather than trying to be excellent with all of them.

“So Which Technique Is the Best?”



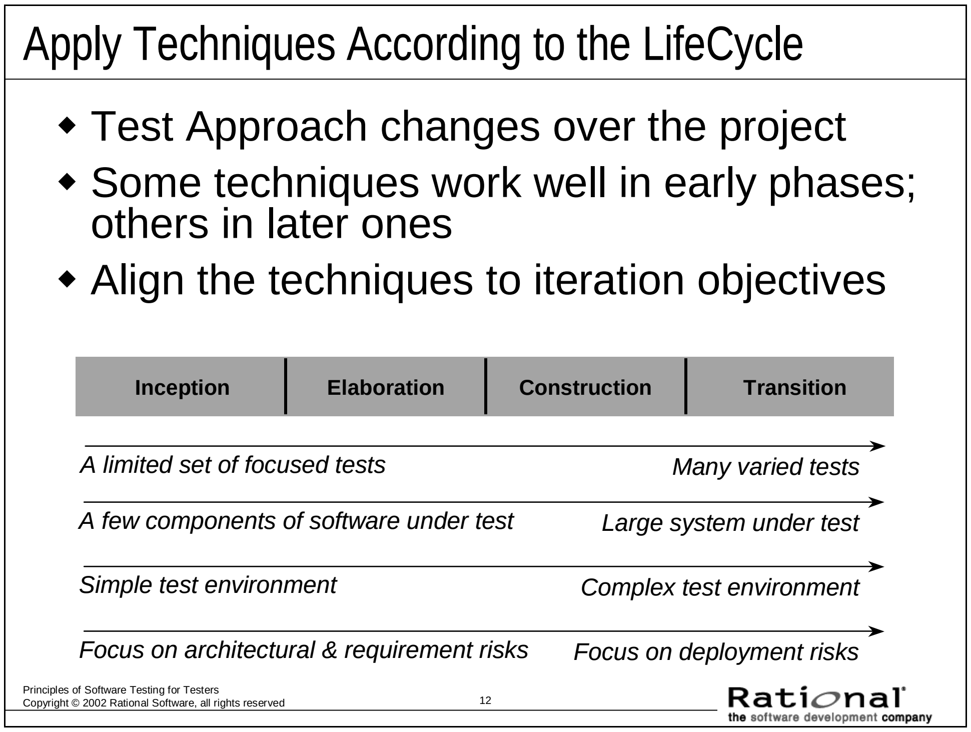
“So Which Technique Is the Best?”

- Each has strengths and weaknesses
- Think in terms of complement
- There is no “one true way”
- Mixing techniques can improve coverage

Technique A
Technique H
Technique B
Testers
Covera
Technique G
Technique C
ential pr
ctivities
Technique F
Technique D
Technique E

11

Apply Techniques According to the LifeCycle



Apply Techniques According to the LifeCycle

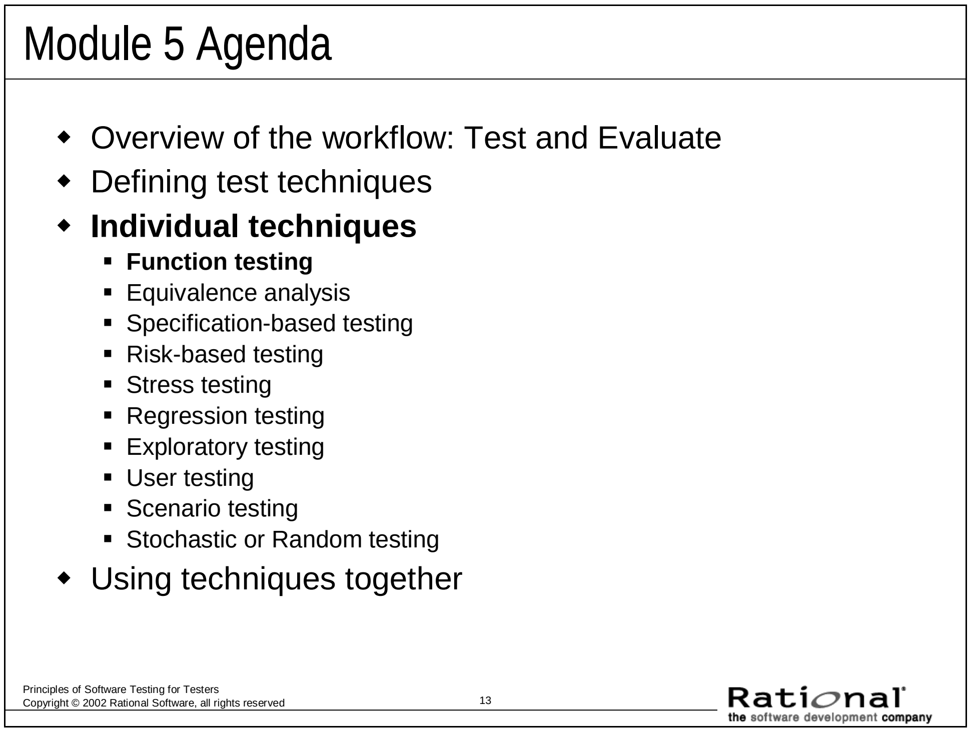
- Test Approach changes over the project
- Some techniques work well in early phases; others in later ones
- Align the techniques to iteration objectives

Inception
Elaboration
Construction
Transition
A limited set of focused tests
Many varied tests
A few components of software under test
Large system under test
Simple test environment
Complex test environment
Focus on architectural & requirement risks
Focus on deployment risks

In Module 3, we introduced the concept of RUP phases and iterations within the phases. In considering and planning test techniques for an iteration, it is important to look at the techniques according to several characteristics.

The techniques that are appropriate in early iterations may lose their effectiveness in later iterations, when the software under test is more robust. Similarly, techniques that are useful in late iterations may be inefficient if applied too early.

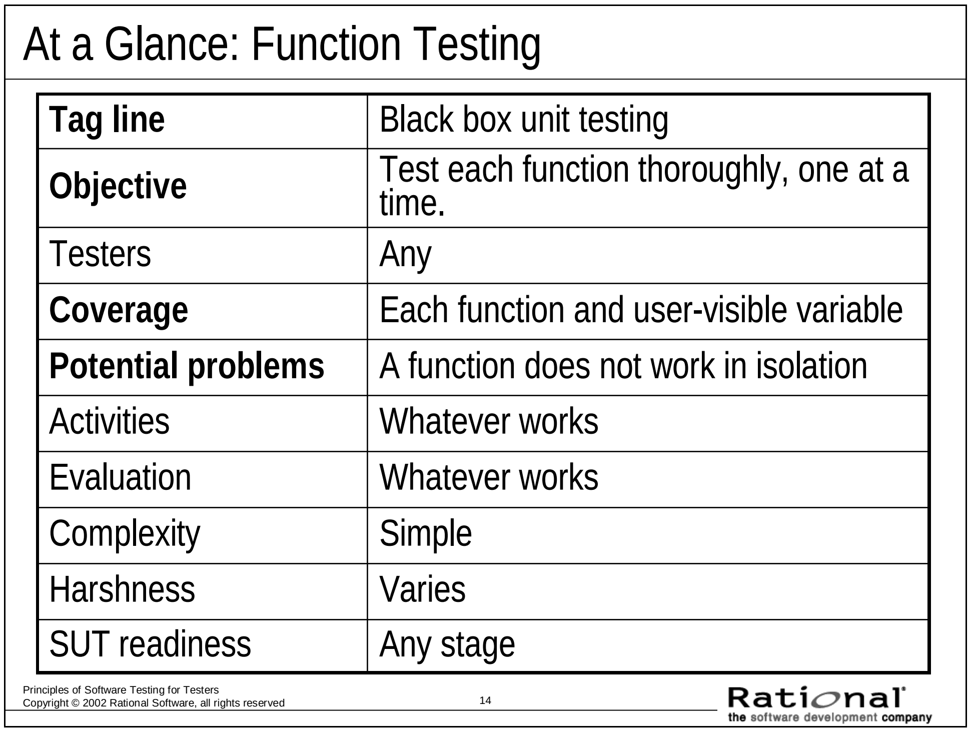
Individual Techniques



Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - **Function testing**
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

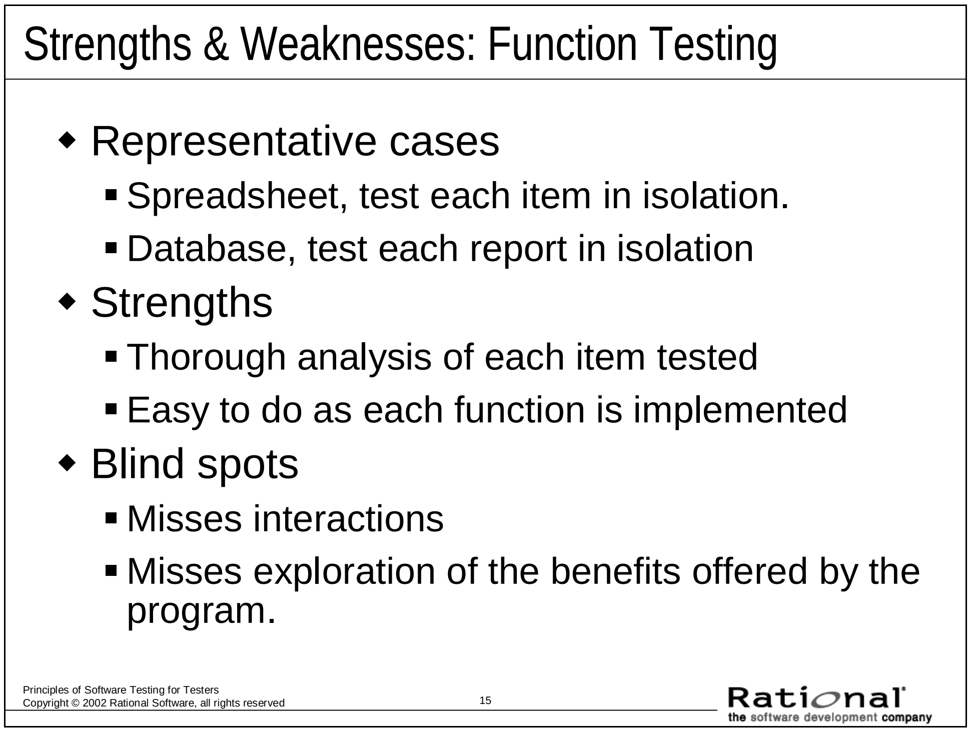
At a Glance: Function Testing



At a Glance: Function Testing

Tag line	Black box unit testing
Objective	Test each function thoroughly, one at a time.
Testers	Any
Coverage	Each function and user-visible variable
Potential problems	A function does not work in isolation
Activities	Whatever works
Evaluation	Whatever works
Complexity	Simple
Harshness	Varies
SUT readiness	Any stage

Strengths & Weaknesses: Function Testing



Strengths & Weaknesses: Function Testing

- Representative cases
  - Spreadsheet, test each item in isolation.
  - Database, test each report in isolation
- Strengths
  - Thorough analysis of each item tested
  - Easy to do as each function is implemented
- Blind spots
  - Misses interactions
  - Misses exploration of the benefits offered by the program.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 15

Rational the software development company

Some function testing tasks:

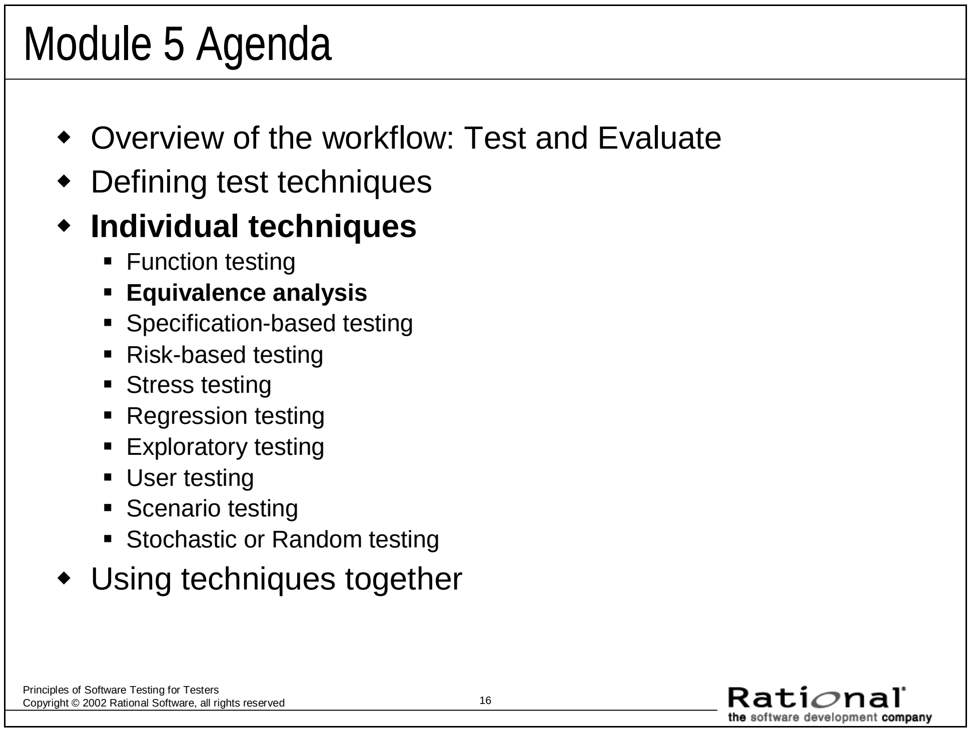
- Identify the program's features / commands
  - From specifications or the draft user manual
  - From walking through the user interface
  - From trying commands at the command line
  - From searching the program or resource files for command names
- Identify variables used by the functions and test their boundaries.
- Identify environmental variables that may constrain the function under test.
- Use each function in a mainstream way (positive testing) and push it in as many ways as possible, as hard as possible.

  Many companies use a function testing approach early in testing, to check whether the basic functionality of the program is present and reasonably stable.

Take Home Exercise (~1 Hour)

1. Agree on a familiar part of a familiar program for everyone to use (e.g. the Bullets and Numbering command in MS Word).
2. Break into pairs, with one computer per pair.
3. Go through the function testing tasks above and make notes.
4. Photocopy your notes, share with other teams and discuss.

Module 5 Agenda

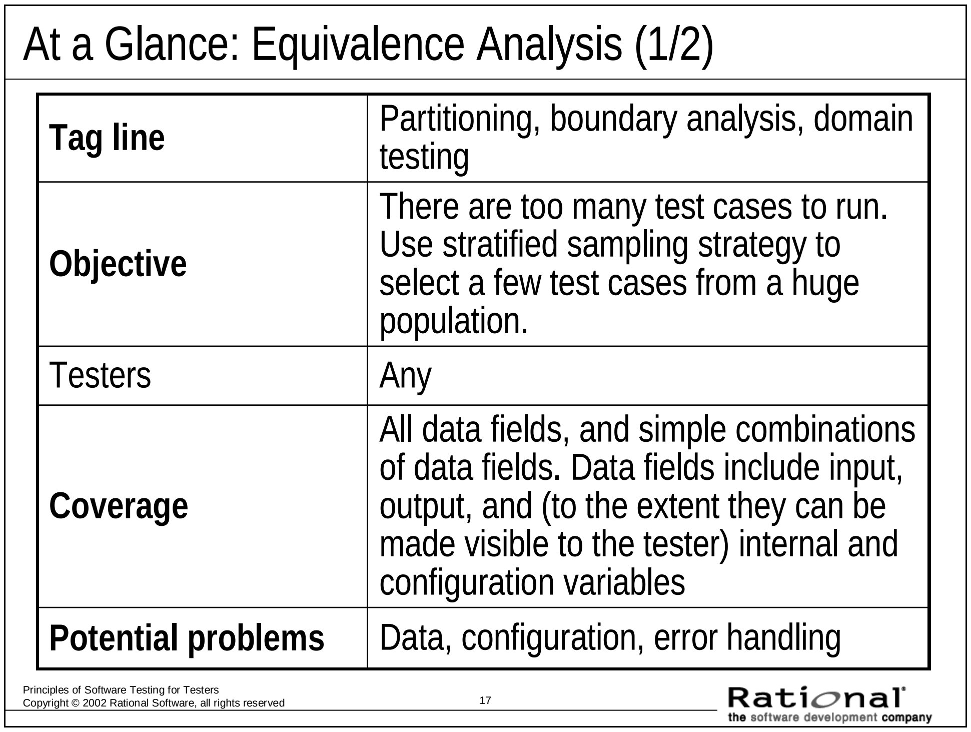


Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - **Equivalence analysis**
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

16

At a Glance: Equivalence Analysis (1/2)



At a Glance: Equivalence Analysis (1/2)

Tag line	Partitioning, boundary analysis, domain testing
Objective	There are too many test cases to run. Use stratified sampling strategy to select a few test cases from a huge population.
Testers	Any
Coverage	All data fields, and simple combinations of data fields. Data fields include input, output, and (to the extent they can be made visible to the tester) internal and configuration variables
Potential problems	Data, configuration, error handling

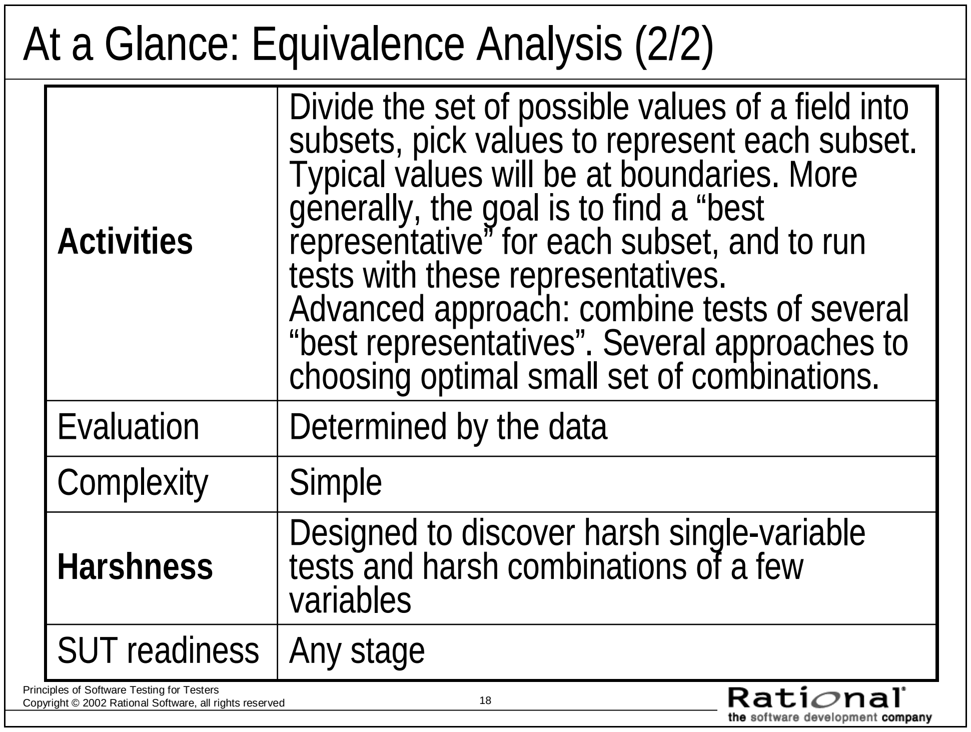
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 17

Glenford J. Myers described equivalence analysis in The Art of Software Testing (1979). It is an essential technique in the arsenal of virtually every professional tester.

To quote from RUP:

> **Equivalence partitioning** is a technique for reducing the required number of tests. For every operation, you should identify the equivalence classes of the arguments and the object states. An **equivalence class** is a set of values for which an object is supposed to behave similarly. For example, a **Set** has three equivalence classes: **empty**, **some element**, and **full**.

At a Glance: Equivalence Analysis (2/2)



At a Glance: Equivalence Analysis (2/2)

Activities	Divide the set of possible values of a field into subsets, pick values to represent each subset. Typical values will be at boundaries. More generally, the goal is to find a "best representative" for each subset, and to run tests with these representatives. Advanced approach: combine tests of several "best representatives". Several approaches to choosing optimal small set of combinations.
Evaluation	Determined by the data
Complexity	Simple
Harshness	Designed to discover harsh single-variable tests and harsh combinations of a few variables
SUT readiness	Any stage

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 18

Rational the software development company

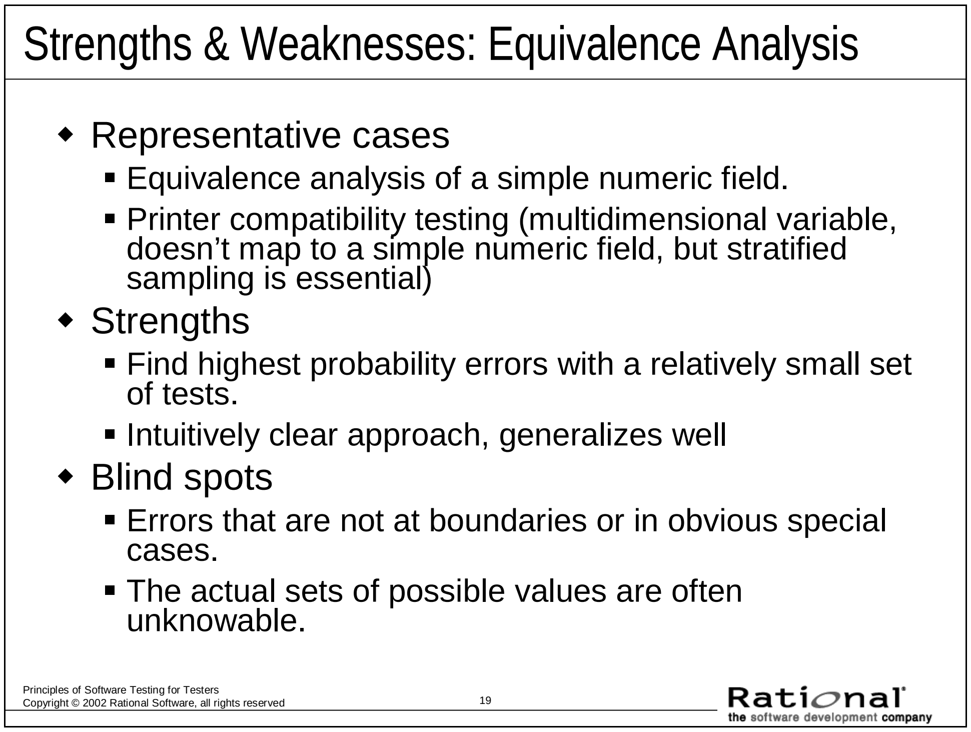
Prof. Kaner draws this comparison:

Public opinion polls like Gallup apply the method of **stratified sampling**. Pollsters can call up 2000 people across the US and predict with some accuracy the results of the election. It's not a random sample. They subdivide the population into equivalence classes. It's not just people who make lots of money, people who make a fair amount of money, people who don't make quite as much, and people who really should make a lot more. That's one dimension, but we also have where people live, what their gender is, what their age is, what their race is, and what kind of car they drive as other variables. But we end up picking somebody who is a point on many different places – this kind of car, that age, and so forth, and we say they represent a bunch of other people who have this kind of car or this kind of income group, and so forth. What you want as a representative -- the best representative from the point of view of pollsters -- is the most typical representative, the one who would vote the way most of them would vote.

