Module 4
Define Evaluation Mission

Principles of Software Testing for Testers

Module 4: Define Evaluation Mission

Topics

Module 4 Agenda

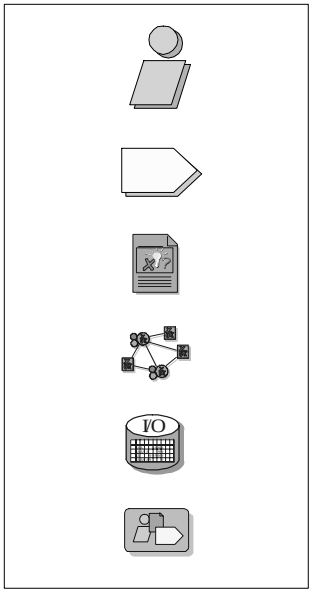
Module 4 Agenda

- **Definition of the workflow:**
  Define Evaluation Mission
- Defining the mission of the test group
- Defining the goal for test documentation

Review: Where We've Been

Review: Where We've Been

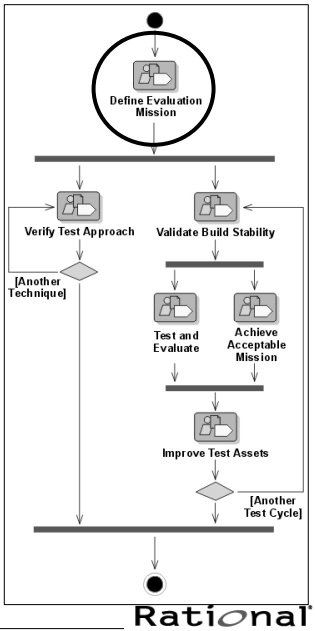
- In the introductory module we discussed the concepts of *quality* and *test ideas*
- In the last module, we introduced some of the basic elements in the RUP Test content
- We'll use those basic RUP elements throughout the remainder of the course to help provide context for what we'll learn.



Define Evaluation Mission Workflow

Define Evaluation Mission

- In this module, we begin with 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)?



4
Rational
the software development company

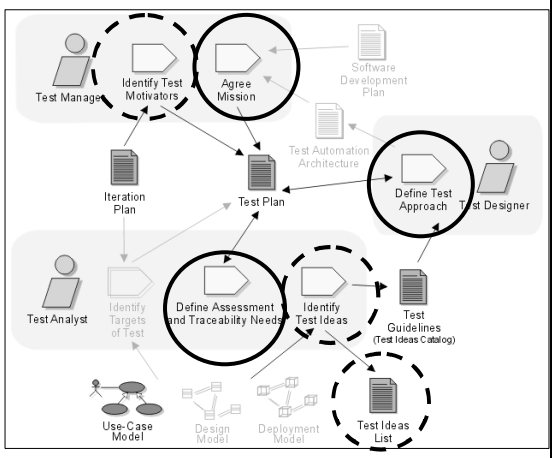
The purpose of this workflow detail is to:

- Identify the appropriate focus of the test effort for the iteration.
- Gain agreement with stakeholders on the corresponding goals that will direct the test effort
- For each iteration, work is focused mainly on:
  - Identifying the objectives for, and deliverables of, the testing effort
  - Identifying a good resource utilization strategy
  - Defining the appropriate scope and boundary for the test effort
  - Outlining the approach that will be used
  - Defining how progress will be monitored and assessed.

Define Evaluation Mission

Define Evaluation Mission

- This module focuses on the activities that capture the goals of our testing efforts.
- We will look at different *Missions* that test teams use, and consider the implications on the corresponding *Test Approach* those teams take.
- These are the activities that create the *Test Plan*.