They're dividing the world 3 or 4 or 5 dimensionally, but they still end up with equivalence classes. And then they call up their list of 2000 great representatives and weight them according to how often that subgroup fits into the population and then predict on what these folks say what the whole subgroup would do. They actually take more than one representative from each subgroup just in case.

That's called stratified sampling. You divide your population into different strata, into different layers, and you make sure you sample from each one. We're doing stratified sampling when we do equivalence class analysis. These strata are just equivalence classes. The core difference between testing and Gallup-poll-type sampling is that, when we pick somebody in this case, we're not looking for the test case that is most like everybody else, *we're looking for the one most likely to show a failure*.

Strengths & Weaknesses: Equivalence Analysis



Strengths & Weaknesses: Equivalence Analysis

- Representative cases
  - Equivalence analysis of a simple numeric field.
  - Printer compatibility testing (multidimensional variable, doesn't map to a simple numeric field, but stratified sampling is essential)
- 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.
  - The actual sets of possible values are often unknowable.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 19

Rational the software development company

Some of the Key Tasks

If you wanted to practice your domain testing skills, here are things that you would practice:

- Partitioning into equivalence classes
- Discovering best representatives of the sub-classes
- Combining tests of several fields
- Create boundary charts
- Find fields / variables / environmental conditions
- Identify constraints (non-independence) in the relationships among variables.

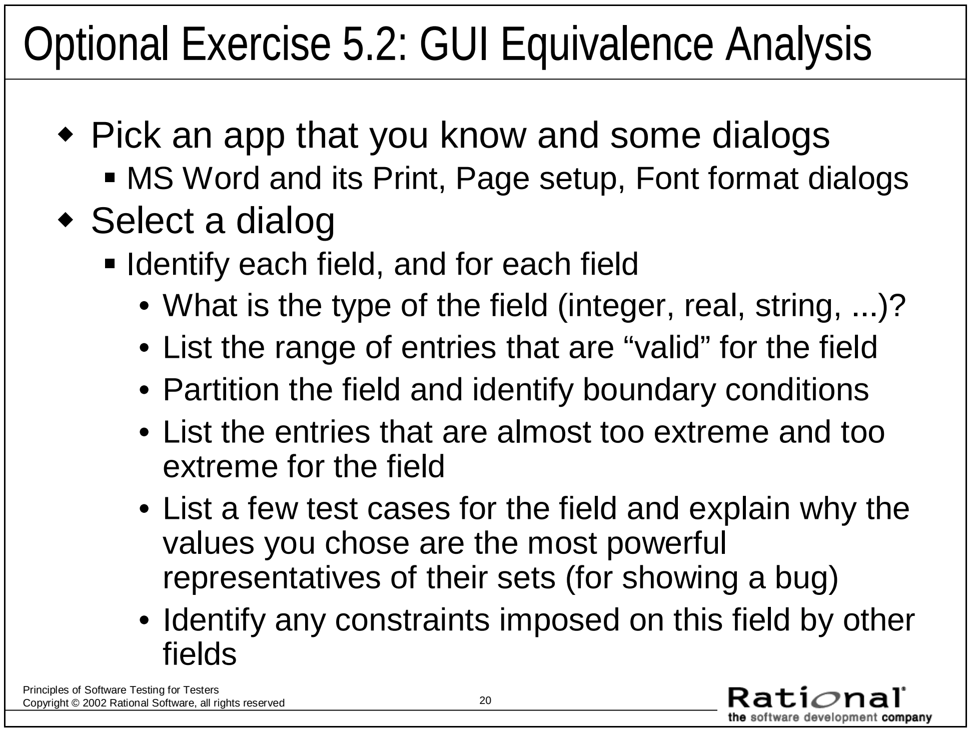
Ideas for Exercises

- Find the biggest / smallest accepted value in a field
- Find the biggest / smallest value that fits in a field
- Partition fields
- Read specifications to determine the actual boundaries
- Create boundary charts for several variables
- Create standard domain testing charts for different types of variables
- For finding variables, see notes on function testing

Further reading

The classic issue with Equivalence Analysis is combinatorial explosion – you get too many test cases. One technique worth learning for reducing the combinations is All Pairs. See *Lessons Learned*, pp. 52-58.

Optional Exercise 5.2: GUI Equivalence Analysis



Optional Exercise 5.2: GUI Equivalence Analysis

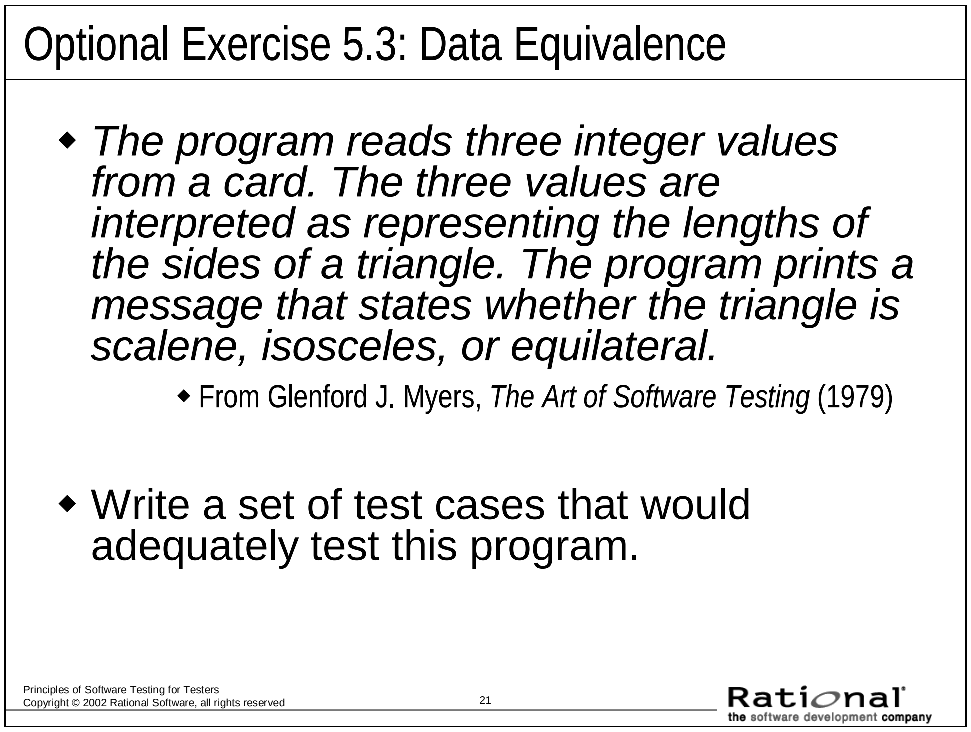
- Pick an app that you know and some dialogs
  - MS Word and its Print, Page setup, Font format dialogs
- Select a dialog
  - Identify each field, and for each field
    - What is the type of the field (integer, real, string, ...)?
    - List the range of entries that are “valid” for the field
    - Partition the field and identify boundary conditions
    - List the entries that are almost too extreme and too extreme for the field
    - List a few test cases for the field and explain why the values you chose are the most powerful representatives of their sets (for showing a bug)
    - Identify any constraints imposed on this field by other fields

Principles of Software Testing for Testers

20

Optional Exercise

Optional Exercise 5.3: Data Equivalence



Optional Exercise 5.3: Data Equivalence

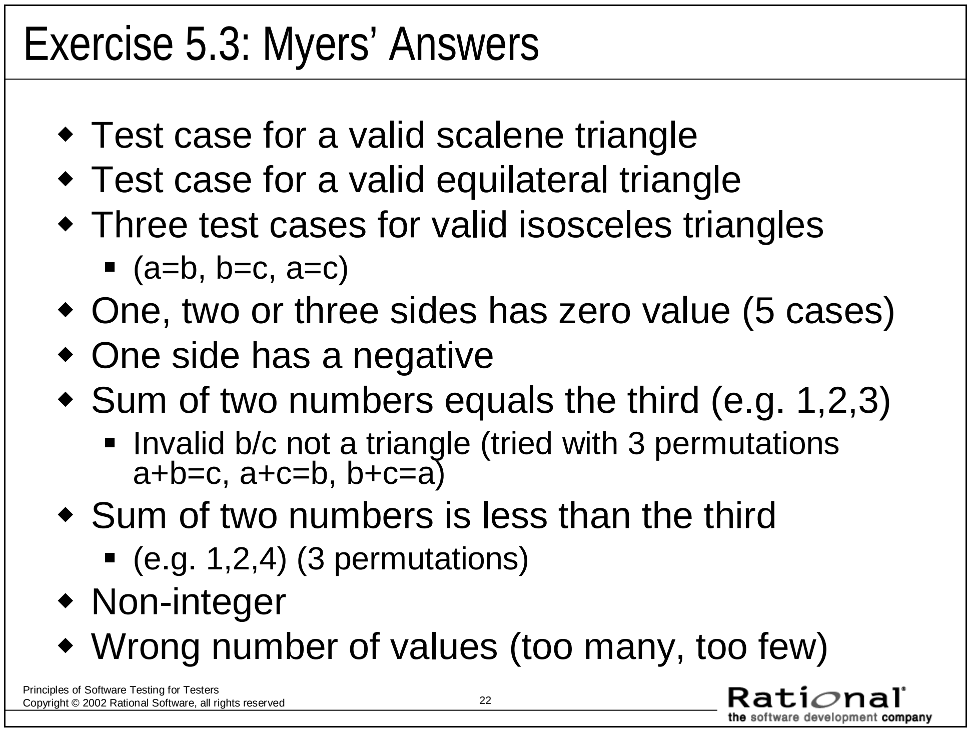
- *The program reads three integer values from a card. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.*
  - From Glenford J. Myers, *The Art of Software Testing* (1979)
- Write a set of test cases that would adequately test this program.

Optional Exercise

Myers' Triangle is probably the best known example of an equivalence problem. It is typical of the cases one would examine for pure data analysis.

It is also characteristic of the analysis you would do for API testing, where a function takes a certain number of arguments and issues a return value.

Exercise 5.3: Myers' Answers



Exercise 5.3: Myers' Answers

- Test case for a valid scalene triangle
- Test case for a valid equilateral triangle
- Three test cases for valid isosceles triangles
  - (a=b, b=c, a=c)
- One, two or three sides has zero value (5 cases)
- One side has a negative
- Sum of two numbers equals the third (e.g. 1,2,3)
  - Invalid b/c not a triangle (tried with 3 permutations a+b=c, a+c=b, b+c=a)
- Sum of two numbers is less than the third
  - (e.g. 1,2,4) (3 permutations)
- Non-integer
- Wrong number of values (too many, too few)

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 22

List 10 tests that you'd run that aren't in Myers' list:

1.

2.

3.

4.

5.

6.

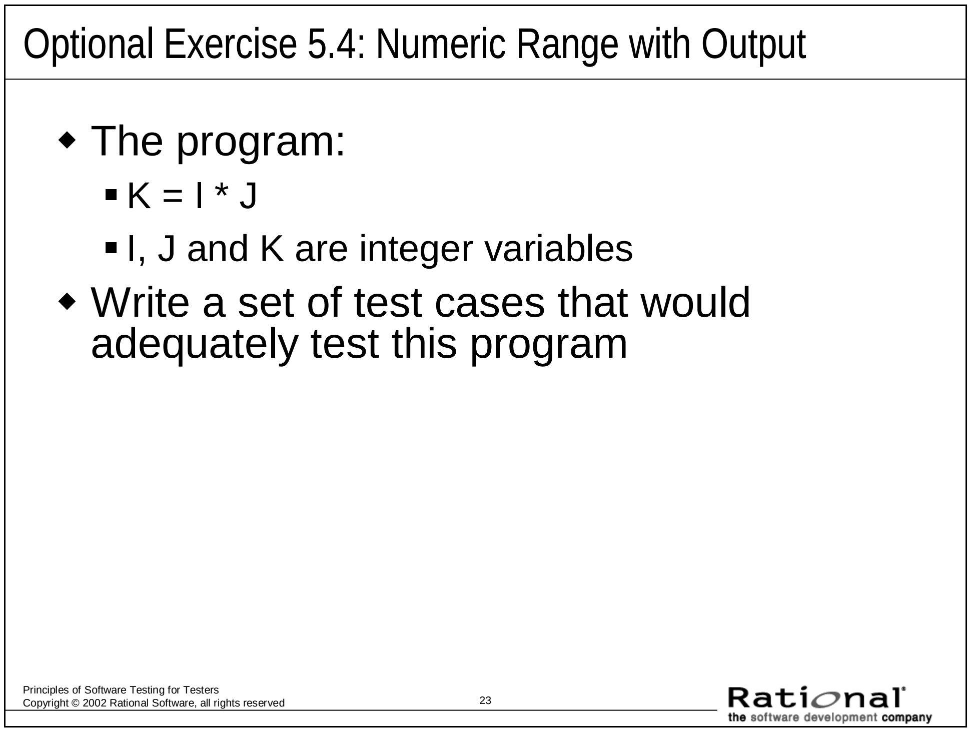
7.

8.

9.

10.

Optional Exercise 5.4: Numeric Range with Output



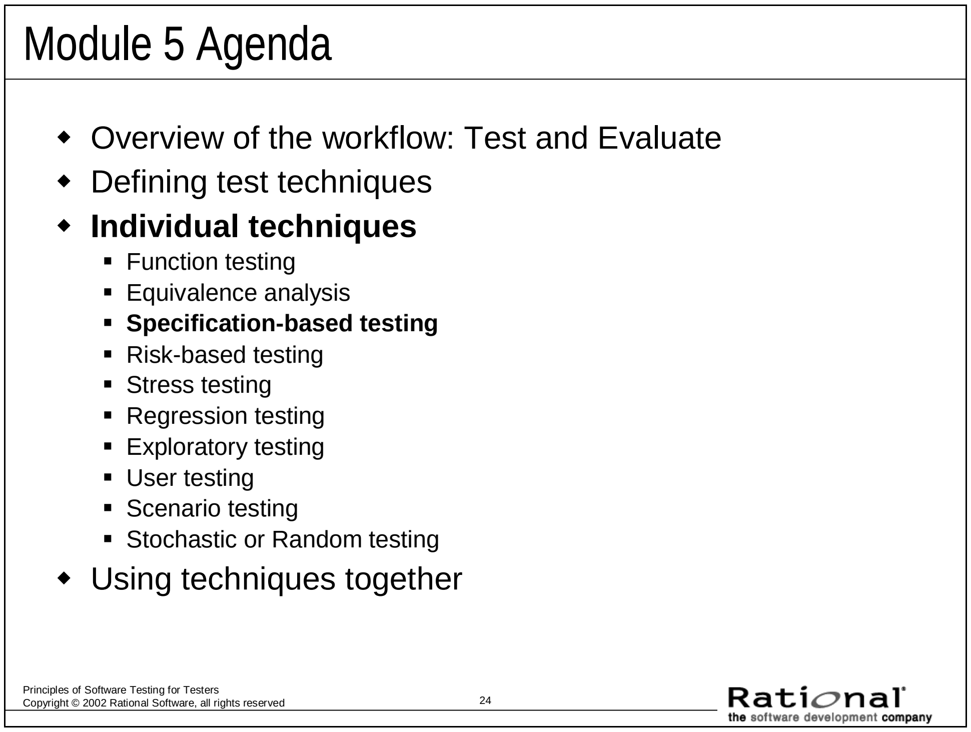
Optional Exercise 5.4: Numeric Range with Output

- The program:
  - K = I * J
  - I, J and K are integer variables
- Write a set of test cases that would adequately test this program

Optional Exercise

This is a typical case with a broad range of values with issues of data type. It is applicable for testing at the GUI or the API.

Module 5 Agenda



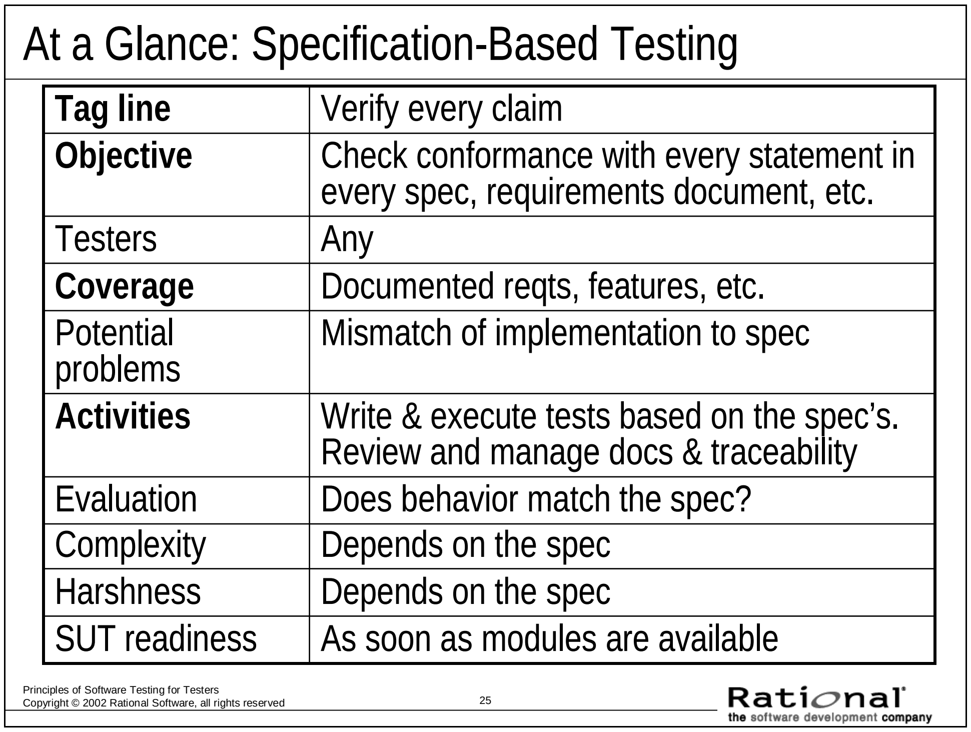
Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - **Specification-based testing**
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

Principles of Software Testing for Testers

24

At a Glance: Specification-Based Testing



At a Glance: Specification-Based Testing

Tag line	Verify every claim
Objective	Check conformance with every statement in every spec, requirements document, etc.
Testers	Any
Coverage	Documented reqts, features, etc.
Potential problems	Mismatch of implementation to spec
Activities	Write & execute tests based on the spec's. Review and manage docs & traceability
Evaluation	Does behavior match the spec?
Complexity	Depends on the spec
Harshness	Depends on the spec
SUT readiness	As soon as modules are available

Rational the software development company

Common Tasks in Spec-Driven Testing

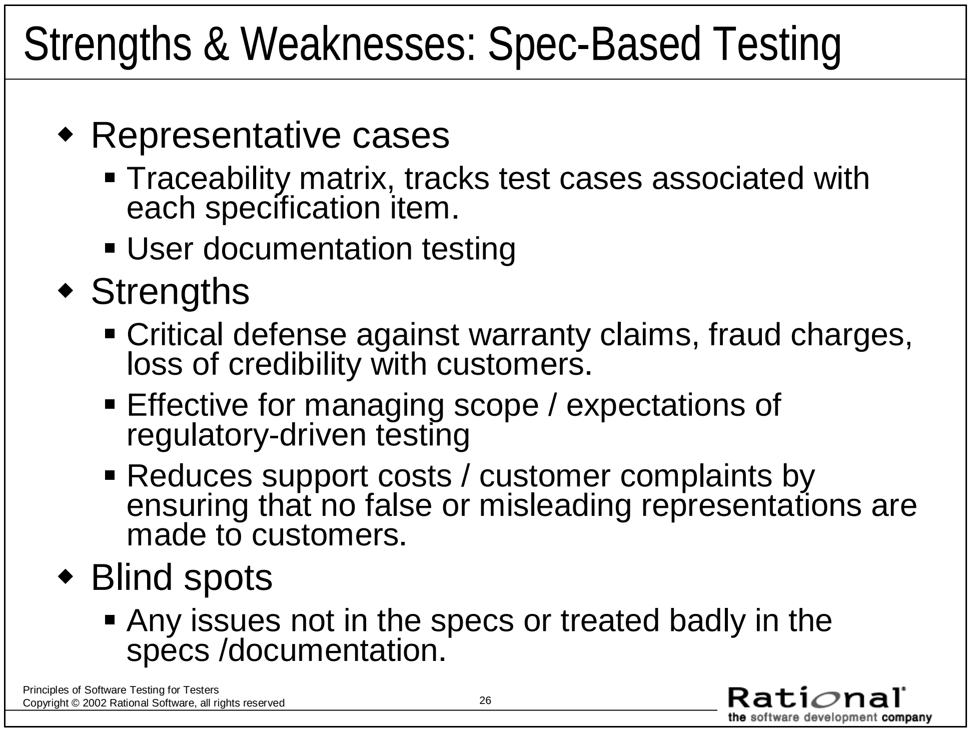
- Review specifications for
  - Ambiguity
  - Adequacy (it covers the issues)
  - Correctness (it describes the program)
  - Content (not a source of design errors)
  - Testability support
- Create traceability matrices
- Document management (spec versions, file comparison utilities for comparing two spec versions, etc.)
- Participate in review meetings

Ideas for Mixing Techniques

Medical device and software makers provide an interesting example of a mixed strategy involving specification-based testing. The Food and Drug Administration requires that there be tests for every claim made about the product. Those tests are normally documented in full detail, and often automated.

However, this is a minimum set, not the level of testing most companies use. Even if the product meets FDA standards, it may be unsafe. The company will therefore run many additional tests, often exploratory. These don't have to be reported to the FDA unless they expose defects. (In which case, the tests are probably added to the regression test suite.)

Strengths & Weaknesses: Spec-Based Testing



Strengths & Weaknesses: Spec-Based Testing

- Representative cases
  - Traceability matrix, tracks test cases associated with each specification item.
  - User documentation 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.

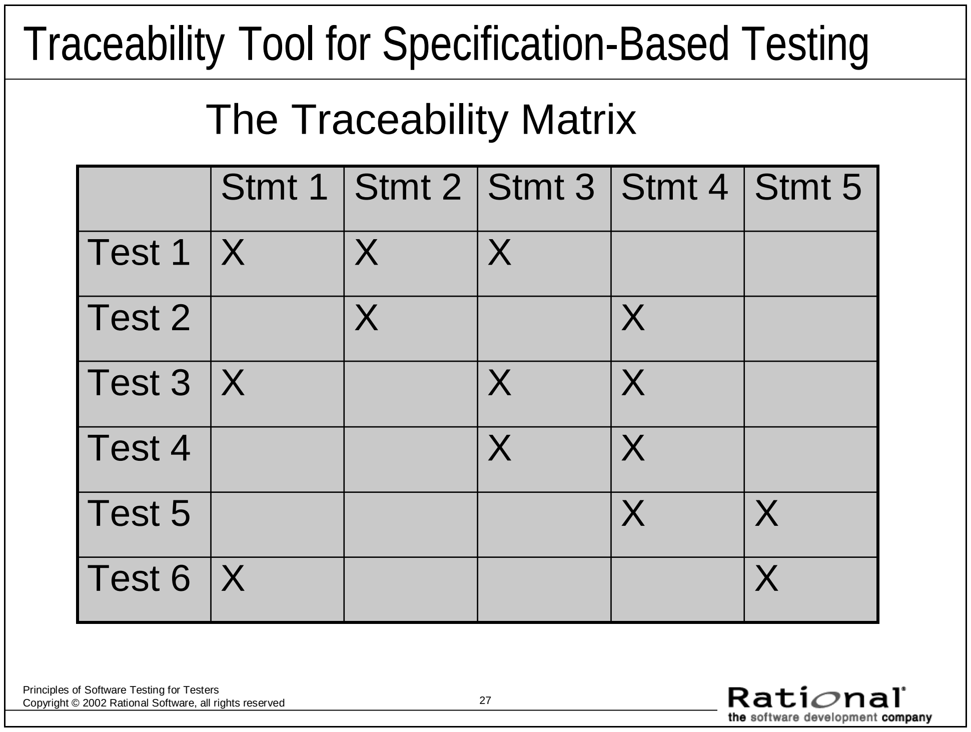
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 26

Rational the software development company

Some of the Skills Involved in Spec-Based Testing

- Understand the level of generality called for when testing a spec item. For example, imagine a field X:
  - We could test a single use of X
  - Or we could partition possible values of X and test boundary values
  - Or we could test X in various scenarios
  - Which is the right one?
- Ambiguity analysis
  - Richard Bender teaches this well. If you can't take his course, you can find notes based on his work in Rodney Wilson's *Software RX: Secrets of Engineering Quality Software*
  - Another book provides an excellent introduction to the ways in which statements can be ambiguous and provides lots of sample exercises: Cecile Cyrul Spector, *Saying One Thing, Meaning Another : Activities for Clarifying Ambiguous Language*

Traceability Tool for Specification-Based Testing



Traceability Tool for Specification-Based Testing

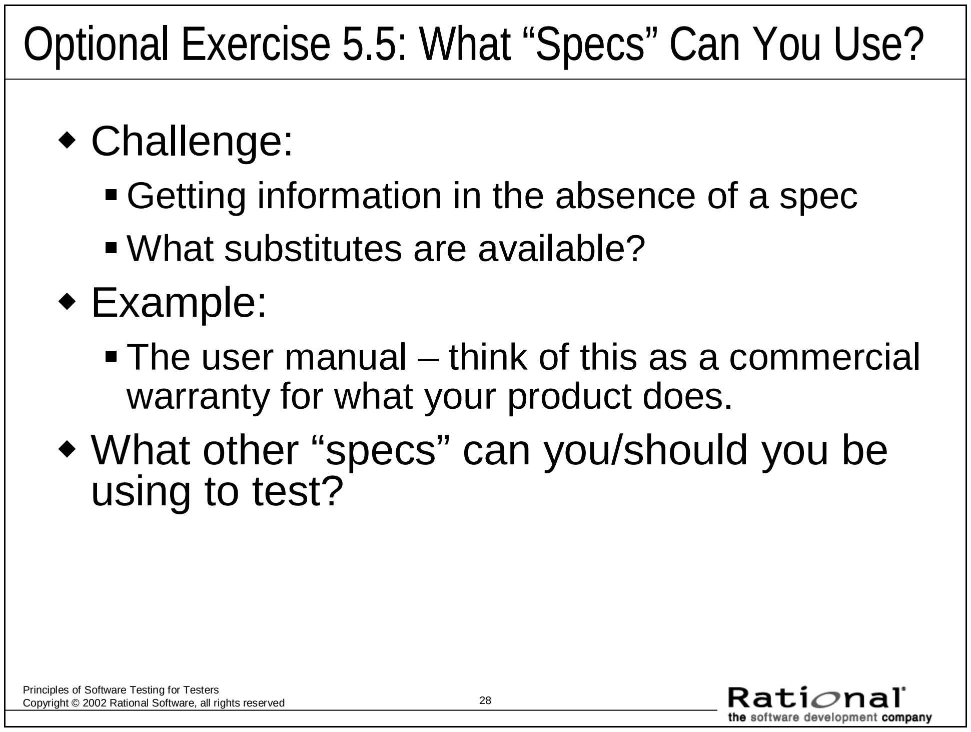
The Traceability Matrix

	Stmt 1	Stmt 2	Stmt 3	Stmt 4	Stmt 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

The traceability matrix is a useful chart for showing what variables (or functions or specification items) are covered by what tests.

- The columns can show any type of test item, such as 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*.

Optional Exercise 5.5: What "Specs" Can You Use?



Optional Exercise 5.5: What "Specs" Can You Use?

- Challenge:
  - Getting information in the absence of a spec
  - What substitutes are available?
- Example:
  - The user manual – think of this as a commercial warranty for what your product does.
- What other "specs" can you/should you be using to test?

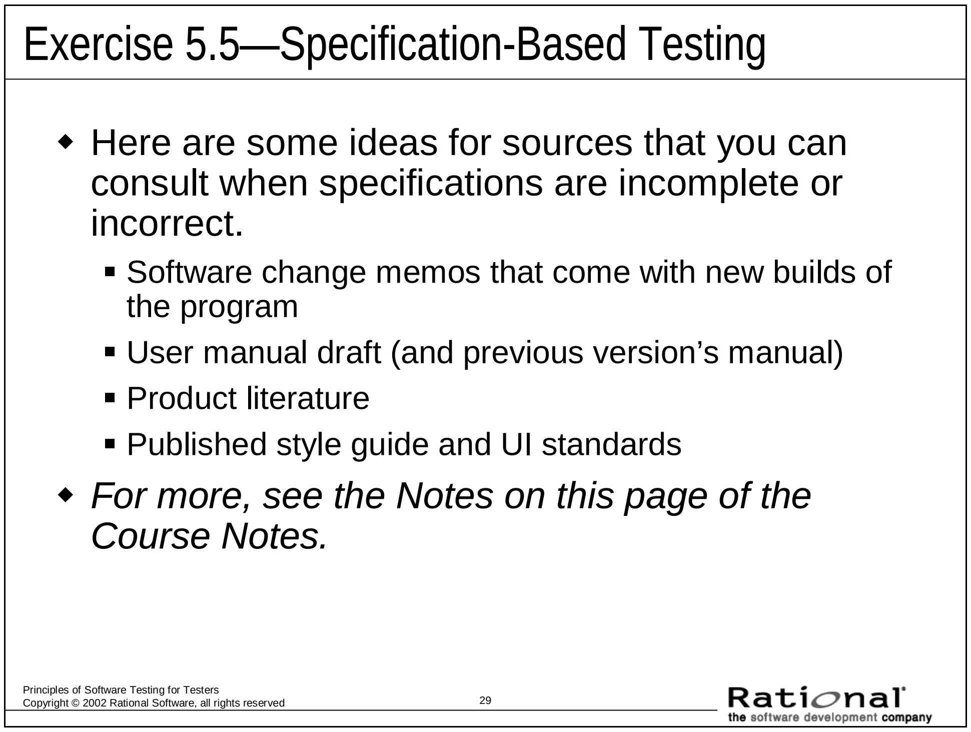
Optional Exercise

Think about standards or expert documents as sources. Imagine you're testing a website. Consider the difference between saying. "I can't navigate…" and saying "This site violates these principles of Jakob Nielsen's *Designing Web Usability…*"

Generally respected texts or standards may not necessarily be for your project, but they are useful.

For example, if you criticize some aspect of the user interface, your criticism might be dismissed as "just your opinion." But if you make the same criticism and then show that this aspect of the UI doesn't conform to a published UI design guidelines document for your platform (there are several books available), the criticism will be taken more seriously. Even if the programmers and marketers don't fix the problem that you identified, they will evaluate your report of the problem as credible and knowledgeable.

Exercise 5.5—Specification-Based Testing



Exercise 5.5—Specification-Based Testing

- Here are some ideas for sources that you can consult when specifications are incomplete or incorrect.
  - Software change memos that come with new builds of the program
  - User manual draft (and previous version's manual)
  - Product literature
  - Published style guide and UI standards
- *For more, see the Notes on this page of the Course Notes.*

Principles of Software Testing for Testers

29

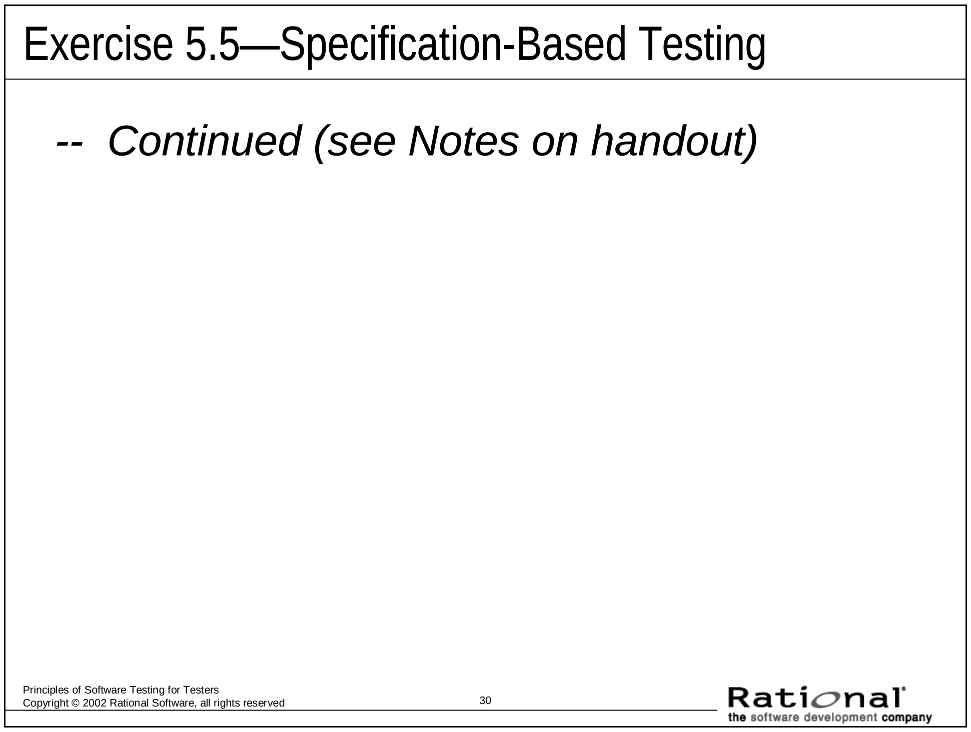
No specification???

Companies vary in the ways they develop software. Even companies that follow the Rational Unified Process will adapt RUP to their needs, and they may not do everything that you might expect them to do.

Some companies write very concise specifications, or very incomplete ones, or they don't update their specs as the project evolves.

Testers have to know how to deal with the project as it is. Sometimes you will be able to influence the fundamental development style of the project, but often, you will have limited influence. In those cases, you still have to know how to do an effective job of testing.

Exercise 5.5—Specification-Based Testing



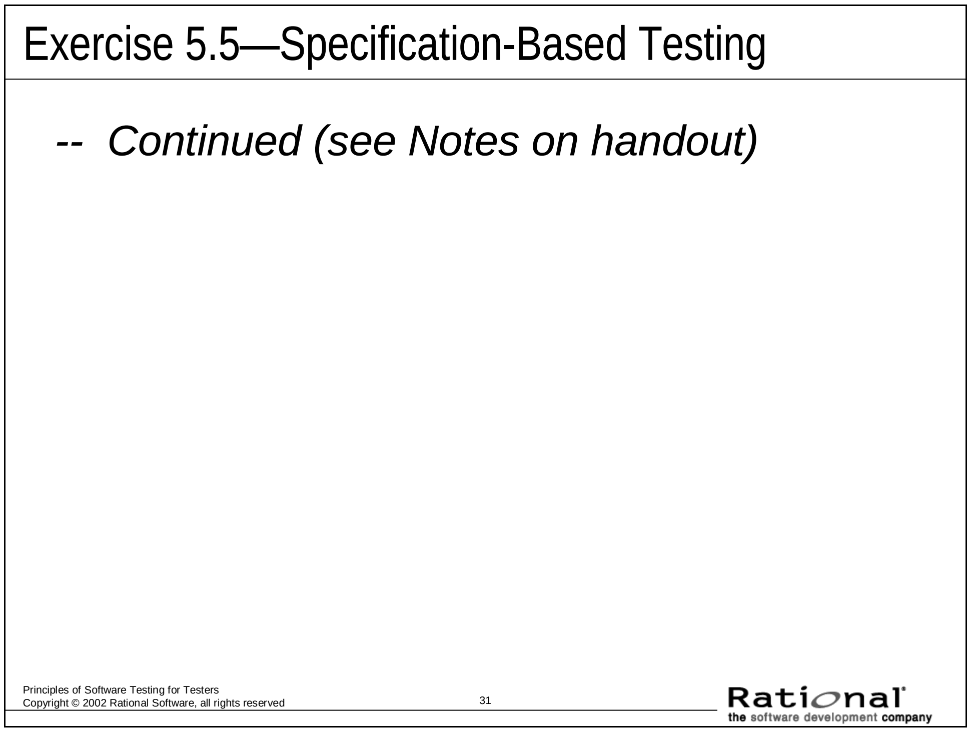
Exercise 5.5—Specification-Based Testing

-- Continued (see Notes on handout)

Sources of information for spec-based testing

- Whatever specs exist
- Software change memos that come with new builds of the program
- User manual draft (and previous version's manual)
- Product literature
- Published style guide and UI standards
- Published standards (such as C-language)
- 3rd party product compatibility test suites
- Published regulations
- Internal memos (e.g. project mgr. to engineers, describing the feature definitions)
- Marketing presentations, selling the concept of the product to management
- Bug reports (responses to them)
- Reverse engineer the program.
- Interview people, such as
    - development lead, tech writer, customer service, subject matter experts, project manager
- Look at header files, source code, database table definitions
- Specs and bug lists for all 3rd party tools that you use

Exercise 5.5—Specification-Based Testing



Exercise 5.5—Specification-Based Testing

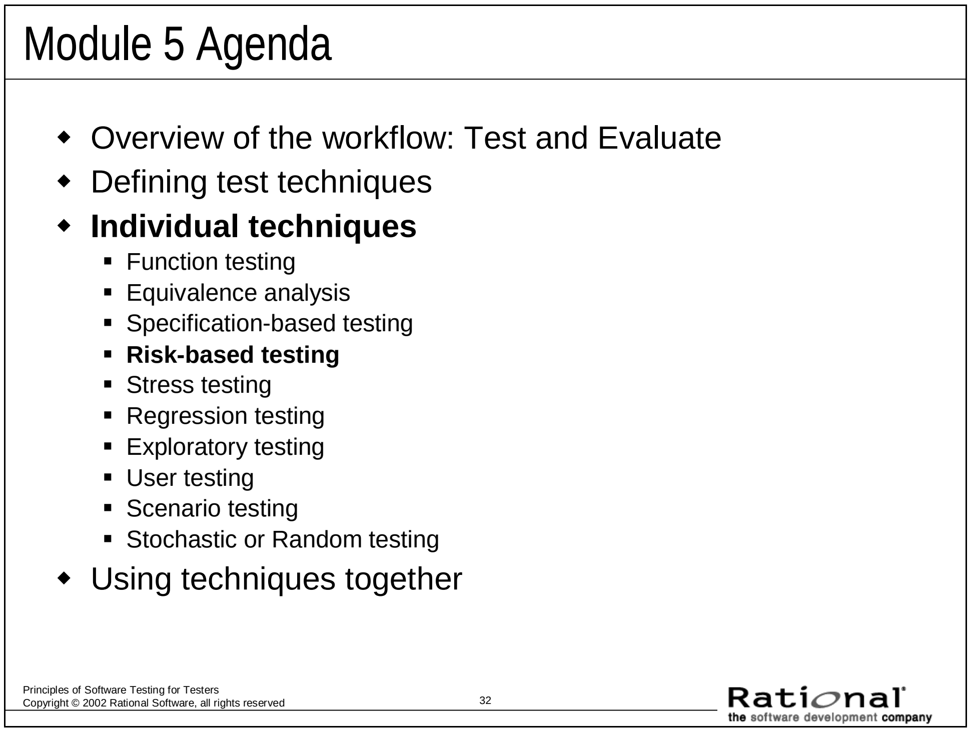
-- Continued (see Notes on handout)

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 31

Sources of information for spec-based testing (continued)

- Prototypes, and lab notes on the prototypes
- Interview development staff from the last version.
- Look at customer call records from the previous version. What bugs were found in the field?
- Usability test results
- Beta test results
- Ziff-Davis SOS CD and other tech support CD's (These are answerbooks sold to help desks), for bugs in your product and common bugs in your niche or on your platform
- BugNet magazine / web site for common bugs, and other bug reporting websites.
- Localization guide (probably one that is published, for localizing products on your platform.)
- Get lists of compatible equipment and environments from Marketing (in theory, at least.)
- Look at compatible products, to find their failures (then look for these in your product), how they designed features that you don't understand, and how they explain their design. See listserv's, NEWS, BugNet, etc.
- Exact comparisons with products you emulate
- Content reference materials (e.g. an atlas to check your on-line geography program)

Module 5 Agenda



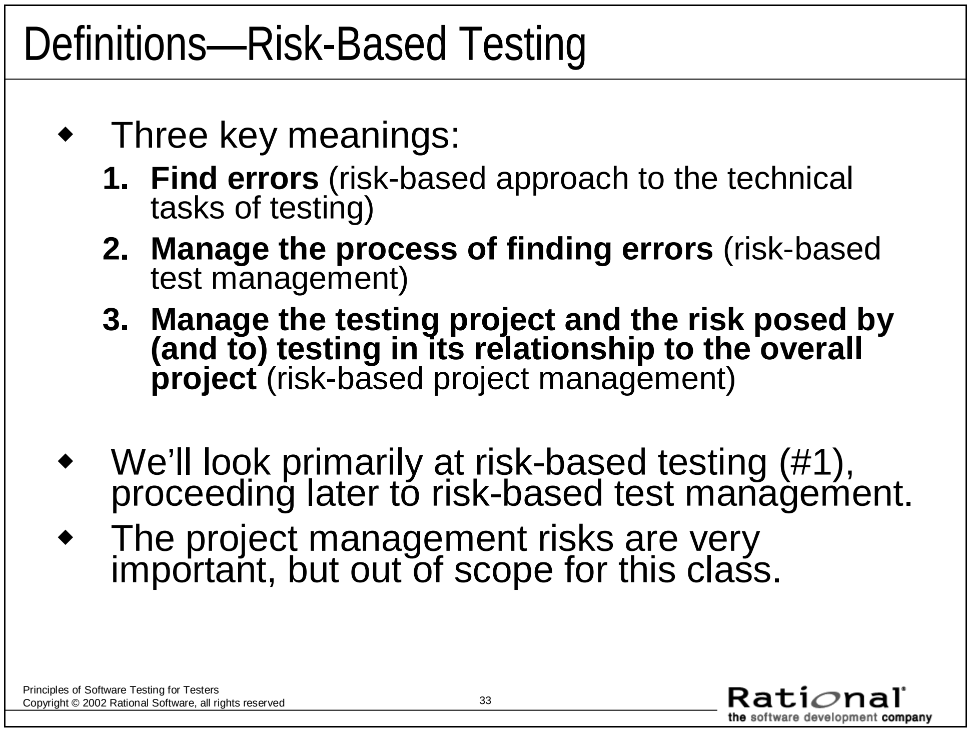
Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - **Risk-based testing**
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 32

Rational the software development company

Definitions—Risk-Based Testing



Definitions—Risk-Based Testing

- Three key meanings:
  1. **Find errors** (risk-based approach to the technical tasks of testing)
  2. **Manage the process of finding errors** (risk-based test management)
  3. **Manage the testing project and the risk posed by (and to) testing in its relationship to the overall project** (risk-based project management)
- We'll look primarily at risk-based testing (#1), proceeding later to risk-based test management.
- The project management risks are very important, but out of scope for this class.

33

Here's an everyday analogy for thinking about risk based testing.

Hazard:

A dangerous condition (something that could trigger an accident)

Risk:

Possibility of suffering loss or harm (probability of an accident caused by a given hazard).

Accident:

A hazard is encountered, resulting in loss or harm.

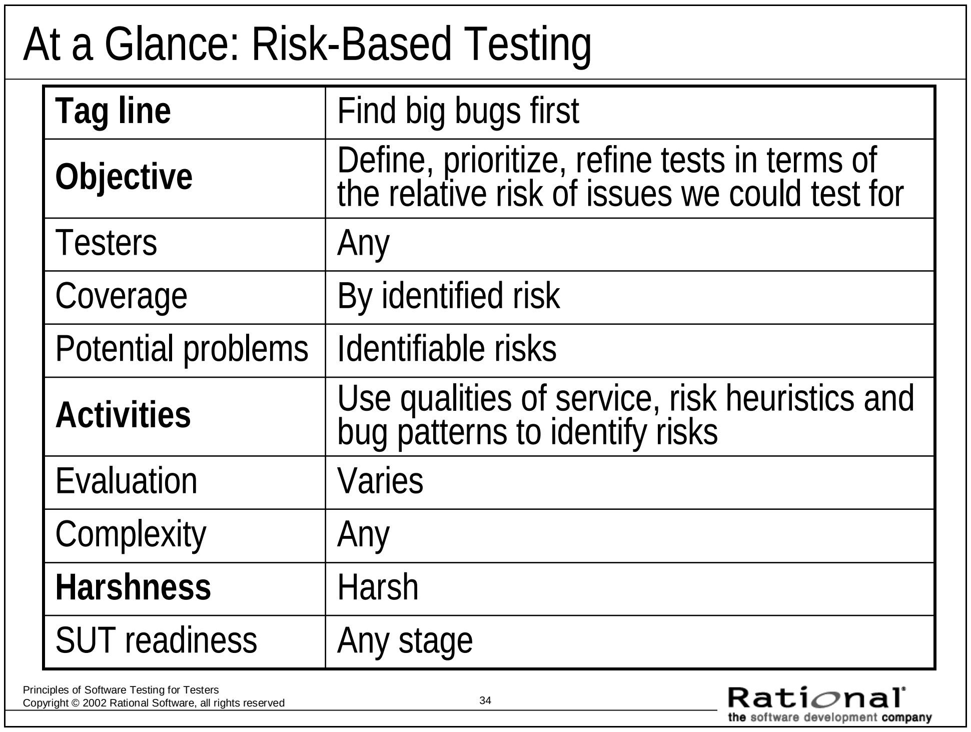
A term that is sometimes used for this is FMEA – Failure Mode Effects Analysis. In FMEA, you start with a list of the ways that a product could fail. These are the failure modes. Next you ask what the effects of the failure could be. Based on that analysis, you decide how to focus your testing and what problems to look for.