5
Rational the software development company

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

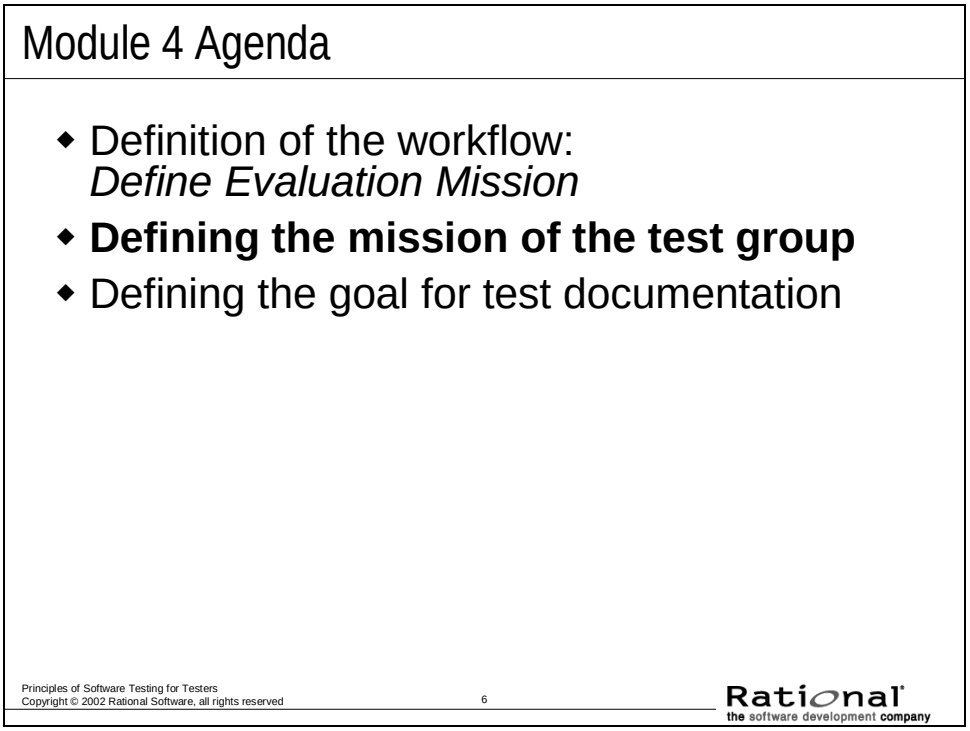
In earlier modules, we discussed how identifying test ideas is a useful way to reason about tests early in the lifecycle without needing to completely define each specific test. We also looked at some of the basic elements that are used to define the Rational Unified Process.

In this module we'll look at a how different test teams need to use different evaluation missions, depending on their specific context.

In the next module, we'll talk more about applying different techniques in our tests effort.

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

Define the Mission of the Test Group



Exercise 4.1: Which Group is Better?

Exercise 4.1: Which Group is Better?

Testing Group 1

	Found pre-release
Function A	100
Function B	0
Function C	0
Function D	0
Function E	0
Total	100

Testing Group 2

Function A	50
Function B	6
Function C	6
Function D	6
Function E	6
Total	74

From Marick, *Classic Testing Mistakes*

Two groups test the same program.

- The functions are equally important
- The bugs are equally significant

Principles of Software Testing for Testers

7
Rational the software development company

Imagine giving the same product to two completely independent test groups. The product has lots of functions, but you pick out five that you consider the most important:

- These 5 functions are equally important.
- You expect that on average, the bugs found in any one of the functions will be as serious or significant as bugs found in the others.

Test Group 1 starts with a broad test, trying all the functions with easy values. No obvious bugs show up in Functions B, C, D or E, but Function A is clearly broken from the start.

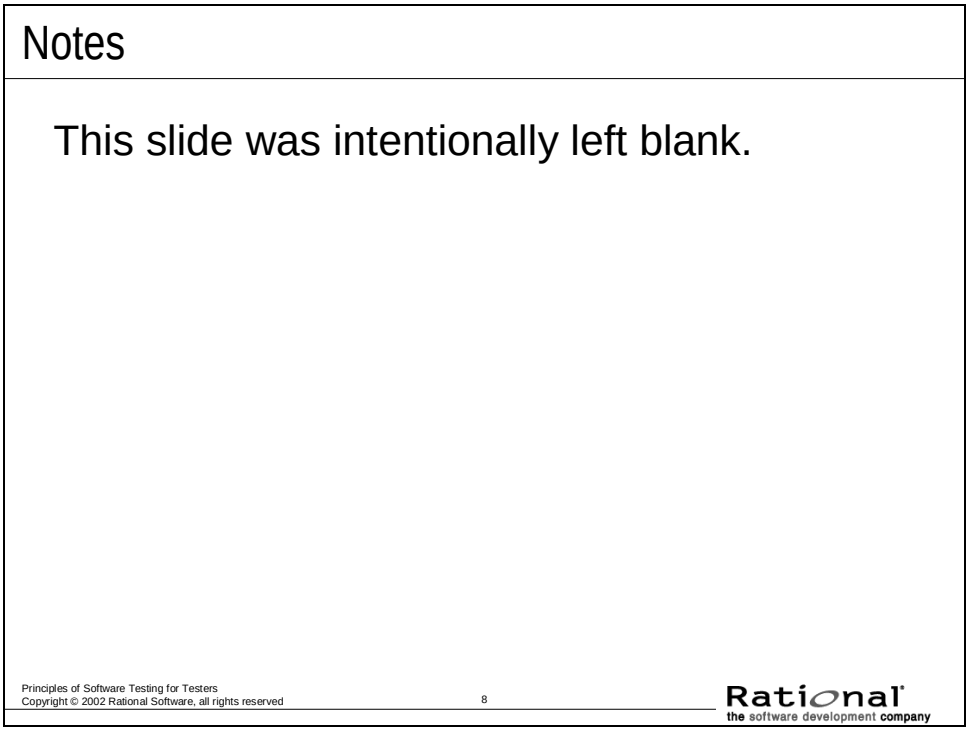
Over the next few weeks, Test Group 1 keeps testing the program. Their goal is to find lots of bugs. They spend a little more time on Functions B, C, D and E, but primarily focus on Function A, where it is all too easy to find bugs. Test Group 1 never finds any bug in Function B, C, D, or E, but they find 100 in Function A.