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. (We gain confidence if it passes our best tests). Subjecting a program to easy tests doesn't tell us much about what will happen to the program in the field.

In risk-based testing, we create harsh tests for vulnerable areas of the program.

At a Glance: Risk-Based Testing



At a Glance: Risk-Based Testing

Tag line	Find big bugs first
Objective	Define, prioritize, refine tests in terms of the relative risk of issues we could test for
Testers	Any
Coverage	By identified risk
Potential problems	Identifiable risks
Activities	Use qualities of service, risk heuristics and bug patterns to identify risks
Evaluation	Varies
Complexity	Any
Harshness	Harsh
SUT readiness	Any stage

Principles of Software Testing for Testers
34

Rational the software development company

Examples of Risk-Based Testing Tasks

- Identify risk factors (hazards: ways in which the program could go wrong)
- For each risk factor, create tests that have power against it.
- Assess coverage of the testing effort program, given a set of risk-based tests. Find holes in the testing effort.
- Build lists of bug histories, configuration problems, tech support requests and obvious customer confusions.
- Evaluate a series of tests to determine what risk they are testing for and whether more powerful variants can be created.

Here's one way: **Risk-Based Equivalence Class Analysis**

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:

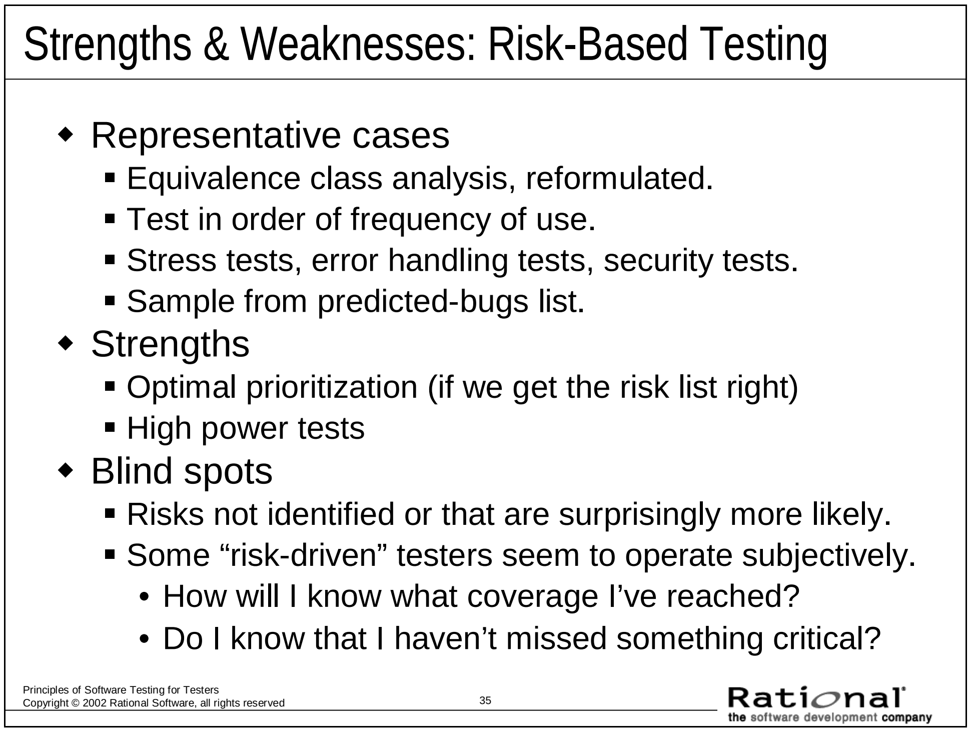
- *Two test cases can only be equivalent by reference to a specifiable risk.*

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.

Strengths & Weaknesses: Risk-Based Testing



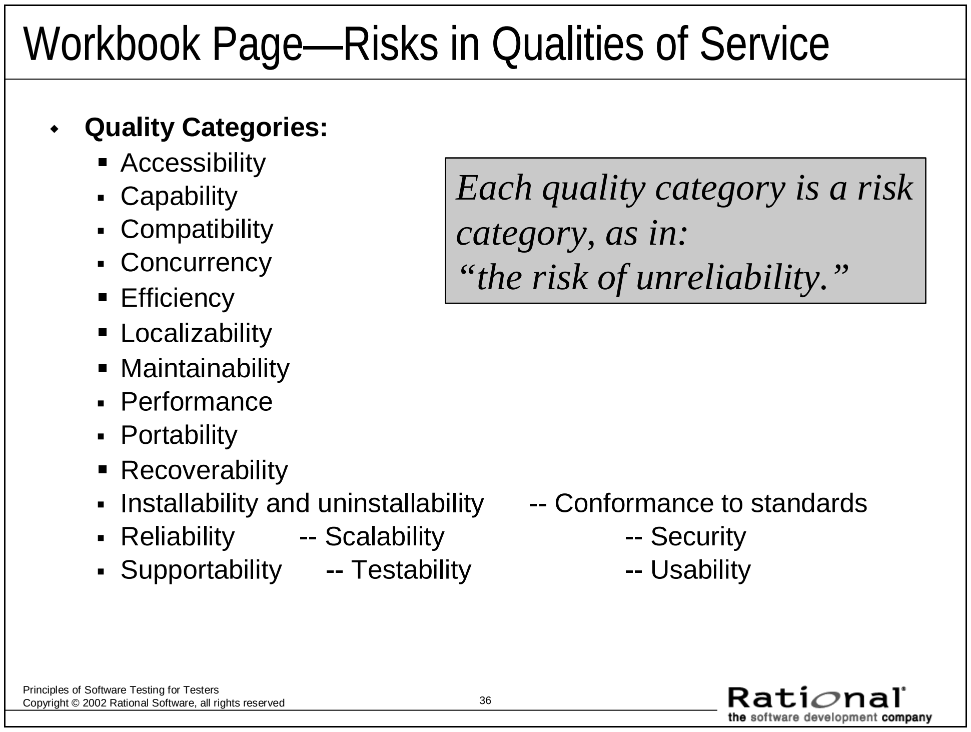
Strengths & Weaknesses: Risk-Based Testing

- Representative cases
  - Equivalence class analysis, reformulated.
  - Test in order of frequency of use.
  - Stress tests, error handling tests, security tests.
  - Sample from predicted-bugs list.
- Strengths
  - Optimal prioritization (if we get the risk list right)
  - High power tests
- Blind spots
  - Risks not identified or that are surprisingly more likely.
  - Some "risk-driven" testers seem to operate subjectively.
    - How will I know what coverage I've reached?
    - Do I know that I haven't missed something critical?

Principles of Software Testing for Testers
35

Rational the software development company

Workbook Page—Risks in Qualities of Service



Workbook Page—Risks in Qualities of Service

- **Quality Categories:**
  - Accessibility
  - Capability
  - Compatibility
  - Concurrency
  - Efficiency
  - Localizability
  - Maintainability
  - Performance
  - Portability
  - Recoverability
  - Installability and uninstallability -- Conformance to standards
  - Reliability -- Scalability -- Security
  - Supportability -- Testability -- Usability

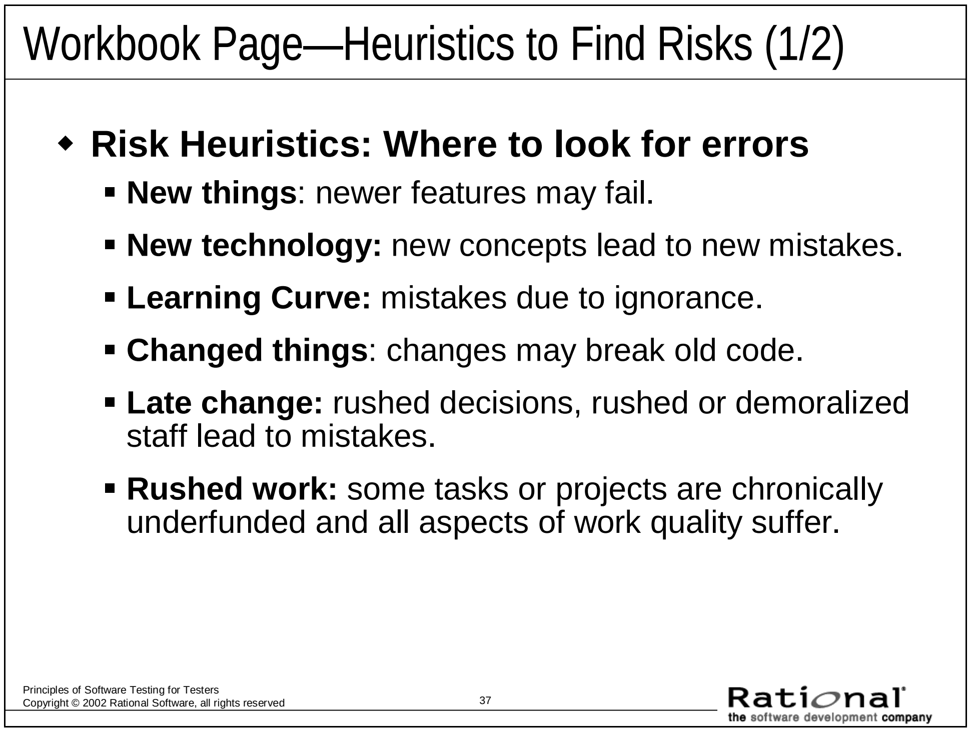
Each quality category is a risk category, as in: "the risk of unreliability."

Take-Home Exercises

The intent of this list of exercises is to illustrate the thinking that risk-based testers use. You can do these at work, after the course either alone or, preferably, in pairs with another tester.

- List ways that the program could fail. For each case:
  - Describe two ways to test for that possible failure
  - Explain how to make your tests more powerful against that type of possible failure
  - Explain why your test is powerful against that hazard.
- Given a list of test cases
  - Identify a hazard that the test case might have power against
  - Explain why this test is powerful against that hazard.
- Collect or create some test cases for the software under test. Make a variety of tests:
  - Mainstream tests that use the program in "safe" ways
  - Boundary tests
  - Scenario tests
  - Wandering walks through the program
  - If possible, use tests the students have suggested previously.
- For each test, ask:
  - How will this test find a defect?
  - What kind of defect did the test author probably 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?

Workbook Page—Heuristics to Find Risks (1/2)



Workbook Page—Heuristics to Find Risks (1/2)

- **Risk Heuristics: Where to look for errors**
  - **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.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 37

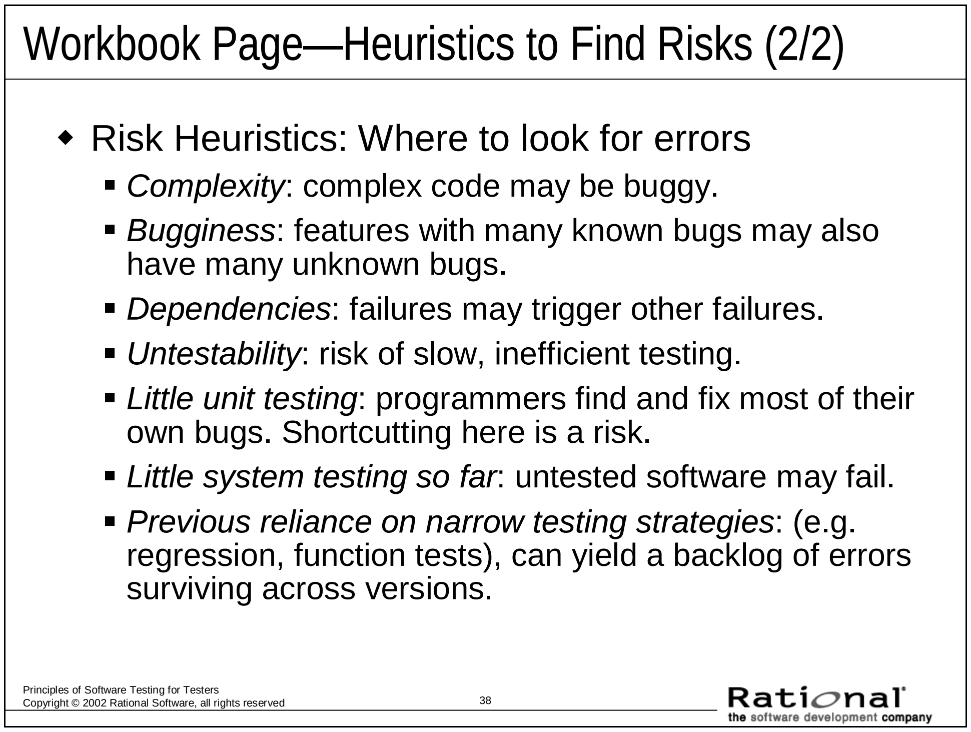
Rational
the software development company

Here are some more risk heuristics to consider:

- *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.:* (Not invented here) external components can cause problems.
- *N.I.B.:* (Not in budget) Unbudgeted tasks may be done shoddily.
- *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.

These heuristics are adapted from a course developed by James Bach, and reprinted in *Lessons Learned*, p. 61-62.

Workbook Page—Heuristics to Find Risks (2/2)



Workbook Page—Heuristics to Find Risks (2/2)

- Risk Heuristics: 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.

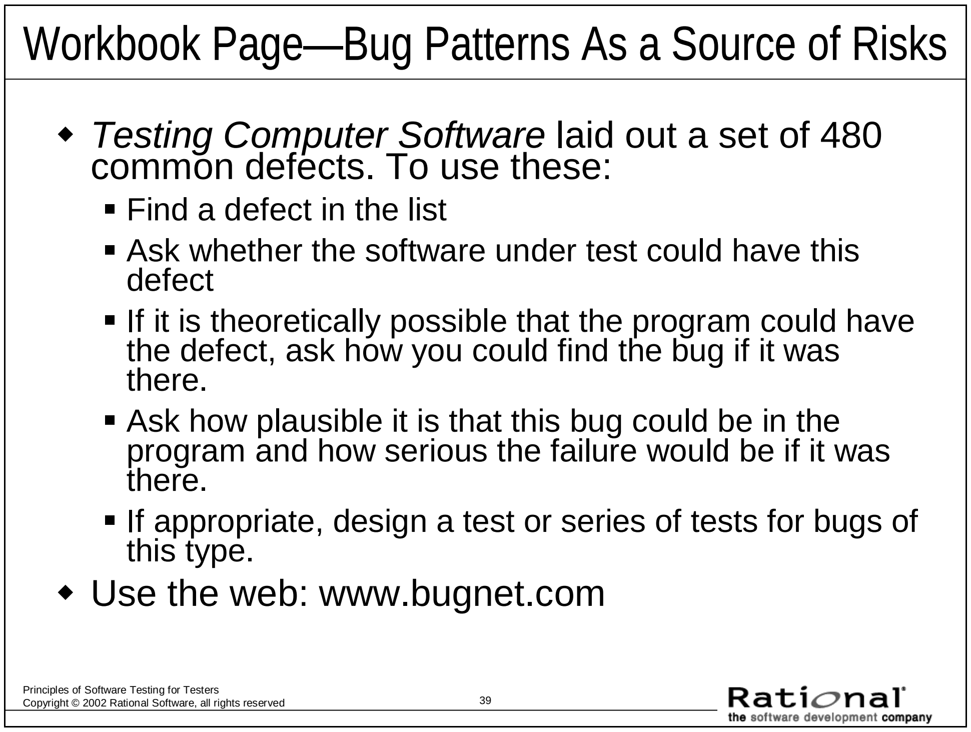
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 38

Rational the software development company

more risk heuristics (continued):

- *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/)
- *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*
- *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.

Workbook Page—Bug Patterns As a Source of Risks



Workbook Page—Bug Patterns As a Source of Risks

- *Testing Computer Software* laid out a set of 480 common defects. To use these:
  - 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.
- Use the web: www.bugnet.com

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 39

Rational
the software development company

Prof. Kaner, senior author of *Testing Computer Software*, says:

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 and further discussion from Hung Nguyen (co-author of *Testing Computer Software*):

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.

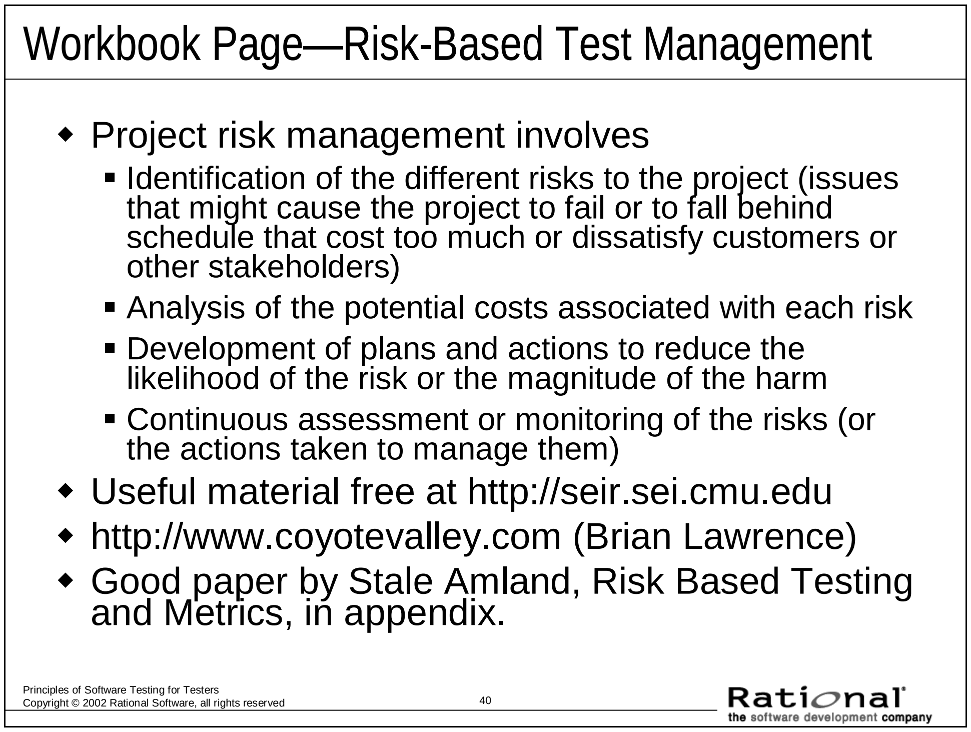
Client 1 and 2 both want to enter a name and client 1 and 2 both use the same new name. Both instances of the name are new relative to their local compare list and therefore, they are accepted, and we now have two instances of the same name.

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.

There are plenty of sources to check for common failures in the common platforms, such as www.bugnet.com and www.cnet.com

Workbook Page—Risk-Based Test Management



Workbook Page—Risk-Based Test Management

- 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 that cost too much or 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)
- Useful material free at http://seir.sei.cmu.edu
- http://www.coyotevalley.com (Brian Lawrence)
- Good paper by Stale Amland, Risk Based Testing and Metrics, in appendix.

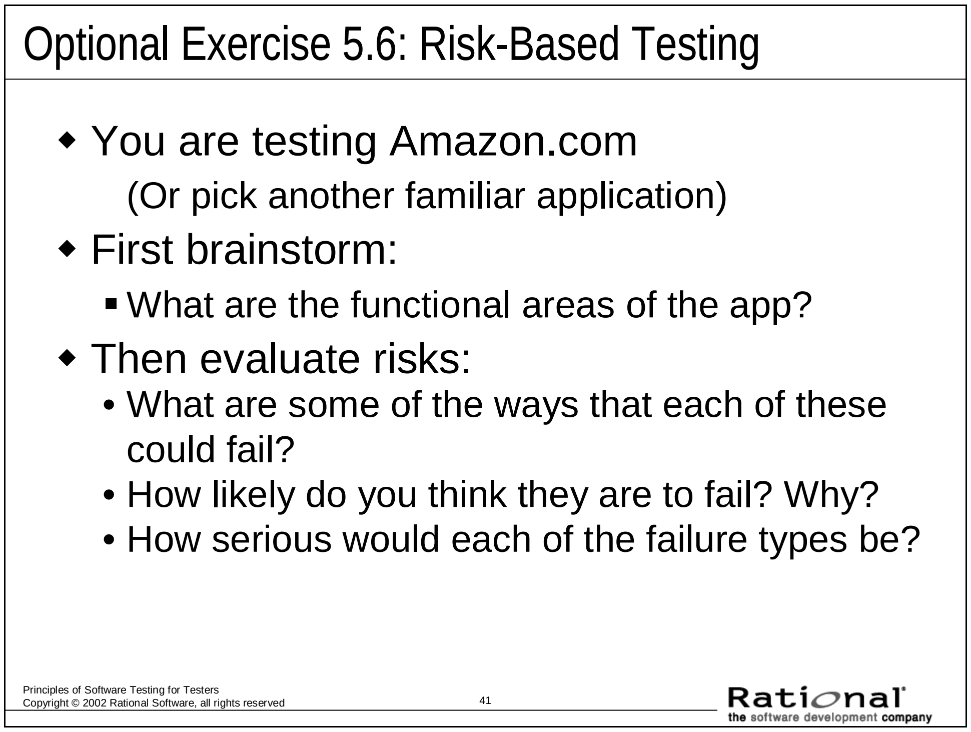
Principles of Software Testing for Testers

40
Rational the software development company

Common Tasks

- List all areas of the program that could require testing
- On a scale of 1-5, assign a probability-of-failure estimate to each
- On a scale of 1-5, assign a severity-of-failure estimate to each
- For each area, identify the specific ways that the program might fail and assign probability-of-failure and severity-of-failure estimates for those
- Prioritize based on estimated risk
- Develop a stop-loss strategy for testing untested or lightly-tested areas, to check whether there is easy-to-find evidence that the areas estimated as low risk are not actually low risk.

Optional Exercise 5.6: Risk-Based Testing



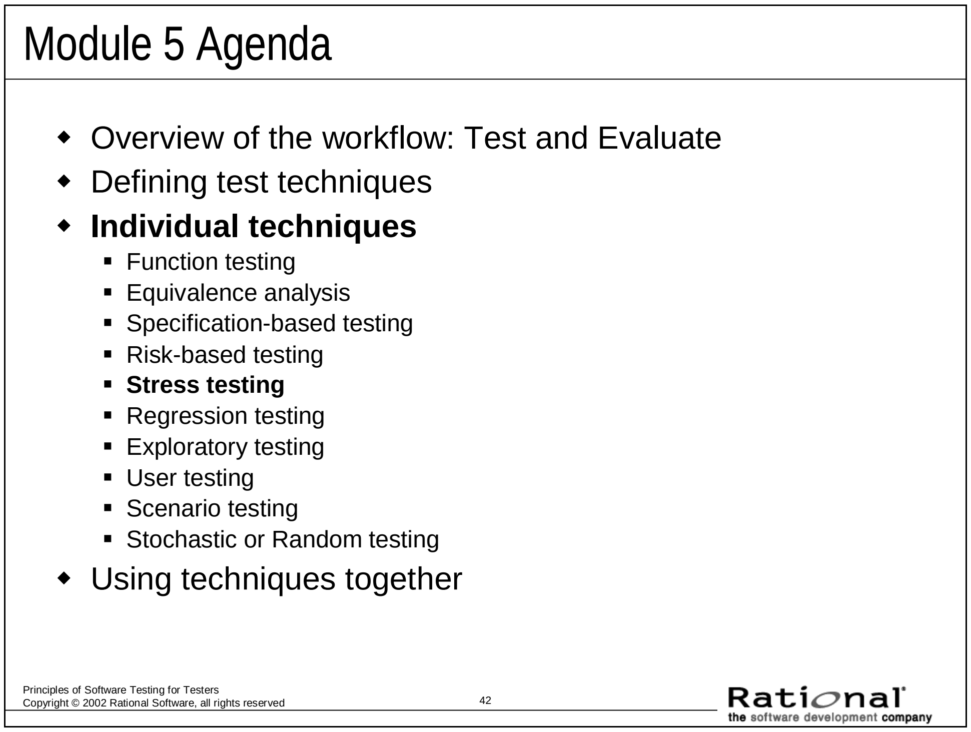
Optional Exercise 5.6: Risk-Based Testing

- You are testing Amazon.com
  (Or pick another familiar application)
- First brainstorm:
  - What are the functional areas of the app?
- Then evaluate risks:
  - What are some of the ways that each of these could fail?
  - How likely do you think they are to fail? Why?
  - How serious would each of the failure types be?

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 41

Optional Exercise

Module 5 Agenda



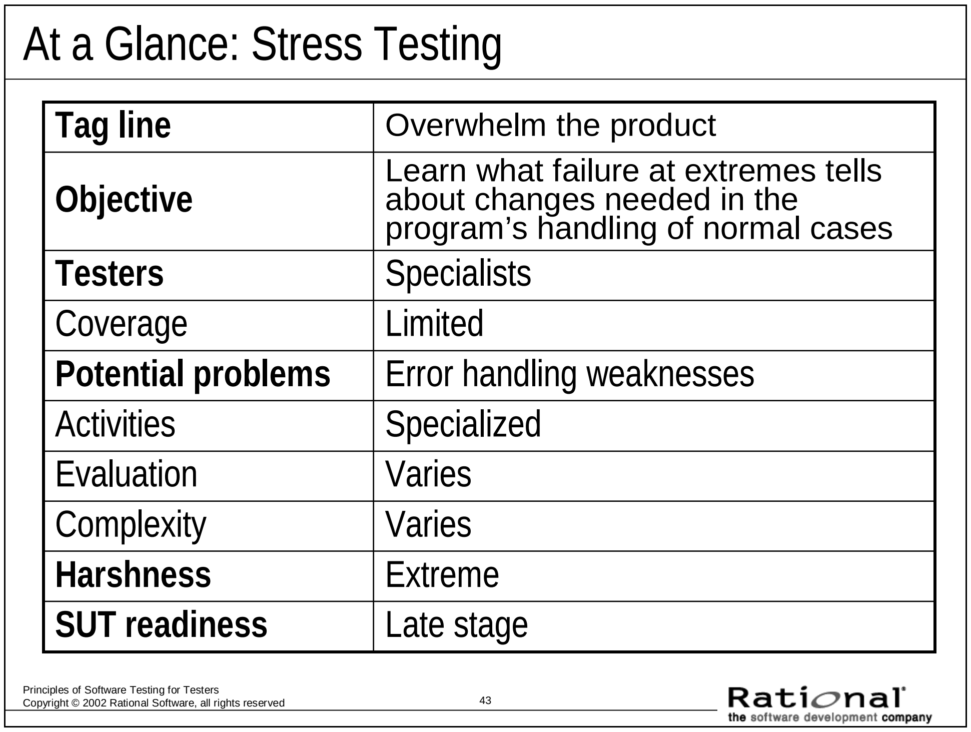
Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - **Stress testing**
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

Principles of Software Testing for Testers

42

At a Glance: Stress Testing



At a Glance: Stress Testing

Tag line	Overwhelm the product
Objective	Learn what failure at extremes tells about changes needed in the program's handling of normal cases
Testers	Specialists
Coverage	Limited
Potential problems	Error handling weaknesses
Activities	Specialized
Evaluation	Varies
Complexity	Varies
Harshness	Extreme
SUT readiness	Late stage

43

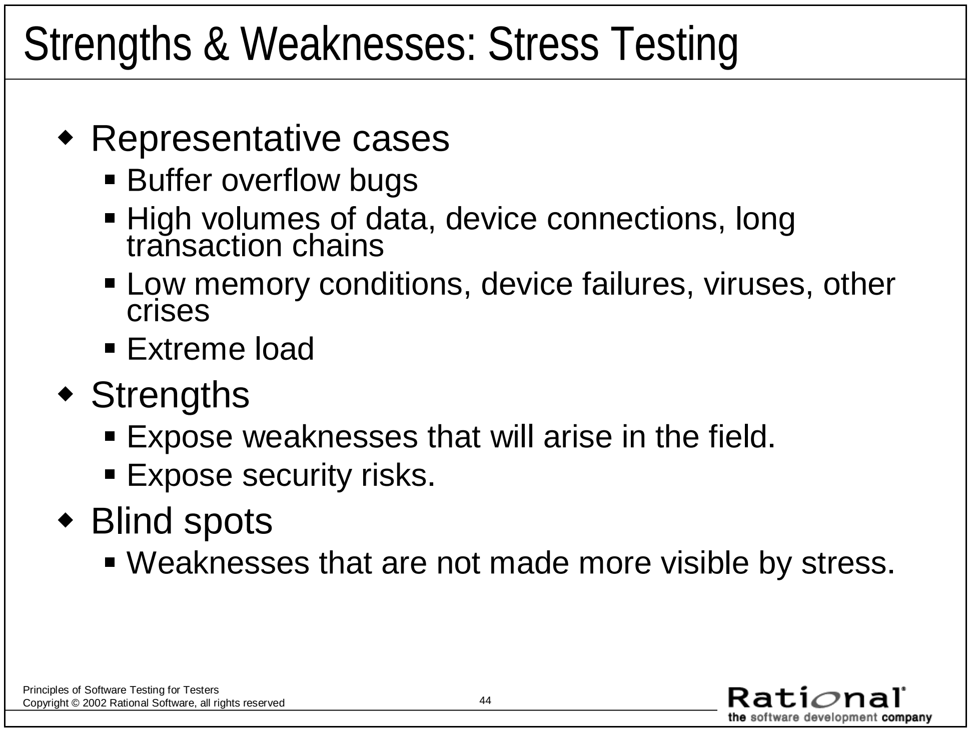
There are a few different definitions of stress testing. This one is focused on doing things that are so difficult for the program to handle that it will eventually fail.

- How does it fail? Does the program handle the failure graciously?
- Is that how and when it *should* fail?
- Are there follow-up consequences of this failure? If we kept using the program, what would happen?

This is a specialist's approach. For example,

- Some security testing experts use this to discover what holes are created in the system when part of the system is taken out of commission.
- Giving the program extremely large numbers is a form of stress testing. Crashes that result from these failures are often dismissed by programmers, but many break-ins start by exploiting a buffer over-run. For more on this approach, see James Whittaker, *How to Break Software* (2002).
- Some people use load testing tools to discover functional weaknesses in the program. Logic errors sometimes surface as the program gets less stable (because of the high load and the odd patterns of data that the program has to deal with during high load.)
- This is an extreme form of risk-based testing.

Strengths & Weaknesses: Stress Testing



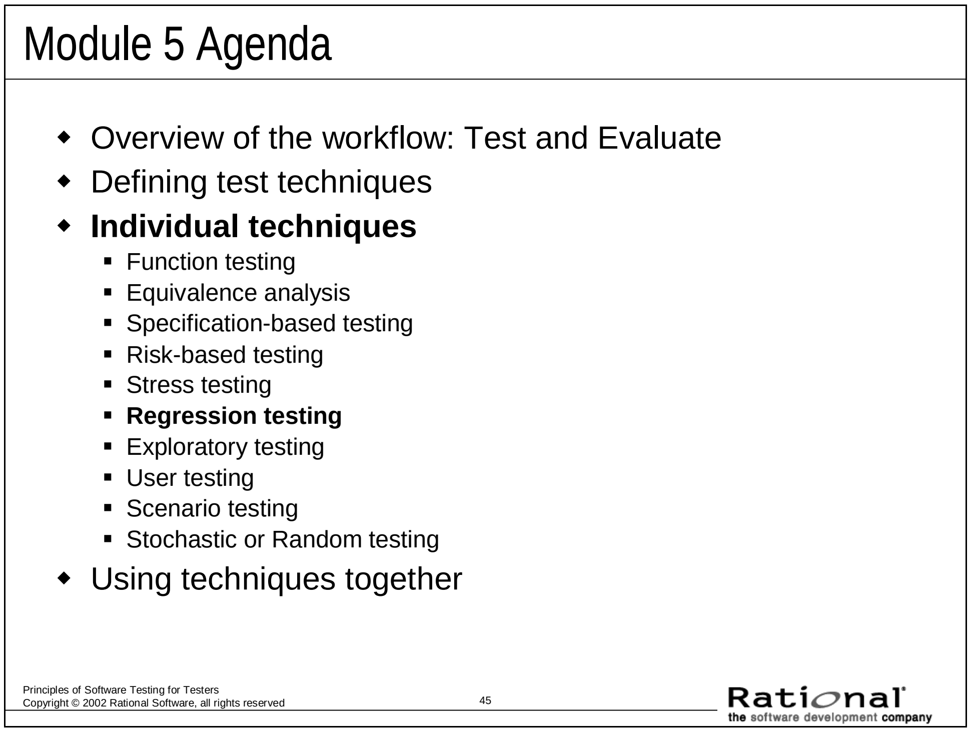
Strengths & Weaknesses: Stress Testing

- Representative cases
  - Buffer overflow bugs
  - High volumes of data, device connections, long transaction chains
  - Low memory conditions, device failures, viruses, other crises
  - Extreme load
- Strengths
  - Expose weaknesses that will arise in the field.
  - Expose security risks.
- Blind spots
  - Weaknesses that are not made more visible by stress.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 44

This is what hackers do when they pummel your site with denial of service attacks. A good vision for stress testing is that the nastiest and most skilled hacker should be a tester on your team, who uses the technique to find functional problems.

Module 5 Agenda



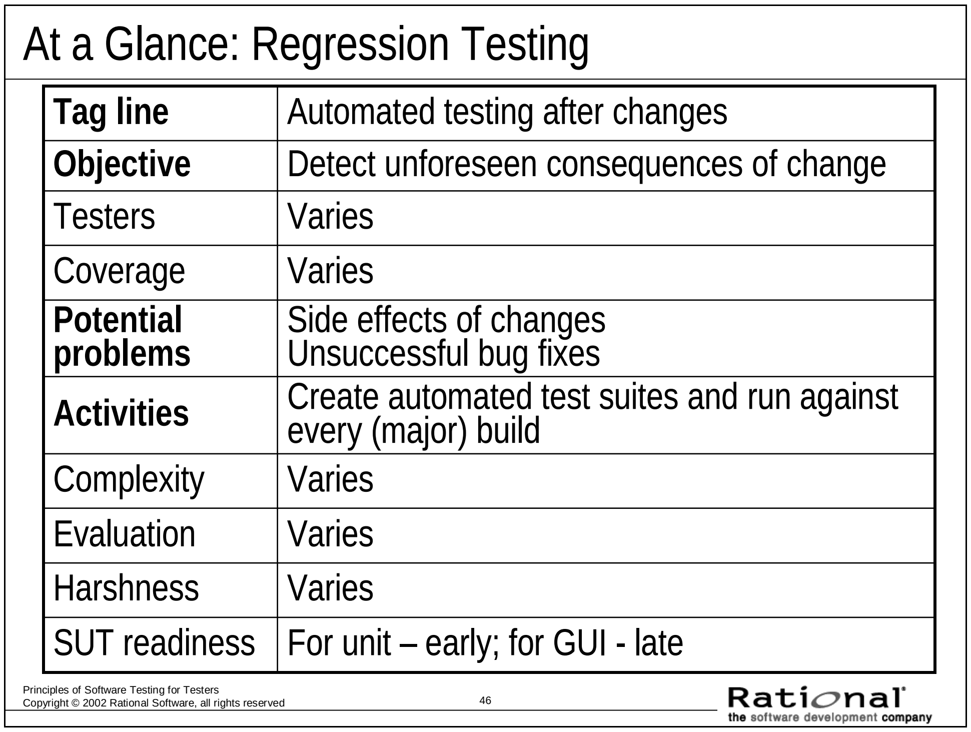
Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - **Regression testing**
  - Exploratory testing
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

Principles of Software Testing for Testers

45
Rational
the software development company

At a Glance: Regression Testing



At a Glance: Regression Testing

Tag line	Automated testing after changes
Objective	Detect unforeseen consequences of change
Testers	Varies
Coverage	Varies
Potential problems	Side effects of changes Unsuccessful bug fixes
Activities	Create automated test suites and run against every (major) build
Complexity	Varies
Evaluation	Varies
Harshness	Varies
SUT readiness	For unit – early; for GUI - late

46
Rational
the software development company

Regression testing refers to the automated testing of the SUT after changes. The name implies that its primary function is to prevent regression, i.e. the reappearance of a defect previously fixed, but in practice, the term is widely used to refer to any test automation that repeats the same tests over and over.

Regression testing is most effective when **combined with** other testing techniques, which we'll discuss at the end of this module.

Where should you use regression testing? Where **efficiency of executing** the tests time and time again is a primary concern. For example:

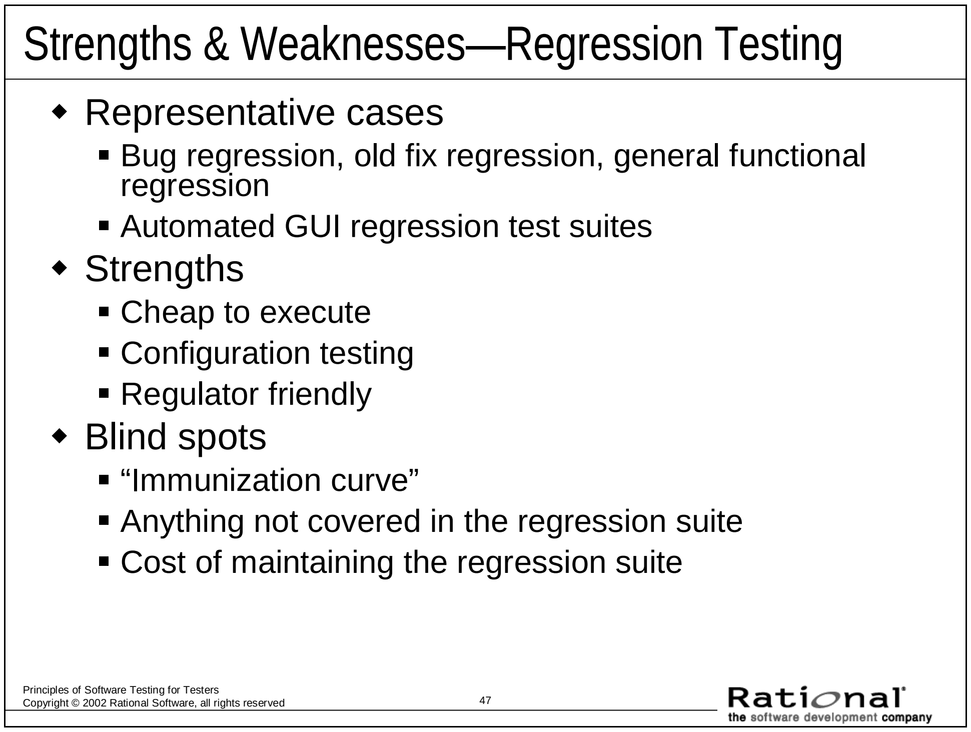
- **Build Verification Tests** (BVTs or "smoke tests") are a form of regression testing used to determine whether to accept a build into further testing and are covered in Module 8 of the course.
- **Configuration Tests**, where you check that an application functions identically with different operating systems, database servers, web servers, web browsers, etc., are another example where you need highly efficient execution.

Pay careful attention to the stability of the interfaces that you use to drive the SUT. **Testing through an API** is generally a better strategy than testing through the GUI, for two reasons.

1. GUIs change much more frequently than APIs, as usability issues are discovered and improvements are made.
2. It's usually much easier to achieve high coverage of the underlying program logic by using the API. The majority of the code in any modern system deals with error conditions that may be hard to trigger through the GUI alone.

If you have a highly stateful application, you may want to combine tests where you stimulate through the API and observe at the GUI, or vice-versa.

Strengths & Weaknesses—Regression Testing



Strengths & Weaknesses—Regression Testing

- Representative cases
  - Bug regression, old fix regression, general functional regression
  - Automated GUI regression test suites
- Strengths
  - Cheap to execute
  - Configuration testing
  - Regulator friendly
- Blind spots
  - "Immunization curve"
  - Anything not covered in the regression suite
  - Cost of maintaining the regression suite

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 47

Lessons Learned, Chapter 5, has useful guidance for regression testing.

In planning regression testing, be sure that you understand the extent to which you can vary the tests effectively for coverage and track the variance in the test results.

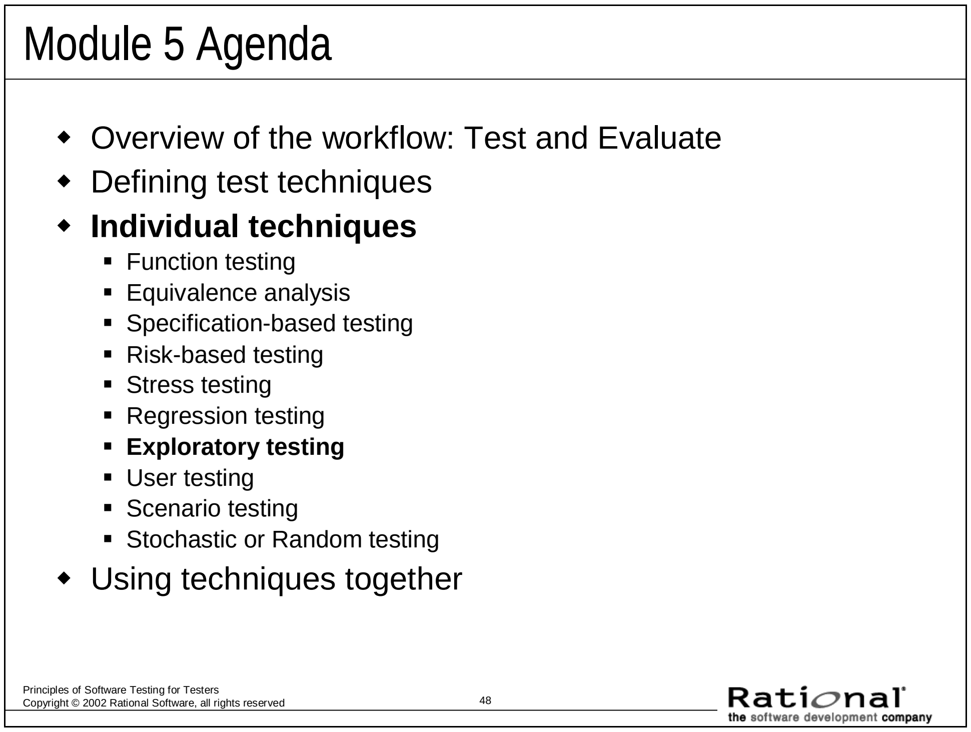
- Use different sequences (see scenario testing)
- Apply data for different equivalence class analyses
- Vary options and program settings, and
- Vary configurations.

Carefully plan the testability of the software under test to match the capabilities of any test tool you apply.

Do testing that essentially focuses on similar risks from build to build but not necessarily with the identical test each time.

There are a few cases (such as BVTs) where you may want to limit the variation.