Test Group 2 starts with essentially the same broad test and finds the same result. Function A appears to be full of bugs. As to B, C, D, and E, no obvious bugs show up in the first round of testing.

Test Group 2 follows a different testing strategy. They do a lot more testing of B, C, D, and E – even though they don't find as many bugs in these functions, because they want to achieve a certain baseline level of coverage. As a result, they find only 50 bugs in A and a few bugs in B, C, D, and E.

WHICH IS THE BETTER TESTING GROUP?

Notes



Your Notes:

Exercise 4.2: Which Group is Better?

Exercise 4.2: Which Group is Better?

	Found pre-release	Found later	Total
Function A	100	0	100
Function B	0	12	12
Function C	0	12	12
Function D	0	12	12
Function E	0	12	12
Total	100	48	148
Function A	50	50	100
Function B	6	6	12
Function C	6	6	12
Function D	6	6	12
Function E	6	6	12
Total	74	74	148

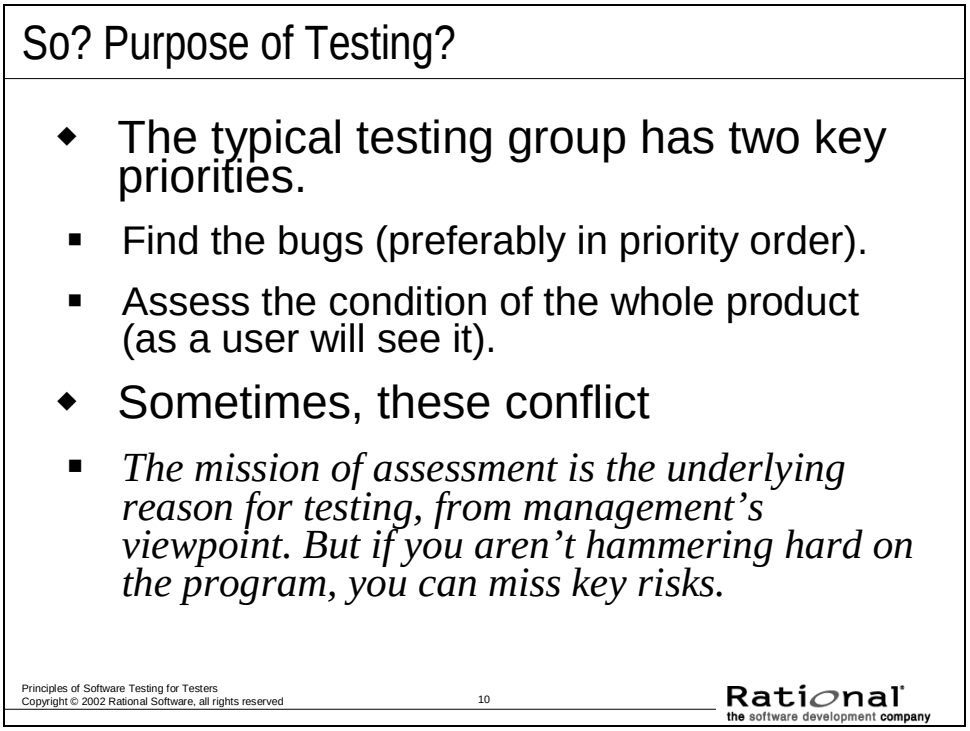
Here's some more data.

The company shipped the product. Six months later, we look at customer support call data and we see a bunch of new bugs found by customers.

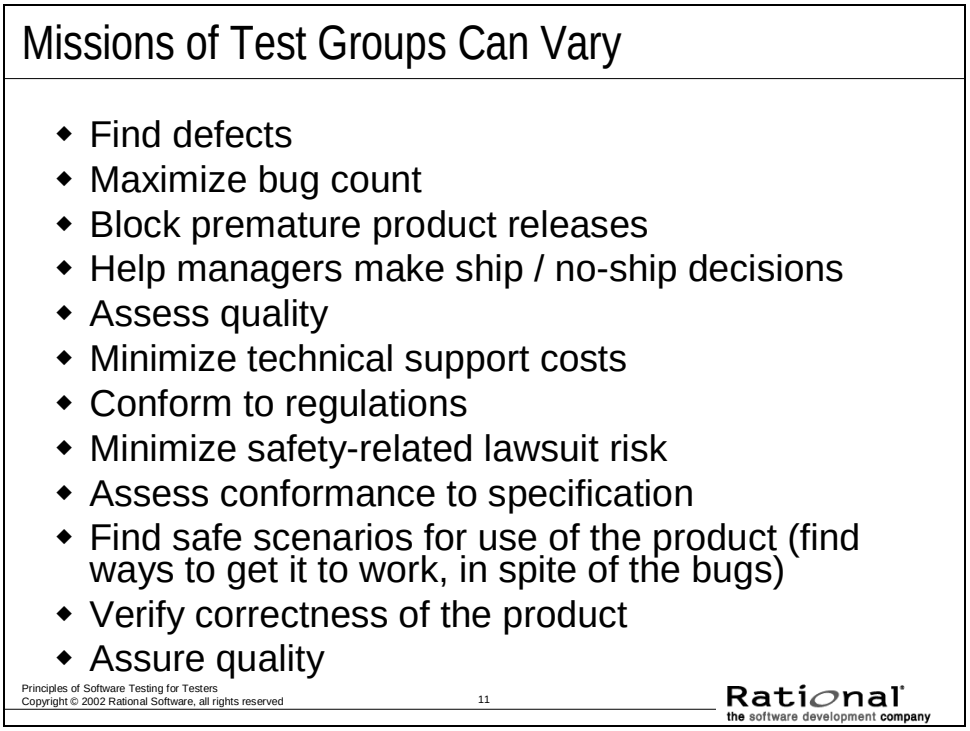
- The first group found all the bugs in Function A but missed 48 bugs in B,C, D and E.
- The second group found half the bugs of each function.
- The first group found 100 of the 148 bugs
- The second group found 74 of the 148 bugs, but they were more evenly distributed across functions.

WHICH GROUP IS BETTER? WHY?

So? Purpose of Testing?



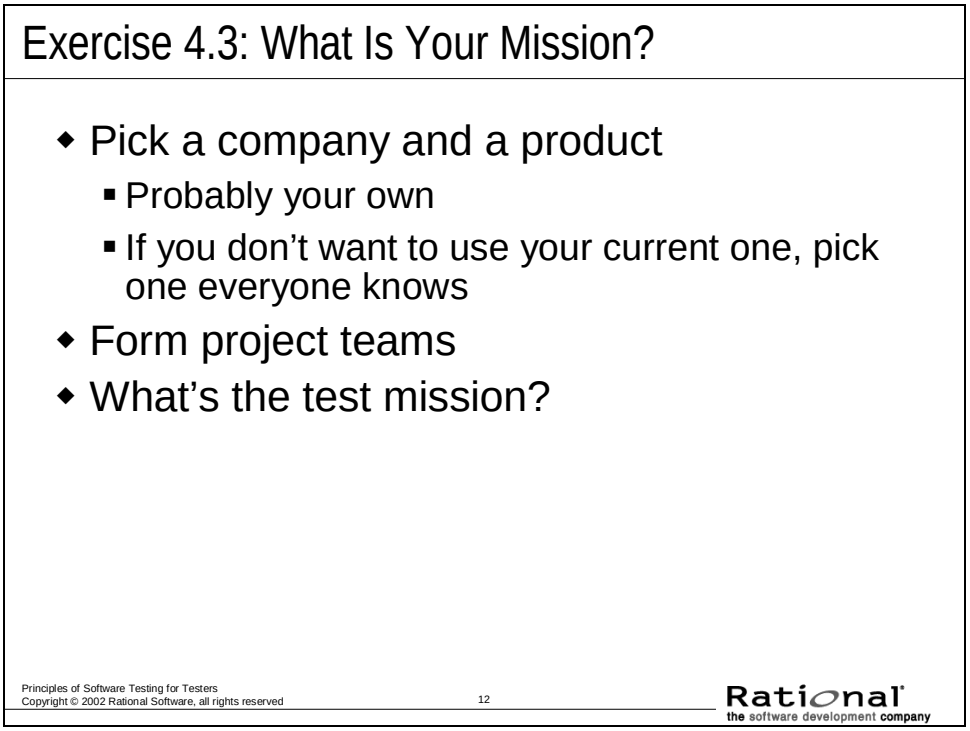
Missions of Test Groups Can Vary



Even if the test group has an overall mission, its objectives will vary over the life of the project. For example, a group whose primary role was defect-hunting through most of the project might be expected to provide quality evaluations as the project gets closer to its planned release date.