Module 5 Agenda



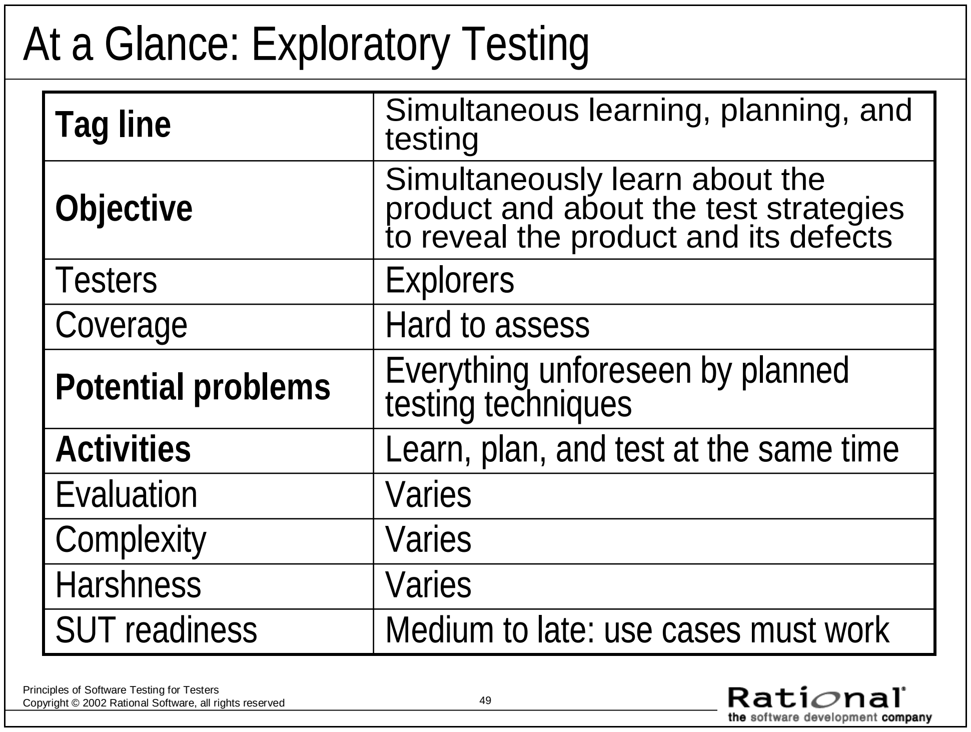
Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - **Exploratory testing**
  - User testing
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

Principles of Software Testing for Testers

48

At a Glance: Exploratory Testing



At a Glance: Exploratory Testing

Tag line	Simultaneous learning, planning, and testing
Objective	Simultaneously learn about the product and about the test strategies to reveal the product and its defects
Testers	Explorers
Coverage	Hard to assess
Potential problems	Everything unforeseen by planned testing techniques
Activities	Learn, plan, and test at the same time
Evaluation	Varies
Complexity	Varies
Harshness	Varies
SUT readiness	Medium to late: use cases must work

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 49

Rational
the software development company

With exploratory testing you **simultaneously**:

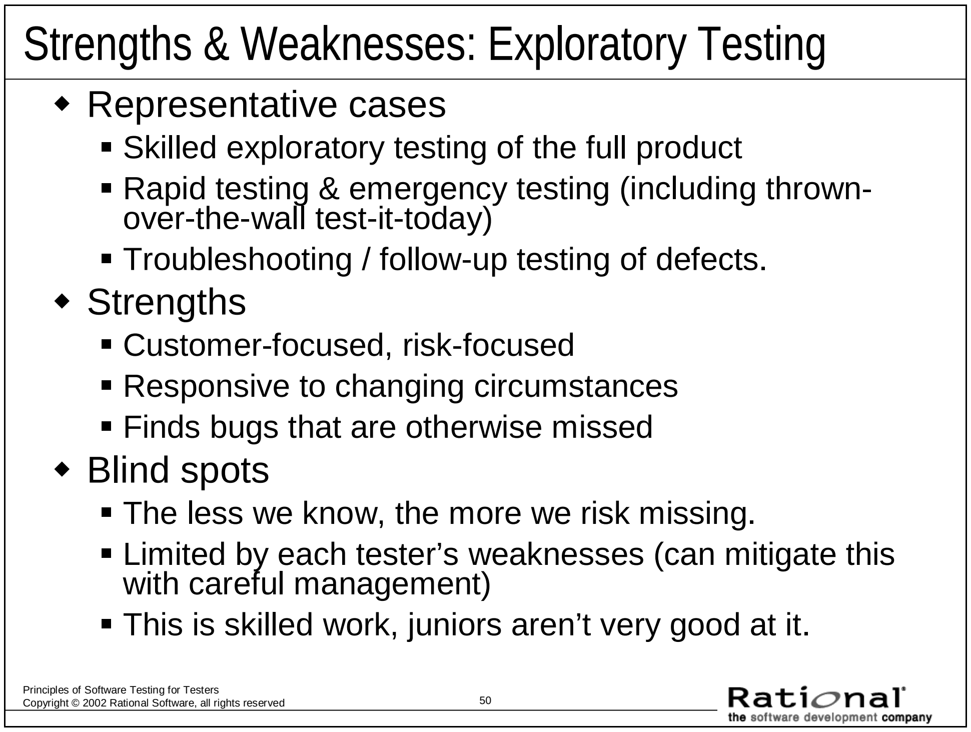
- Learn about the product
- Learn about the market
- Learn about the ways the product could fail
- Learn about the weaknesses of the product
- Learn about how to test the product
- Test the product
- Report the problems
- Advocate for repairs
- ***Develop new tests based on what you have learned so far.***

Everyone does some exploratory testing. For example, whenever you do follow-up testing to try to narrow the conditions underlying a failure or to try to find a more serious variation of a failure, you are doing exploratory testing. Most people do exploratory testing while they design tests. If you test the program while you design tests, trying out some of your approaches and gathering more detail about the program as you go, you are exploring.

If you do testing early in the process – during elaboration or in the first few iterations of implementation – the product is still in an embryonic state. Many artifacts that would be desirable for testing are just not available yet, and so the testers either have to not do the testing (this would be very bad) or learn as they go.

Acknowledgement: Many of these slides are derived from material given to us by James Bach (www.satisfice.com) and many of the ideas in these notes were reviewed and extended at the 7th Los Altos Workshop on Software Testing. We appreciate the assistance of the LAWST 7 attendees: Brian Lawrence, III, Jack Falk, Drew Pritsker, Jim Bampos, Bob Johnson, Doug Hoffman, Cem Kaner, Chris Agruss, Dave Gelperin, Melora Svoboda, Jeff Payne, James Tierney, Hung Nguyen, Harry Robinson, Elisabeth Hendrickson, Noel Nyman, Bret Pettichord, & Rodney Wilson.

Strengths & Weaknesses: Exploratory Testing



Strengths & Weaknesses: Exploratory Testing

- Representative cases
  - Skilled exploratory testing of the full product
  - Rapid testing & emergency testing (including thrown-over-the-wall test-it-today)
  - Troubleshooting / follow-up testing of defects.
- Strengths
  - Customer-focused, risk-focused
  - Responsive to changing circumstances
  - Finds bugs that are otherwise missed
- 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.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 50

Doing Exploratory Testing

- Keep your mission clearly in mind.
- Distinguish between testing and observation.
- While testing, be aware of the limits of your ability to detect problems.
- Keep notes that help you report what you did, why you did it, and support your assessment of product quality.
- Keep track of questions and issues raised in your exploration.

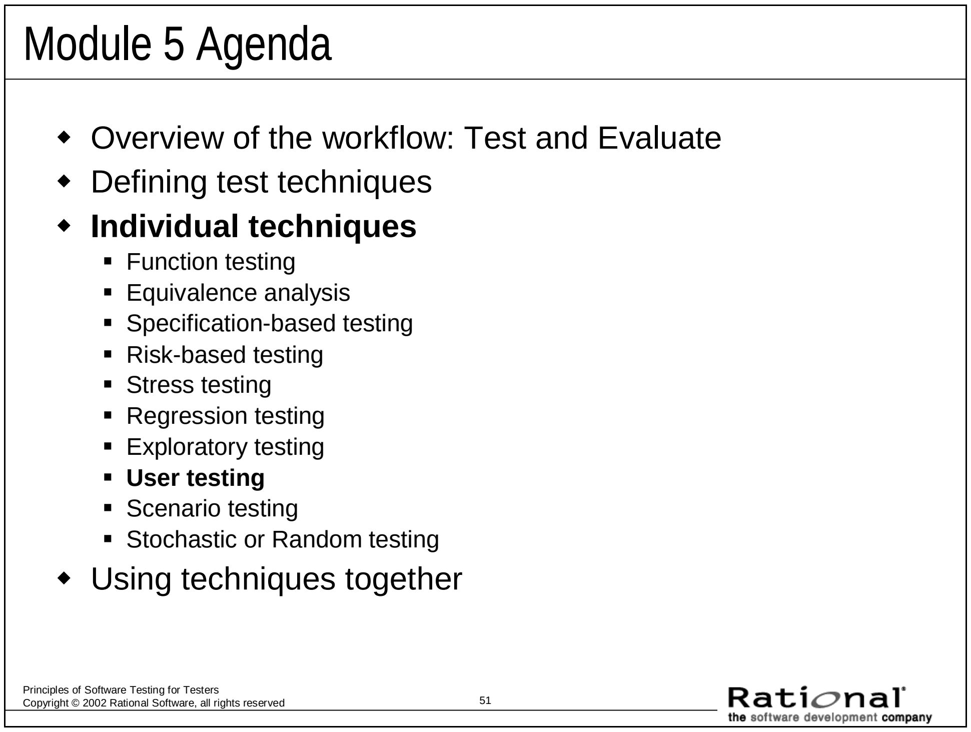
Problems to Be Aware Of

- Habituation may cause you to miss problems.
- Lack of information may impair exploration.
- Expensive or difficult product setup may increase the cost of exploring.
- Exploratory feedback loop my be too slow.
- Old problems may pop up again and again.
- High MTBF may not be achievable without well defined test cases and procedures, in addition to exploratory approach.

The question is not whether testers should do exploratory testing (that’s like asking whether people should breathe). Instead, we should ask:

- How systematically should people explore?
- How visible should exploratory testing practices be in the testing process?
- How much exploratory training should testers have?

Module 5 Agenda



Module 5 Agenda

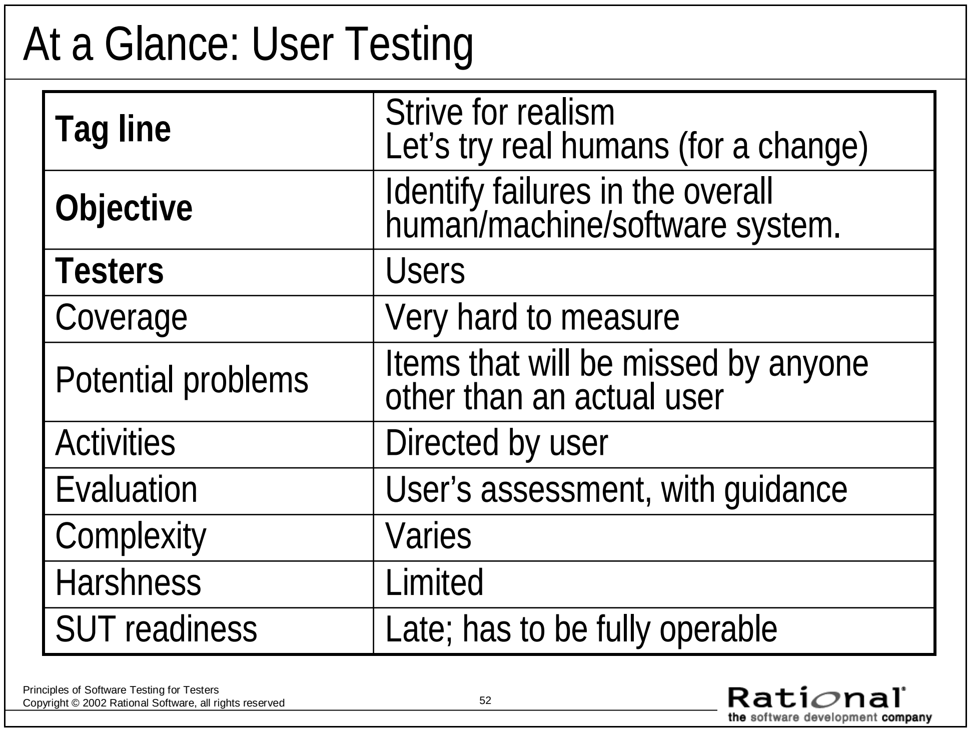
- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - **User testing**
  - Scenario testing
  - Stochastic or Random testing
- Using techniques together

Principles of Software Testing for Testers

51

Rational
the software development company

At a Glance: User Testing



At a Glance: User Testing

Tag line	Strive for realism Let's try real humans (for a change)
Objective	Identify failures in the overall human/machine/software system.
Testers	Users
Coverage	Very hard to measure
Potential problems	Items that will be missed by anyone other than an actual user
Activities	Directed by user
Evaluation	User's assessment, with guidance
Complexity	Varies
Harshness	Limited
SUT readiness	Late; has to be fully operable

Principles of Software Testing for Testers
52

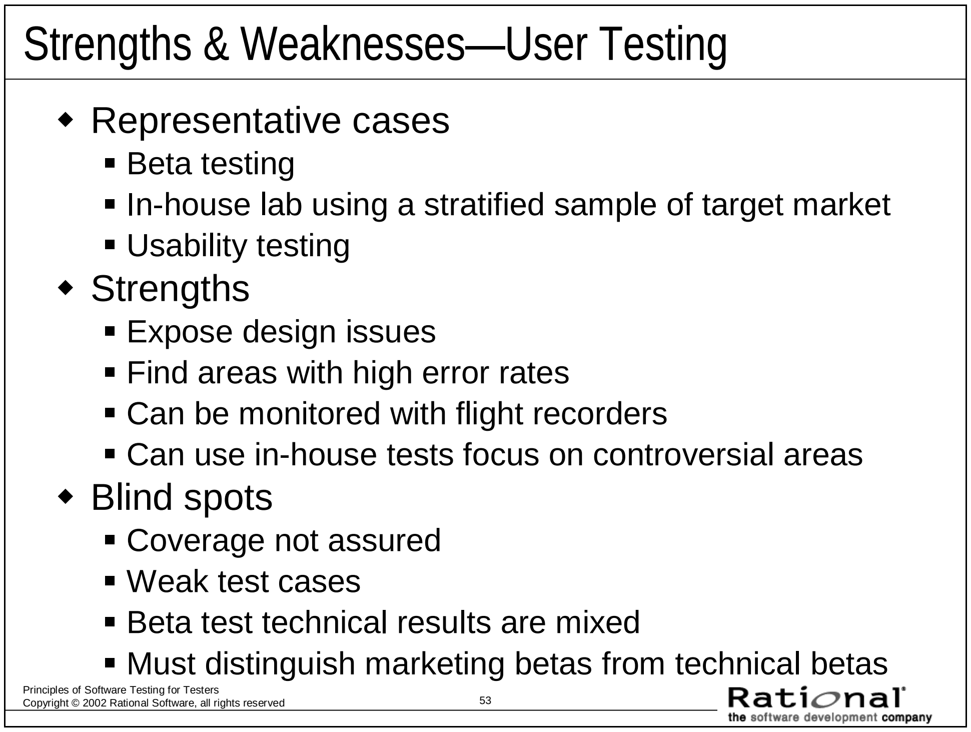
Beta testing is normally defined as testing by people who are outside of your company. These are often typical members of your market, but they may be selected in other ways.

Beta tests have different objectives. It's important to time and structure your test(s) in ways that help you meet your goals:

- Expert advice—the expert evaluates the program design. It is important to do this as early as possible, when basic changes are still possible.
- Configuration testing—the beta tester runs the software on her equipment, and tells you the results.
- Compatibility testing—the beta tester (possibly the manufacturer of the other software) runs the software in conjunction with other software, to see whether they are compatible.
- Bug hunting—the beta tester runs the software and reports software errors.
- Usability testing—the beta tester runs the software and reports difficulties she had using the product.
- Pre-release acceptance tests—the beta tester runs the product to discover whether it behaves well on her system or network. The goal is convincing the customer that the software is OK, so that she'll buy it as soon as it ships.
- News media reviews—some reporters want early software. They are gratified by corporate responsiveness to their suggestions for change. Others expect finished software and are intolerant of pre-release bugs.

For more discussion of the diversity of beta tests, see Kaner, Falk & Nguyen, *Testing Computer Software*, pp. 291-294.

Strengths & Weaknesses—User Testing



Strengths & Weaknesses—User Testing

- Representative cases
  - Beta testing
  - In-house lab using a stratified sample of target market
  - Usability testing
- Strengths
  - Expose design issues
  - Find areas with high error rates
  - Can be monitored with flight recorders
  - Can use in-house tests focus on controversial areas
- Blind spots
  - Coverage not assured
  - Weak test cases
  - Beta test technical results are mixed
  - Must distinguish marketing betas from technical betas

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 53

Rational the software development company

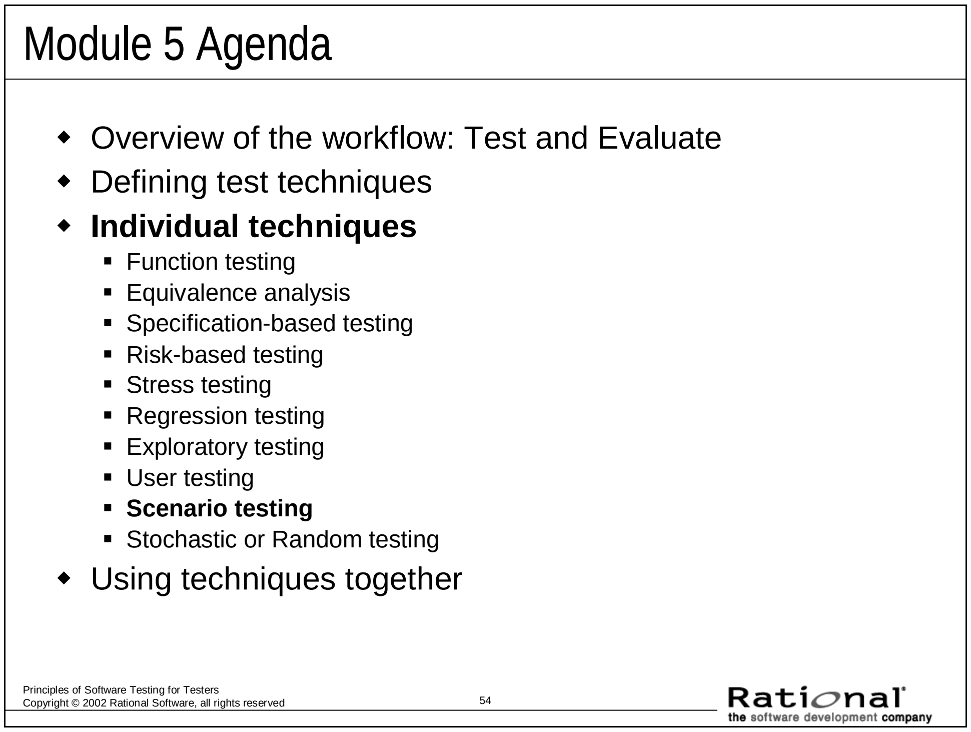
Prof. Kaner comments:

There is a very simple example of something that we did at Electronic Arts. We made many programs that printed in very fancy ways on color printers. We gave you the files to print as part of the beta, you made print outs and wrote on the back of the page what your printer was and what your name was. If you were confused about the settings, when we got your page back, we called you up. We had a large population of people with a large population of strange and expensive printers that we couldn't possibly afford to bring in-house.

So we could tell whether it passed or failed, we also did things like sending people parts of the product and a script to walk through and we would be on the phone with them and say what do you see on the screen? We wanted to do video compatibility where they're across the continent.

So you are relying on their eyes to be your eyes. But you're on the phone, you don't ask them if it looks okay, you ask them what is in this corner? And you structure what you're going to look at If you think you are at risk on configuration you should have some sense of how configurations will show up the configuration failures. Write tests to expose those, get them to your customers, and then find out whether those tests passed or failed by checking directly on these specific tests.

Module 5 Agenda



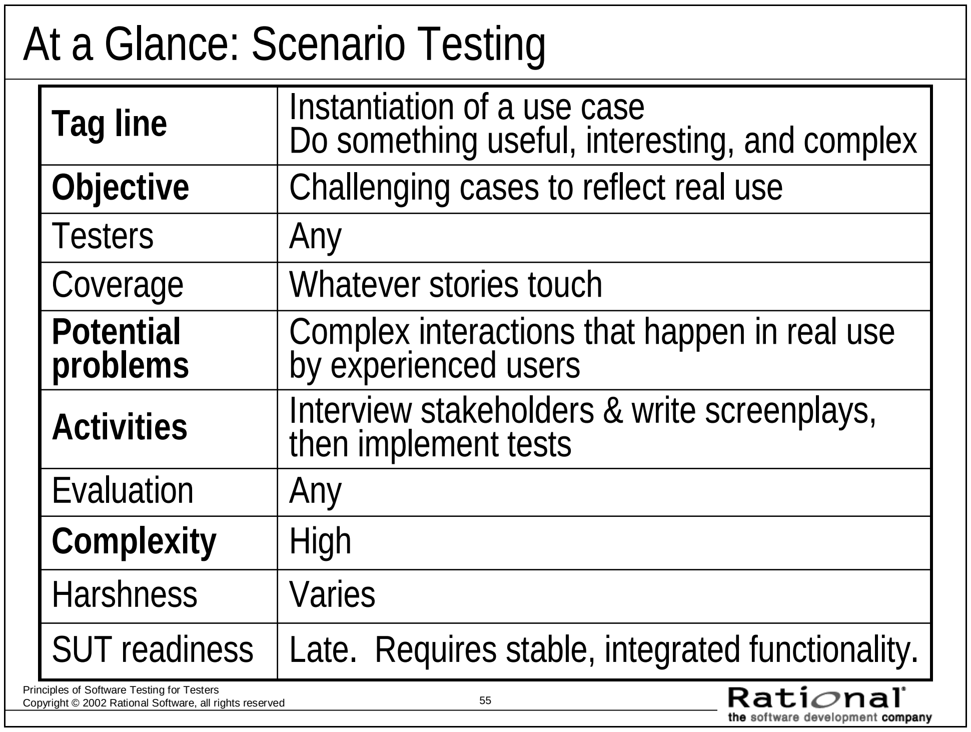
Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - **Scenario testing**
  - Stochastic or Random testing
- Using techniques together

54

Rational
the software development company

At a Glance: Scenario Testing

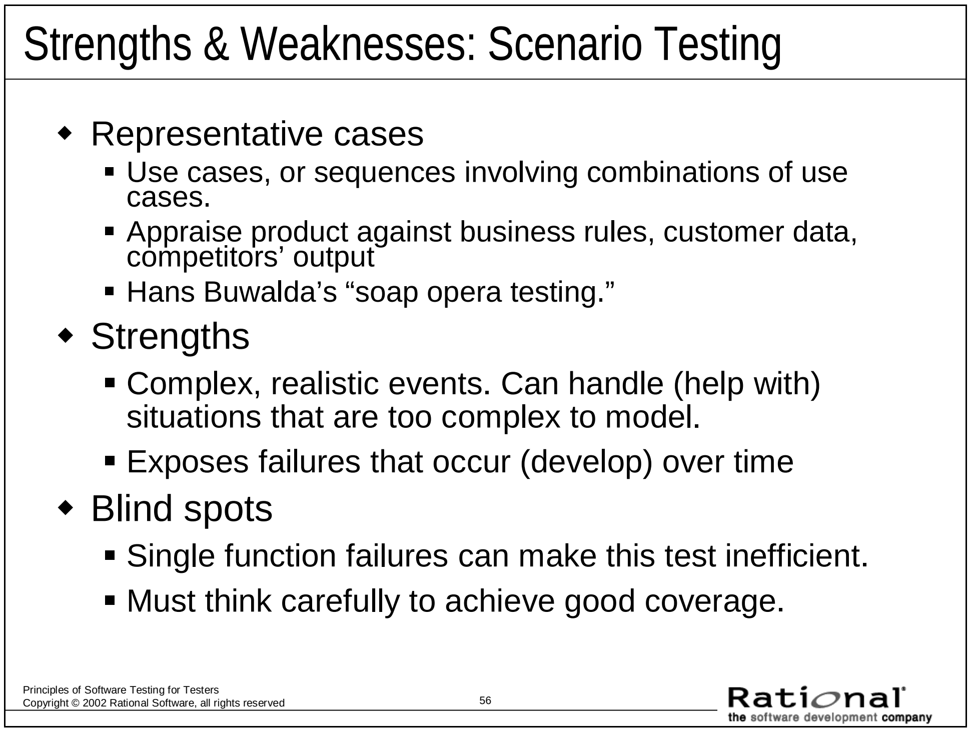


At a Glance: Scenario Testing

Tag line	Instantiation of a use case Do something useful, interesting, and complex
Objective	Challenging cases to reflect real use
Testers	Any
Coverage	Whatever stories touch
Potential problems	Complex interactions that happen in real use by experienced users
Activities	Interview stakeholders & write screenplays, then implement tests
Evaluation	Any
Complexity	High
Harshness	Varies
SUT readiness	Late. Requires stable, integrated functionality.