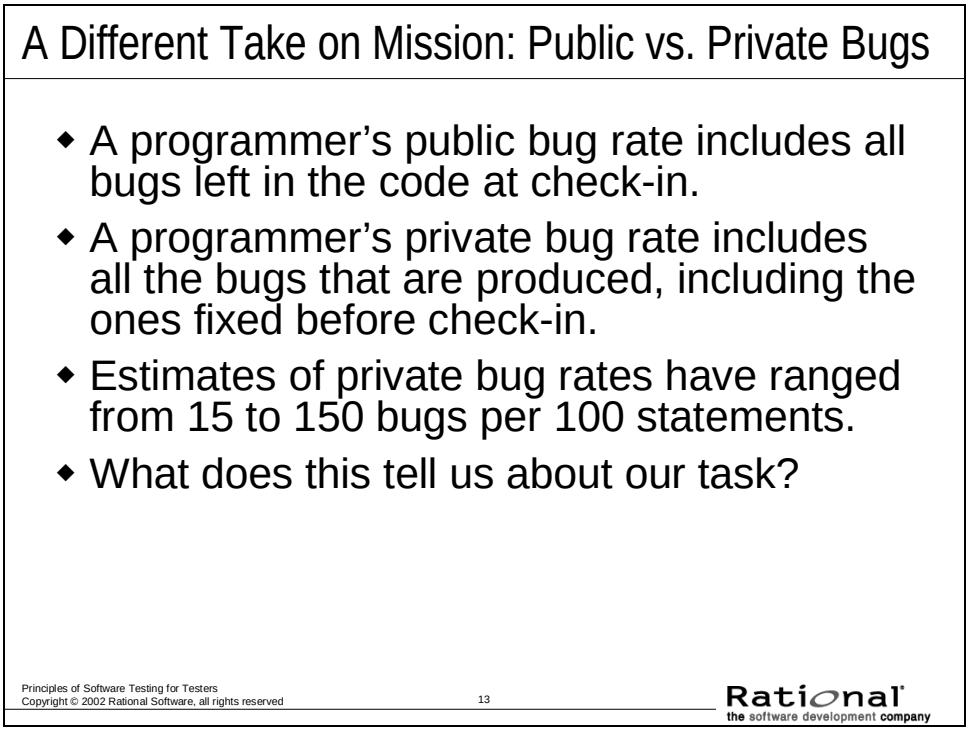
It is important for the test group to decide its guiding objectives for each iteration, and to reassess these as part of the preparation for each iteration.

Exercise 4.3: What Is Your Mission?



Optional Exercise

A Different Take on Mission: Public vs. Private Bugs



A programmer's public bug rate includes all bugs left in the code when it is given to someone else (such as a tester.) Rates of one bug per hundred statements are not unusual, and several programmers' rates are higher (such as three bugs per hundred).

A programmer's private bug rate includes all the bugs that are produced, including the ones already fixed before passing the program to testing.

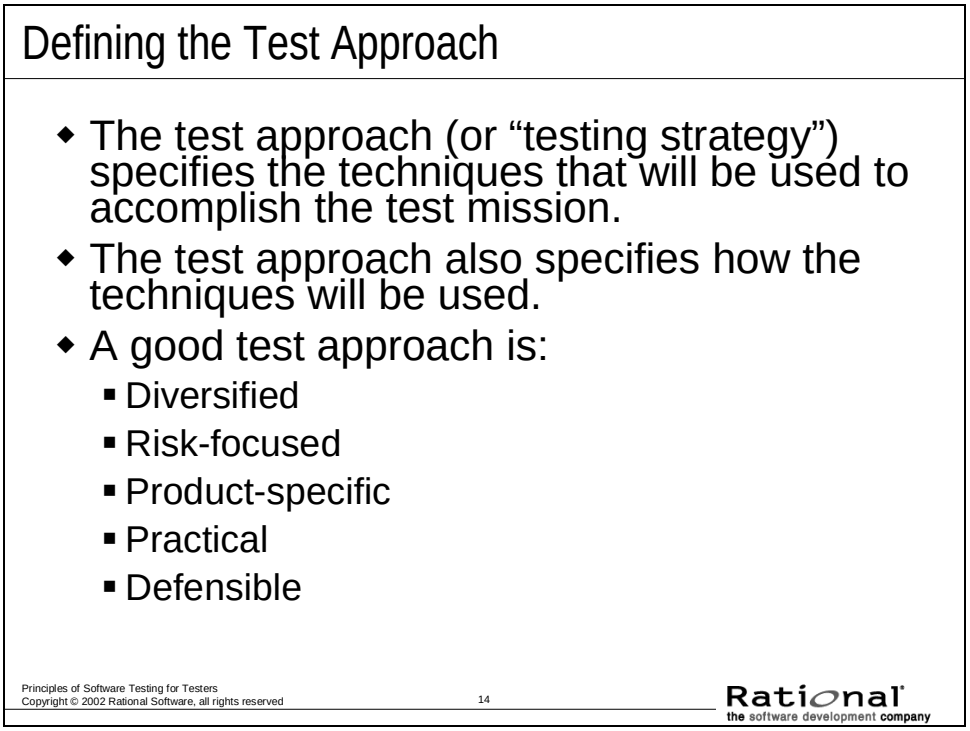
Estimates of private bug rates have ranged from 15 to 150 bugs per 100 statements. Therefore, programmers must be finding and fixing between 80% and 99.3% of their own bugs before their code goes into test. (Even the sloppy ones find and fix a lot of their own bugs.)

What does this tell us about our task?

It says that we're looking into the programmer's (and tools') blind spots. Merely repeating the types of tests that the programmers did won't yield more bugs. That's one of the reasons that an alternative approach is so valuable.

Conclusion: Unless the tester's methods are different from the programmer's, the tester will be going over already well tested grounds.

Defining the Test Approach



Diversified. Include a variety of techniques. Each technique is tailored to expose certain types of problems, and is virtually blind to others. Combining them allows you to find problems that would be hard to find if you spent the same resource on a narrower collection of techniques.

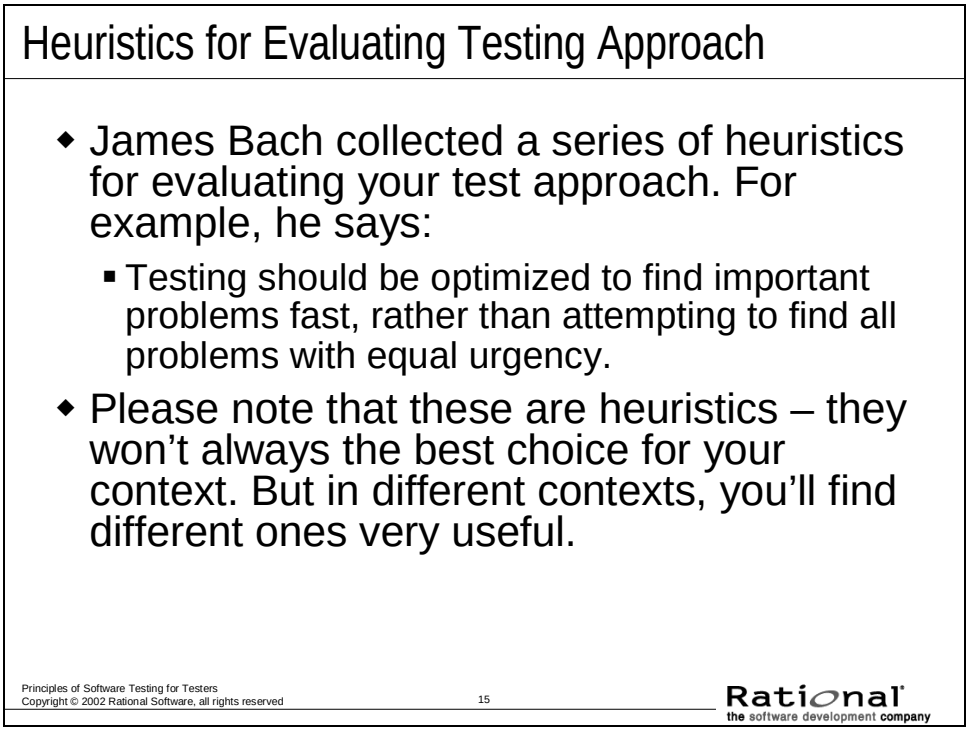
Risk-focused. Tests give you the opportunity to find defects or attributes of the software that will disappoint, alienate, or harm a stakeholder. You can’t run all possible tests. To be efficient, you should think about the types of problems that are plausibly in this product or that would make a difference if they were in this product, and make sure that you test for them.

Product-specific. Generic test approaches don’t work. Your needs and resources will vary across products. The risks vary across products. Therefore the balance of investment in different techniques should vary across products.

Practical. There’s no point defining an approach that is beyond your project’s capabilities (including time, budget, equipment, and staff skills). For example, you won’t be likely to succeed if you try to build a fully automated test plan if you have a team full of non-programmers.

Defensible. Can you explain and justify the work that you are doing? Does your approach allow you to track and report progress and effectiveness? If you can’t report or justify your work, are you likely to be funded as well as you need?

Heuristics for Evaluating Testing Approach



A heuristic is a rule of thumb, a rule that is useful but not always correct.

Example of a heuristic

The test approach should focus most effort on areas of potential technical risk, while still putting some effort into low risk areas just in case the risk analysis is wrong.

Basis for this heuristic

Complete testing is impossible, so we have to select the tests to run. Ideally, we would run the tests that promise to provide the most useful information. However, no risk analysis is perfect. We have to put some effort into checking out the areas that appear to be low risk, just in case.

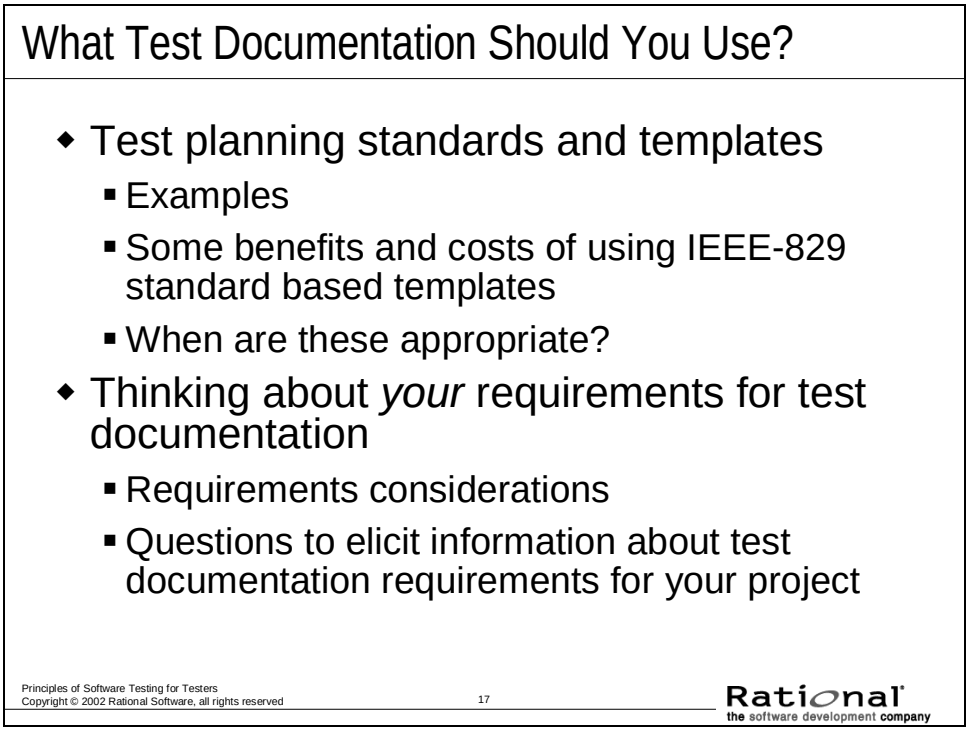
The full list is included as a table in the course textbook, *Lessons Learned in Software Testing*, pages 257-259.