Scenarios are great ways to capture realism in testing. They are much more complex than most other techniques and they focus on end-to-end experiences that users really have.

Strengths & Weaknesses: Scenario Testing



Strengths & Weaknesses: Scenario Testing

- Representative cases
  - Use cases, or sequences involving combinations of use cases.
  - Appraise product against business rules, customer data, competitors' output
  - Hans Buwalda's "soap opera testing."
- 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.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved
56

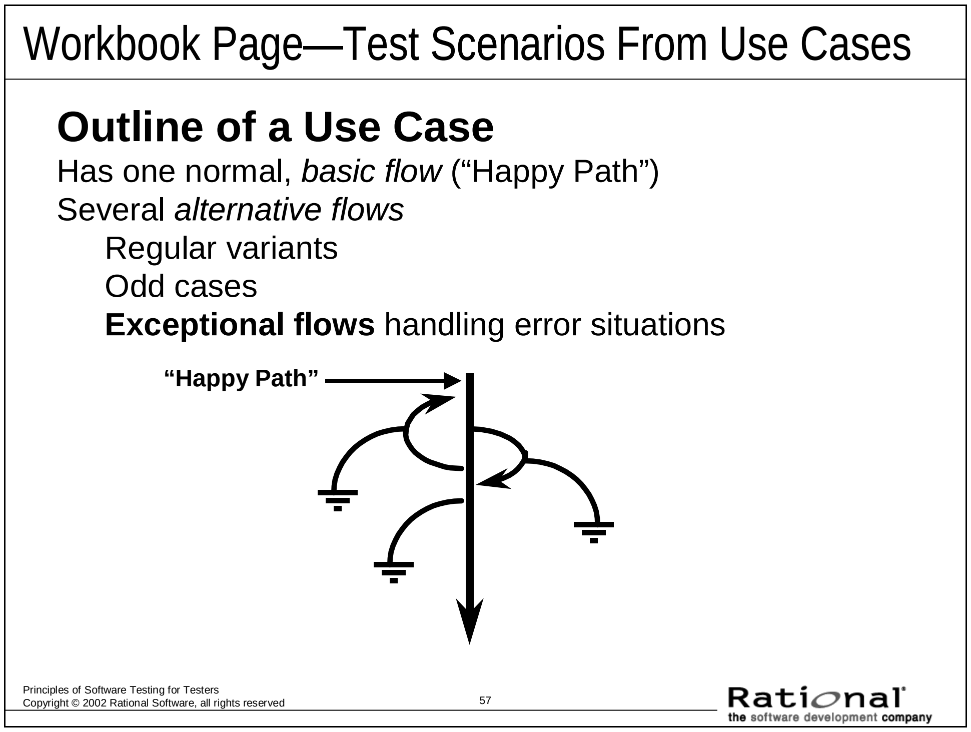
Rational
the software development company

Scenario tests are expensive. So it's important to get them right.

- Realism is important for credibility.
- Don't use scenarios to find simple bugs efficiently. Scenario tests are too complex and tied to too many features.
- Start your testing effort with simpler tests to find the simple defects. If you start with scenario tests, you will be blocked by simple bugs.

There's a risk of missing coverage by relying too heavily on scenario testing alone. A mitigation strategy for that risk is to use a traceability matrix for assessing coverage, as we've shown before.

Workbook Page—Test Scenarios From Use Cases



Workbook Page—Test Scenarios From Use Cases

Outline of a Use Case

Has one normal, *basic flow* (“Happy Path”)

Several *alternative flows*

- Regular variants
- Odd cases
- **Exceptional flows** handling error situations

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 57

Rational the software development company

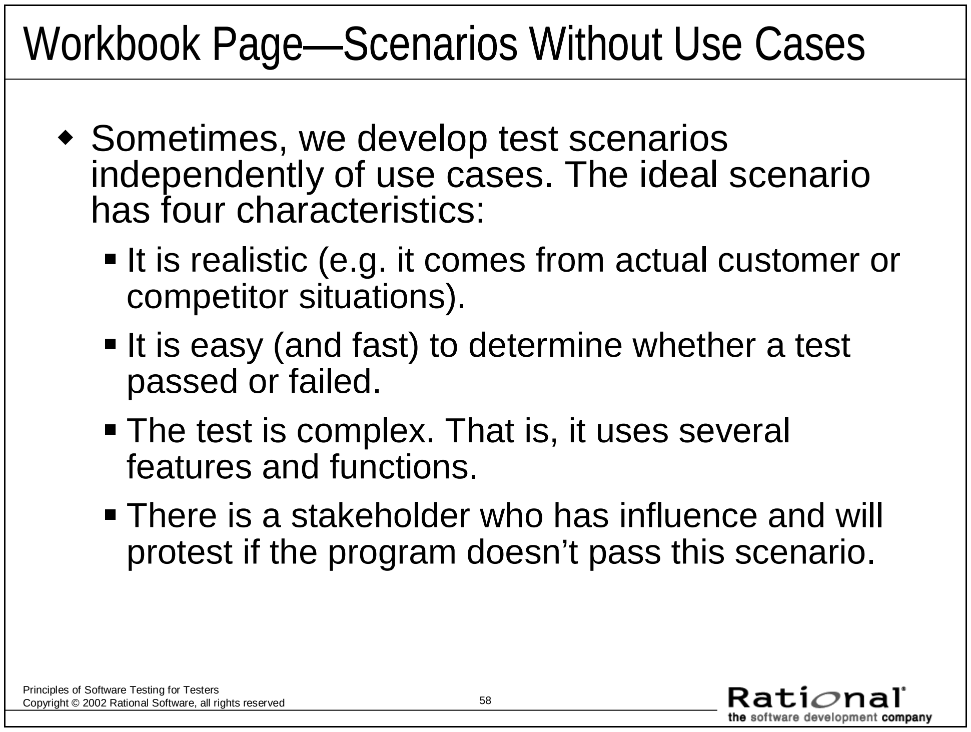
Use Cases may be a good source of test scenarios. Usually, you will want to string several use cases together as a test scenario.

Use-Case Contents

1. Brief Description
2. Flow of Events
   Basic Flow of Events
   Alternative Flows of Events
3. Special Requirements
4. Pre-Conditions
5. Post-Conditions
6. Extension Points
7. Relationships
8. Candidate Scenarios
9. Use-Case Diagrams
10. Other Diagrams/Enclosures

The **Flow of Events** of a use case contains the most important information derived from use-case modeling work. It should describe the use case's flow of events clearly enough for an outsider to easily understand it. Remember the flow of events should present what the system does, not how the system is designed to perform the required behavior.

Workbook Page—Scenarios Without Use Cases



Workbook Page—Scenarios Without Use Cases

- Sometimes, we develop test scenarios independently of use cases. The ideal scenario has four characteristics:
  - It is realistic (e.g. it comes from actual customer or competitor situations).
  - It is easy (and fast) to determine whether a test passed or failed.
  - The test is complex. That is, it uses several features and functions.
  - There is a stakeholder who has influence and will protest if the program doesn't pass this scenario.

Why develop test scenarios independently of use cases?

- Some development teams don't do a thorough job of use case analysis. Certainly, use cases play an important role in RUP. But users of RUP may not adopt all of the RUP recommendations. The testing group has to be prepared to derive test cases from whatever information is available.
- Even if a development team creates a strong collection of use cases, an analysis from outside of the developers' design thinking may expose problems that are not obvious from analysis of the use cases. The tester, collecting data for the scenario test, may well rely on different people's inputs than the development team when it developed use cases.

Some ways to trigger thinking about scenarios:

Benefits-driven: People want to achieve X. How will they do it, for the following X's?

Sequence-driven: People (or the system) typically does task X in an order. What are the most common orders (sequences) of subtasks in achieving X?

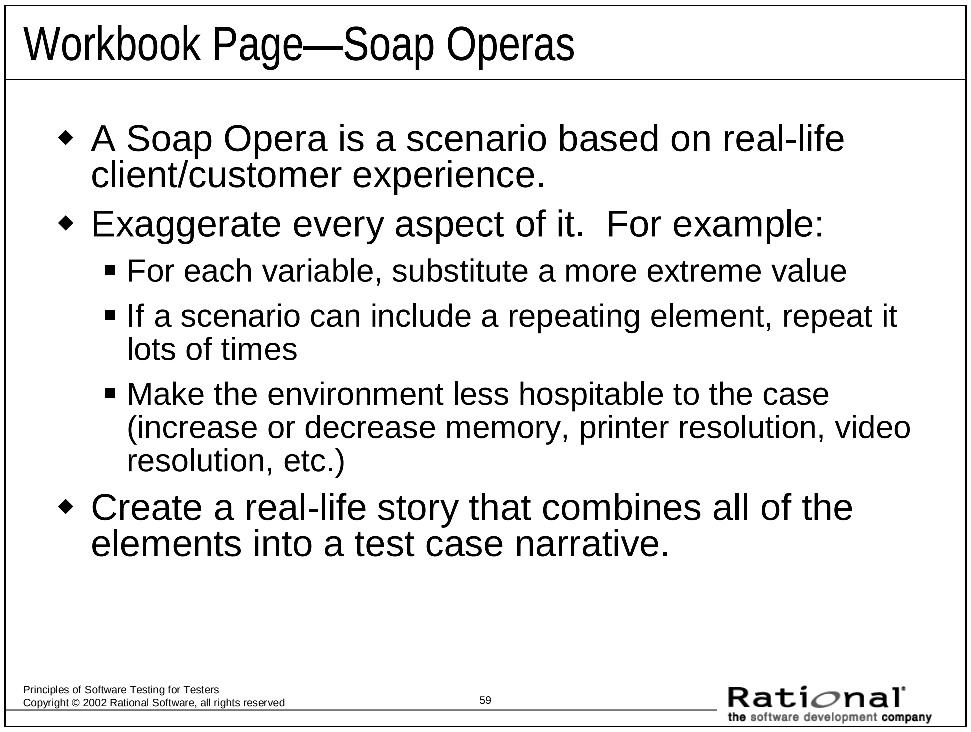
Transaction-driven: We are trying to complete a specific transaction, such as opening a bank account or sending a message. What are the steps, data items, outputs, displays etc.?

Get use ideas from competing product: Their docs, advertisements, help, etc., all suggest best or most interesting uses of their products. How would our product do these things?

Competitor driven: Hey, look at these cool documents they can create. Look at how they display things (e.g. Netscape's superb handling of malformed HTML code). How do we handle this?

Customer's forms driven: Here are the forms the customer produces. How can we work with (read, fill out, display, verify, whatever) them?

Workbook Page—Soap Operas



Workbook Page—Soap Operas

- A Soap Opera is a scenario based on real-life client/customer experience.
- Exaggerate every aspect of it. For example:
  - For each variable, substitute a more extreme value
  - If a scenario can include a repeating element, repeat it lots of times
  - Make the environment less hospitable to the case (increase or decrease memory, printer resolution, video resolution, etc.)
- Create a real-life story that combines all of the elements into a test case narrative.

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 59

Rational the software development company

These are example soap opera scenarios from: Hans Buwalda, *Soap Opera Testing*, Software Testing Analysis & Review conference, Orlando, FL, May 2000.

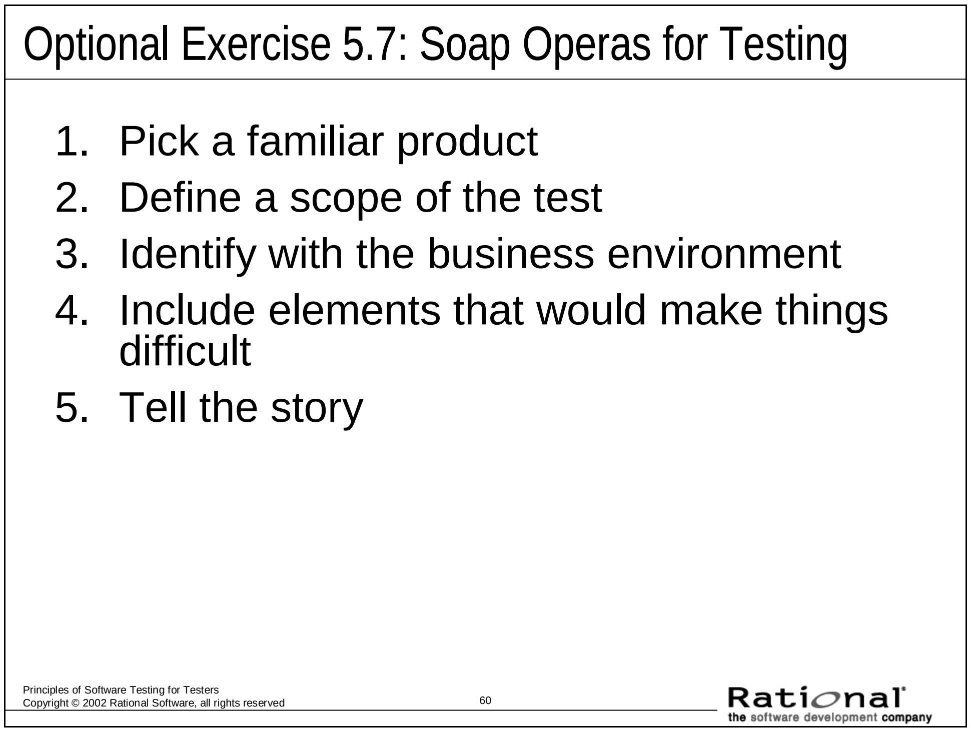
Pension Fund

> William starts as a metal worker for Industrial Entropy Incorporated in 1955. During his career he becomes ill, works part time, marries, divorces, marries again, gets 3 children, one of which dies, then his wife dies and he marries again and gets 2 more children….

World Wide Transaction System for an international Bank

> A fish trade company in Japan makes a payment to a vendor on Iceland. It should have been a payment in Icelandic Kronur, but it was done in Yen instead. The error is discovered after 9 days and the payment is revised and corrected, however, the interest calculation (value dating)…

Optional Exercise 5.7: Soap Operas for Testing



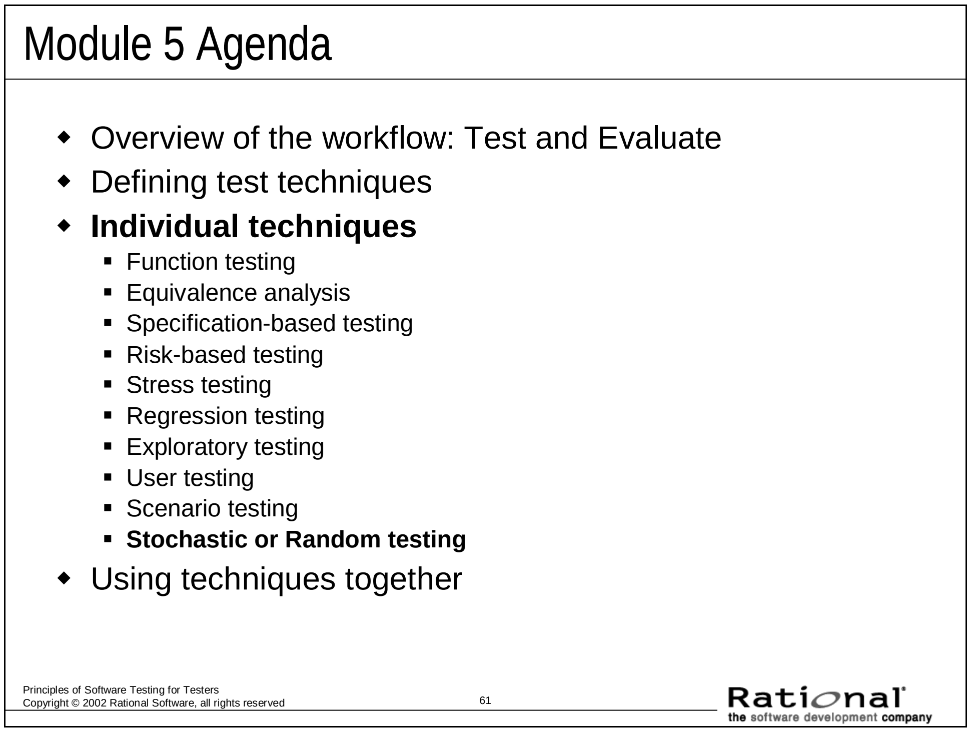
Optional Exercise 5.7: Soap Operas for Testing

1. Pick a familiar product
2. Define a scope of the test
3. Identify with the business environment
4. Include elements that would make things difficult
5. Tell the story

Rational® the software development company

Optional Exercise

Module 5 Agenda



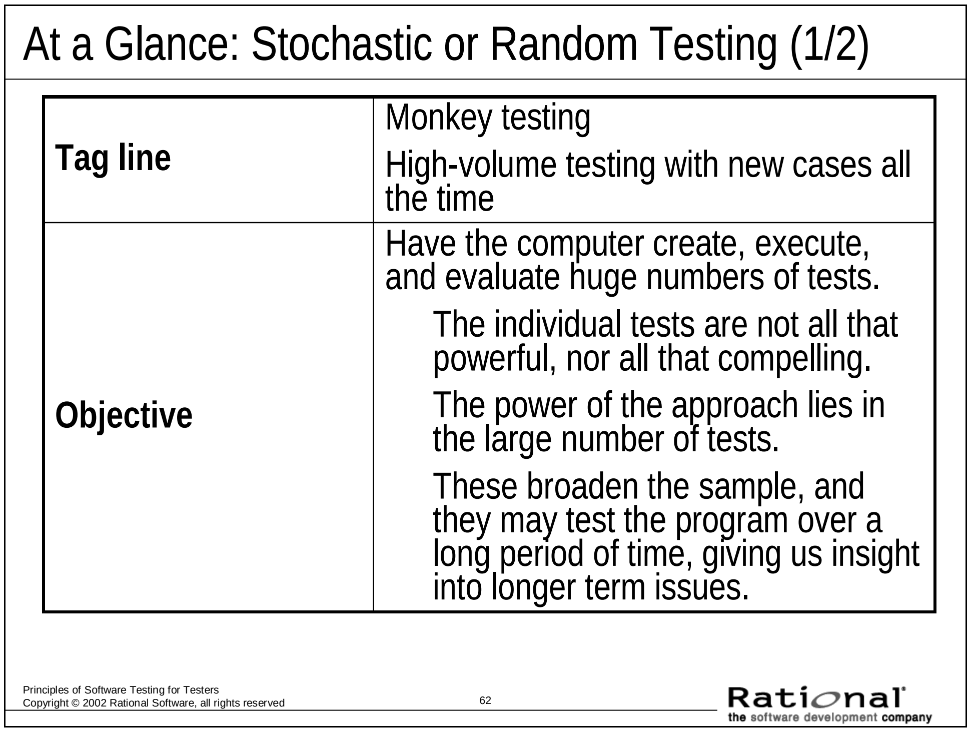
Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- **Individual techniques**
  - Function testing
  - Equivalence analysis
  - Specification-based testing
  - Risk-based testing
  - Stress testing
  - Regression testing
  - Exploratory testing
  - User testing
  - Scenario testing
  - **Stochastic or Random testing**
- Using techniques together

Principles of Software Testing for Testers

61
Rational the software development company

At a Glance: Stochastic or Random Testing (1/2)



At a Glance: Stochastic or Random Testing (1/2)

Tag line	Monkey testing High-volume testing with new cases all the time
Objective	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.

The essence of this technique is that, while the strategy is designed by a human; the individual test cases are generated by machine. Kaner's *Architectures of Test Automation*, in your student kit, discusses this in more detail.

Noel Nyman of Microsoft coined the term "monkey testing" and has developed some of the best material on this subject. The name was inspired by the teaser:

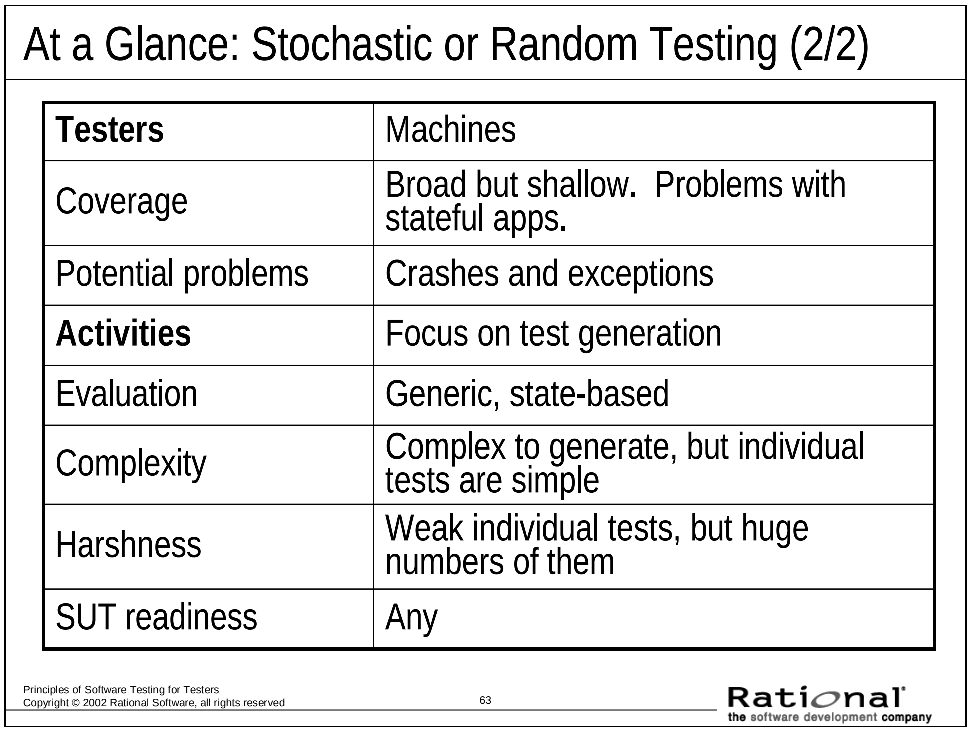
> "If 12 monkeys pound on keyboards at random, how long will it take before they re-create the works of Shakespeare?"

Nyman's description and source code for "Freddy", a monkey tester used for compatibility testing at Microsoft, can be found in is the appendix to Tom Arnold's *VT 6 Bible*. For experience reports, see Noel Nyman, "Using Monkey Test Tools," *Software Test and Quality Engineering Magazine*, January/February 2000, available at www.stickyminds.com

Harry Robinson, also of Microsoft, has published a few papers on this style of test generation at his site www.model-based-testing.org. In Robinson's terminology, the "model" is the combinatorial space and set of algorithms used to generate tests.

> "Monkey testing should not be your only testing. Monkeys don't understand your application, and in their ignorance they miss many bugs."
>
> —Noel Nyman, "Using Monkey Test Tools," STQE, Jan/Feb 2000

At a Glance: Stochastic or Random Testing (2/2)



At a Glance: Stochastic or Random Testing (2/2)

Testers	Machines
Coverage	Broad but shallow. Problems with stateful apps.
Potential problems	Crashes and exceptions
Activities	Focus on test generation
Evaluation	Generic, state-based
Complexity	Complex to generate, but individual tests are simple
Harshness	Weak individual tests, but huge numbers of them
SUT readiness	Any

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 63

Rational the software development company

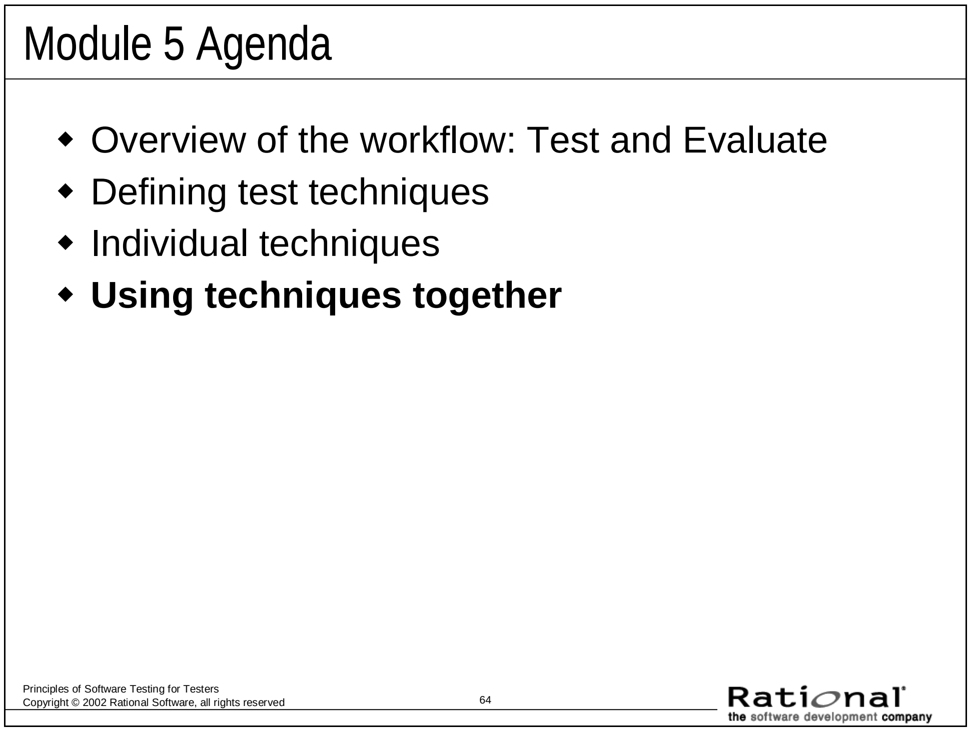
What do we mean by *random* and *stochastic*?

A variable that is *random* has a value that is basically unpredictable. If you're talking about a set of values that are random then the set are basically unpredictable. If the random value depends upon the sequence, then you're not just dealing with something that is random, you're dealing with something that is randomly changing over time -- that is a *stochastic* variable.

For example, if you go to Las Vegas and play Blackjack how much you will win or lose on a given hand is a random variable, but how much is left in your pocket is a stochastic variable. It depends not just on how much you won or lost this time but rather on what's been going on time after time. The Dow Jones Index is a stochastic variable. How much it changes today is the random variable.

In high-volume random testing, where you go next depends on where you are now and the next random variable -- it is a stochastic process. An important theorem is that a stochastic process, that depends only on current position and one random variable to move to the next place, will reach every state that can theoretically be reached in that system, if you run the process for a long enough time. You can prove that over a long enough period you will have 100% state coverage, as long as you can show that the states could ever be reached.

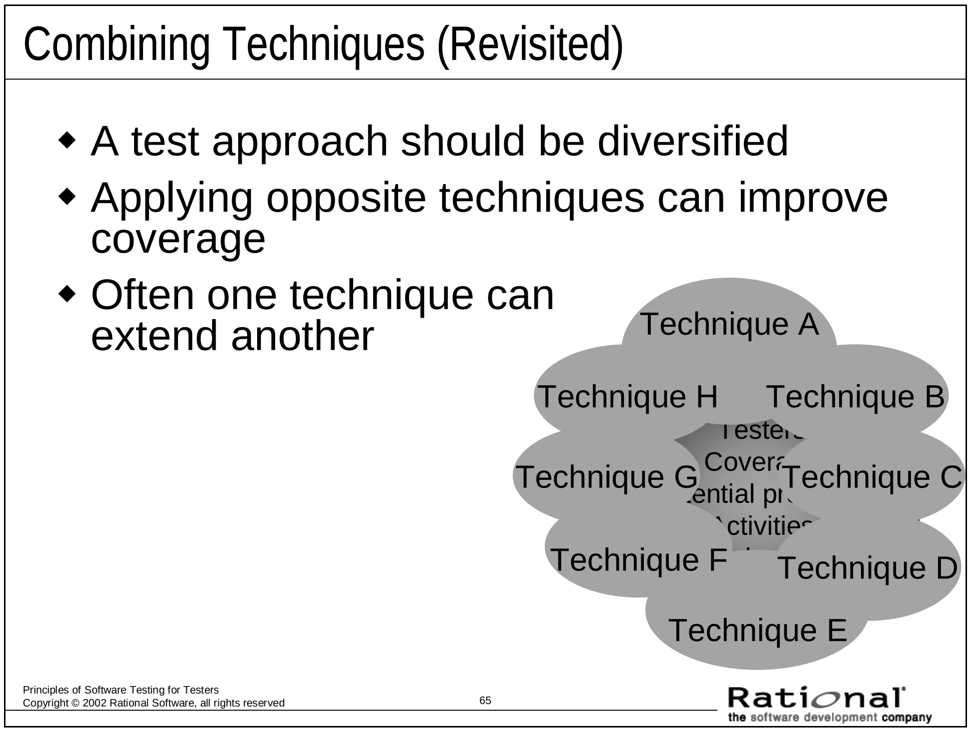
Using Techniques Together



Module 5 Agenda

- Overview of the workflow: Test and Evaluate
- Defining test techniques
- Individual techniques
- **Using techniques together**

Combining Techniques (Revisited)



Combining Techniques (Revisited)

- A test approach should be diversified
- Applying opposite techniques can improve coverage
- Often one technique can extend another

Technique A
Technique B
Technique C
Technique D
Technique E
Technique F
Technique G
Technique H

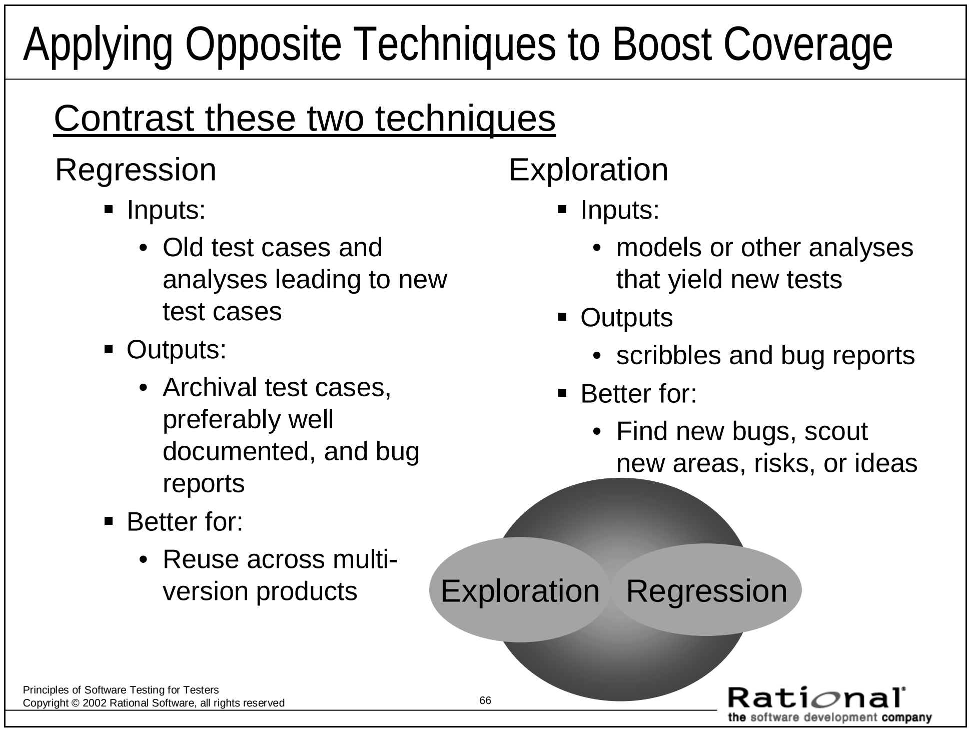
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 65

Earlier in this module, the concept of of complementary techniques was introduced. Now that you have visited the techniques in detail, it's useful to think about two valuable ways of combining them:

1. Using opposite techniques independently
2. Using complementary techniques together

The next two slides cover examples of each.

Applying Opposite Techniques to Boost Coverage



Applying Opposite Techniques to Boost Coverage

Contrast these two techniques

Regression
- Inputs:
  - Old test cases and analyses leading to new test cases
- Outputs:
  - Archival test cases, preferably well documented, and bug reports
- Better for:
  - Reuse across multi-version products

Exploration
- Inputs:
  - models or other analyses that yield new tests
- Outputs
  - scribbles and bug reports
- Better for:
  - Find new bugs, scout new areas, risks, or ideas

66

Rational
the software development company

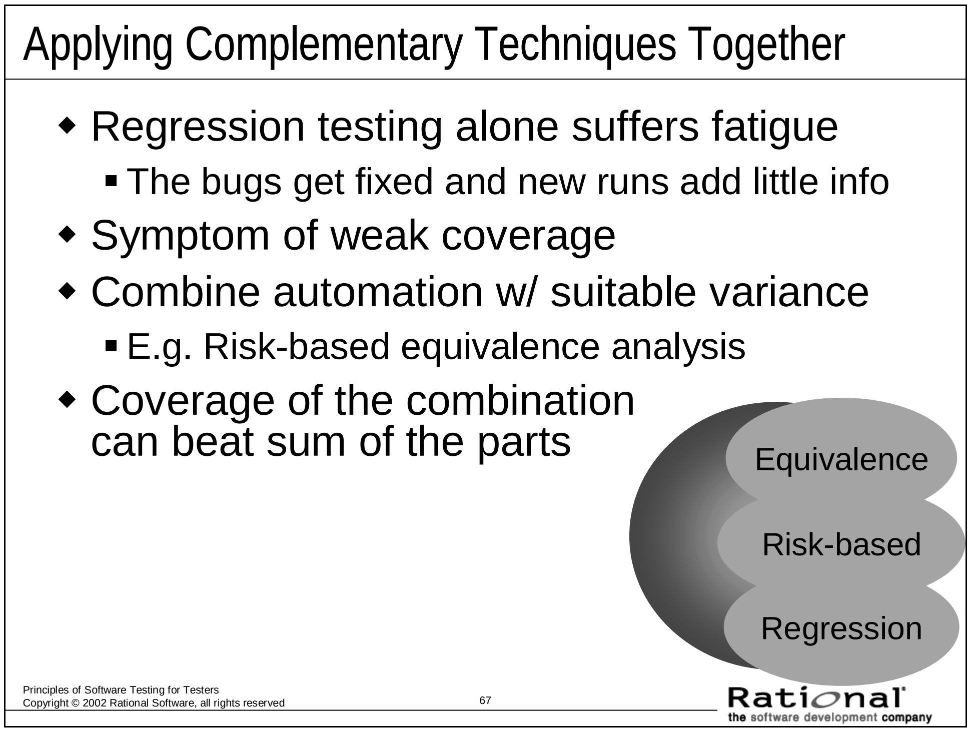
Regression Testing and Exploratory Testing are perhaps the easiest techniques to contrast. Consider the two as processes with inputs and outputs.

The regression tester starts with test cases that he will reuse and the motivations for those test cases. The regression tester executes those tests, discovers some are out of date, some can be stricken, and generates two different types of documents. 1) bug reports and 2) improved tests. The regression tester is focused on creating materials for reuse.

The exploratory tester, on the other hand, comes in with whatever information is available, but not with defined test cases. The exploratory tester does testing and makes notes in a private notebook. From those scribbles the exploratory tester also writes bug reports. But the scribbles in the book are not going anywhere outside this book. There's nothing available for reuse – just the bug reports.

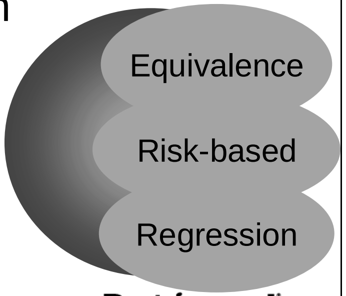
Neither technique would be safe as the only approach to testing. Applying them both, however, significantly improves the diversification of your test approach.

Applying Complementary Techniques Together



Applying Complementary Techniques Together

- Regression testing alone suffers fatigue
  - The bugs get fixed and new runs add little info
- Symptom of weak coverage
- Combine automation w/ suitable variance
  - E.g. Risk-based equivalence analysis
- Coverage of the combination can beat sum of the parts



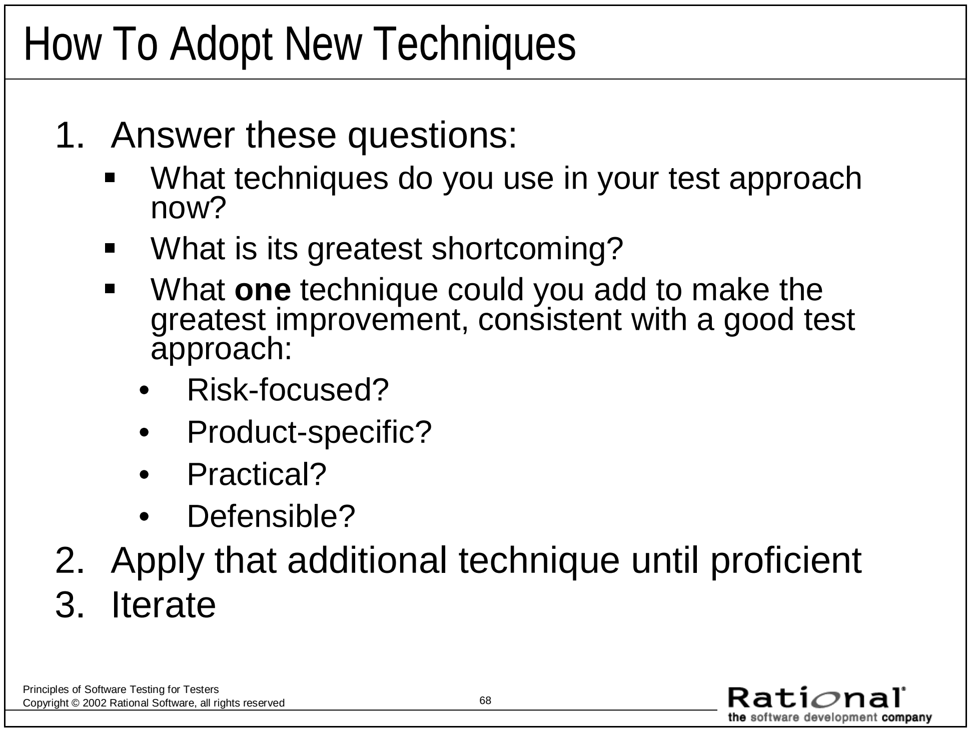
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 67

Rational
the software development company

Another way of combining techniques is to use one technique to extend another. For example, **Regression testing** is much more effective when **extended with other testing techniques** than when used in isolation. Examples of combination include…

- **Equivalence analysis**: There are many techniques available for extending test automation with variable data and all regression tools support variable data. If you have done good **risk-based** equivalence analysis, and can extend function regression testing with good test data, you can achieve the combined benefits of those techniques.
- **Function testing**: XP (eXtreme Programming) advocates that developers produce exhaustive automated unit tests that are run after every coding task to facilitate refactoring (changing code). Because the XP test suites are sufficiently comprehensive and are run continuously, they provide immediate feedback of any unforeseen breakage caused by a change. JUnit is a popular open source tool for this.
- **Specification-based testing**: An important extension to spec-based testing is the practice of Test-first Design (covered in RUP as a developer practice and also advocated by XP). With **Test-first Design**, you use tests as a primary form of requirements specification and rerun the tests on every build to provide immediate feedback on any breakage.
- **Scenario testing**: Some teams have success automating simple scenarios and interactions. This works when you can easily maintain the tests are are conscientious about discarding tests that no longer add useful information. A good heurisitc is to make sure that test maintenance cost is kept low to avoid blocking any test development.

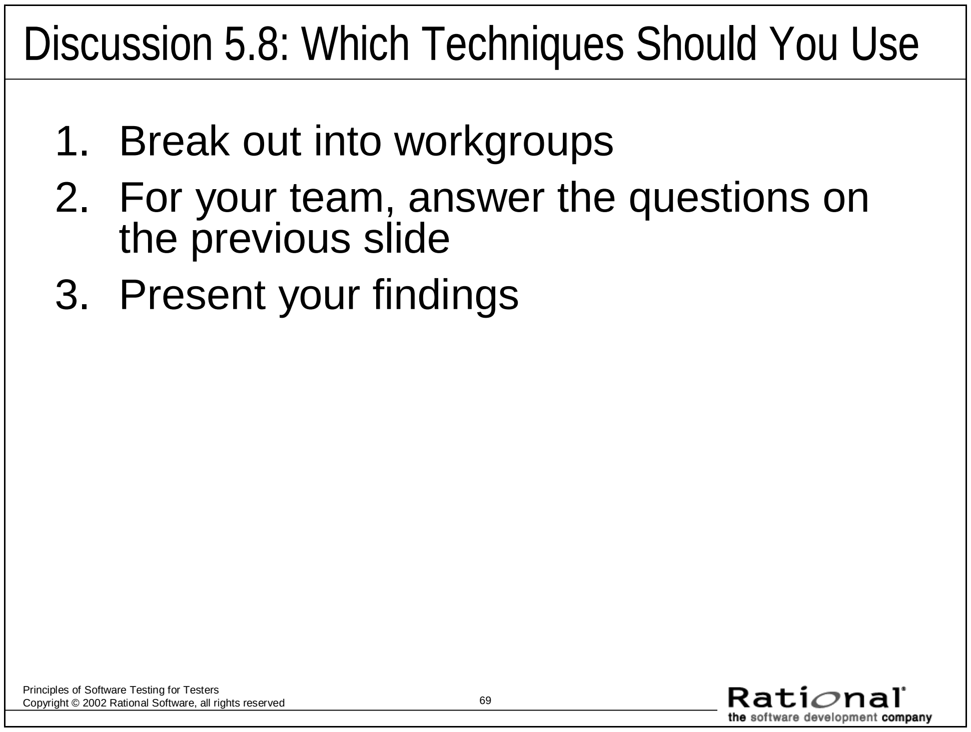
How To Adopt New Techniques/Review



How To Adopt New Techniques

1. Answer these questions:
   - What techniques do you use in your test approach now?
   - What is its greatest shortcoming?
   - What **one** technique could you add to make the greatest improvement, consistent with a good test approach:
     - Risk-focused?
     - Product-specific?
     - Practical?
     - Defensible?
2. Apply that additional technique until proficient
3. Iterate

Discussion 5.8: Which Techniques Should You Use



Discussion 5.8: Which Techniques Should You Use

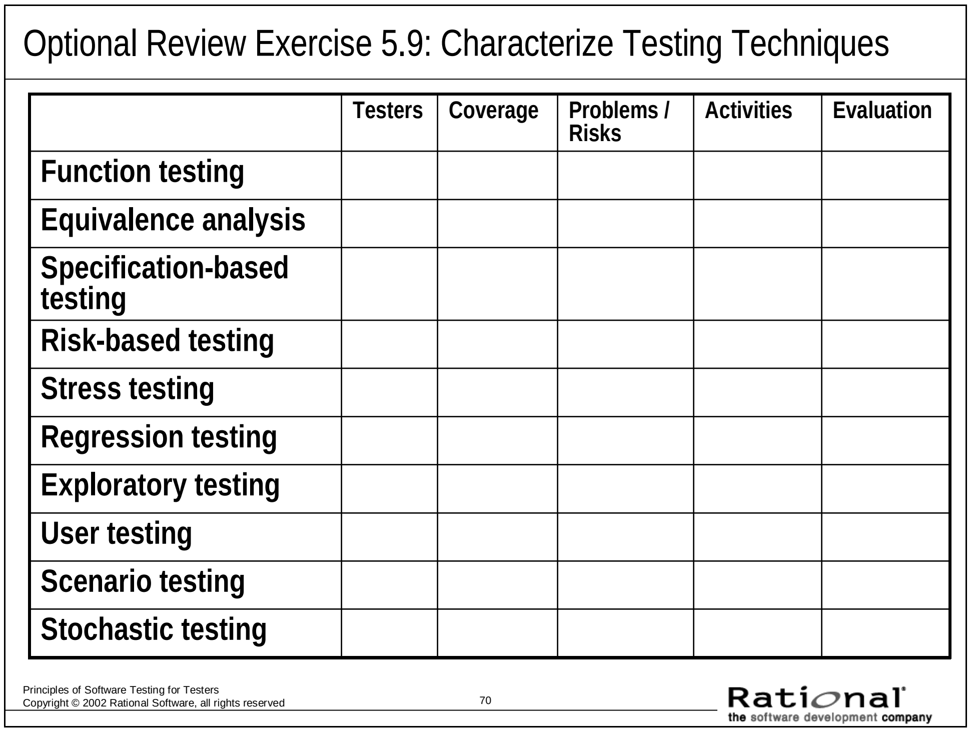
1. Break out into workgroups
2. For your team, answer the questions on the previous slide
3. Present your findings

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved

69

Rational
the software development company

Optional Review Exercise 5.9: Characterize Testing Techniques



Optional Review Exercise 5.9: Characterize Testing Techniques

	Testers	Coverage	Problems / Risks	Activities	Evaluation
Function testing					
Equivalence analysis					
Specification-based testing					
Risk-based testing					
Stress testing					
Regression testing					
Exploratory testing					
User testing					
Scenario testing					
Stochastic testing					

Principles of Software Testing for Testers

70
Rational
the software development company

Optional Take-home Exercise

- **Go back through the testing techniques and characterize the key traits of each.**
- **Which techniques do you use on your current project(s)?**
- **Which would you try next?**
- **Why?**