Define the Goal for Test Documentation

Module 4 Agenda

- Definition of the workflow:
  Define Evaluation Mission
- Defining the mission of the test group
- **Defining the goal for test documentation**

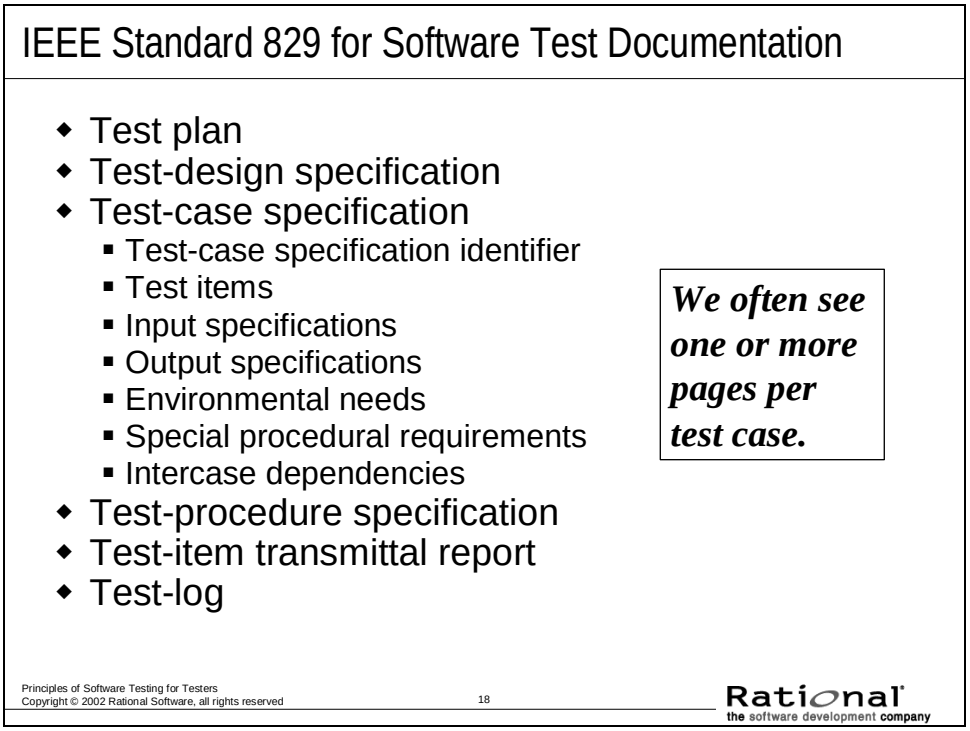
What Test Documentation Should You Use?



IEEE Standard 829 for software test documentation is a standard initially published by the Institute for Electrical and Electronics Engineers (1983) and later approved by the American National Standards Institute.

The standard describes a wide range of types of information that can be included in test documentation. For examples, see the next slide.

IEEE Standard 829 for Software Test Documentation



In the course text, *Lessons Learned in Software Testing*, there are two conflicting lessons:

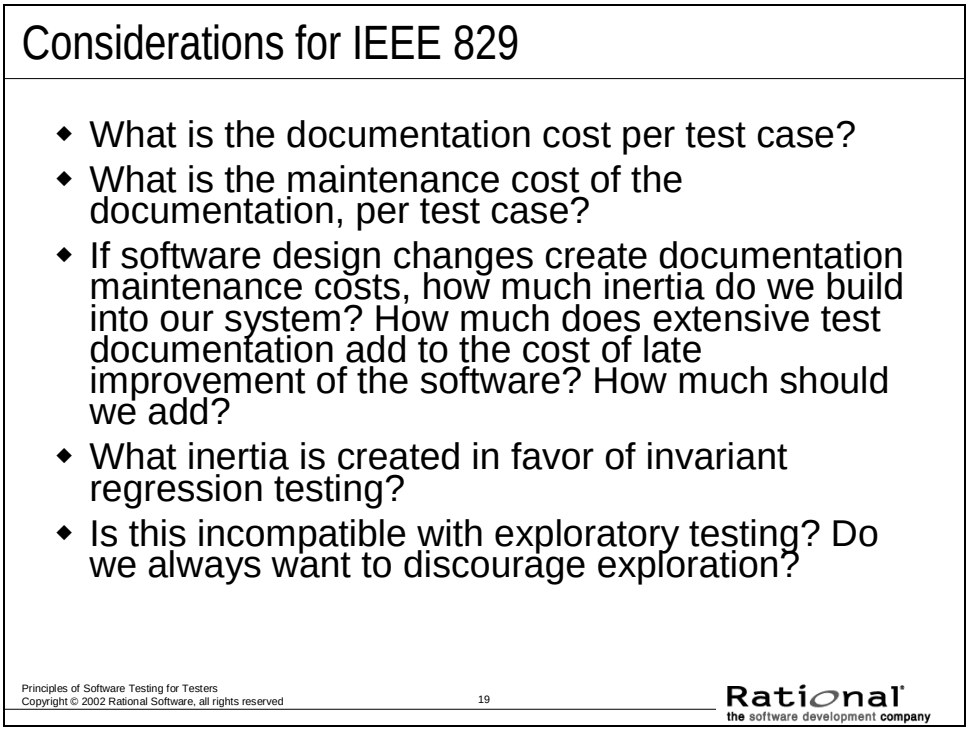
> *Lesson 145: Use the IEEE Standard 829 for test documentation.*
>
> *Lesson 146: Don't use the IEEE Standard 829.*

These two lessons contrast circumstances under which the standard is appropriate and under which it is not.

A critical point to recognize here is that test documentation is not free and can be very expensive.

It is common to see a full page of documentation for a simple test case and many pages for complex test cases. It probably takes 1 to 8 hours to write a page of test documentation (Technical writers take about 8 hours per page on software user manuals, when you include the research, writing formatting and editing time. For more on the productivity of tech writers, see JoAnn Hackos, *Managing Your Documentation Projects*, Wiley.)

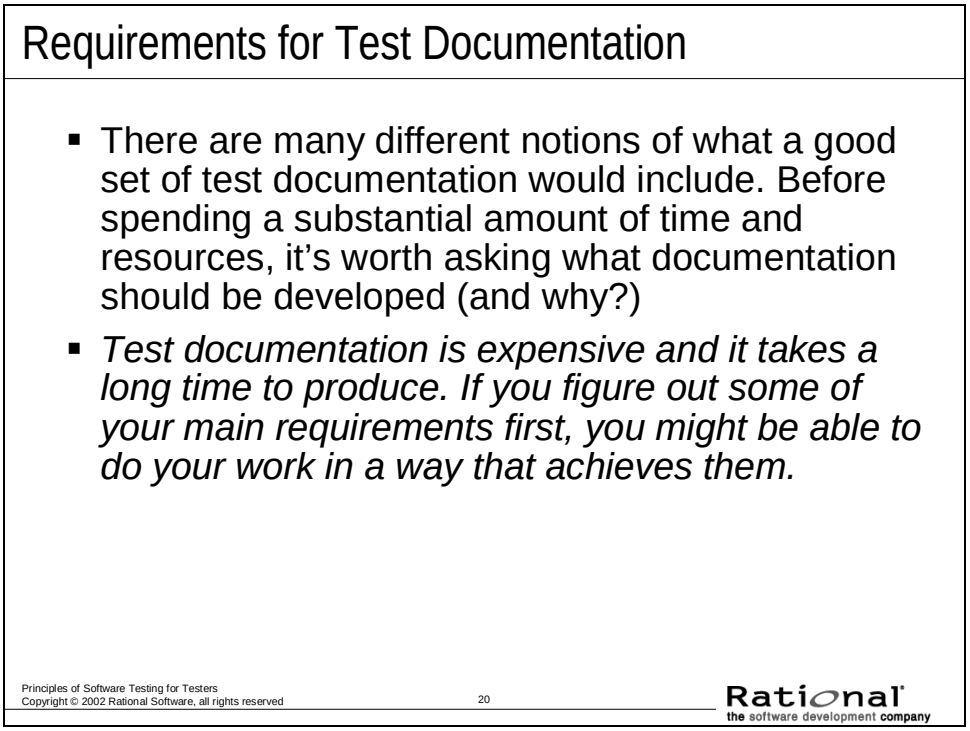
Considerations for IEEE 829



Some other questions to consider are:

- What is the impact on high-volume test automation? If the documentation cost per test case is high, how can you afford to create a multi-million test case project?
- 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 are filled by following a template based on 829?
- Which stakeholders gain a net benefit from IEEE standard documentation?
- What benefits do they gain and why are those benefits important to them?

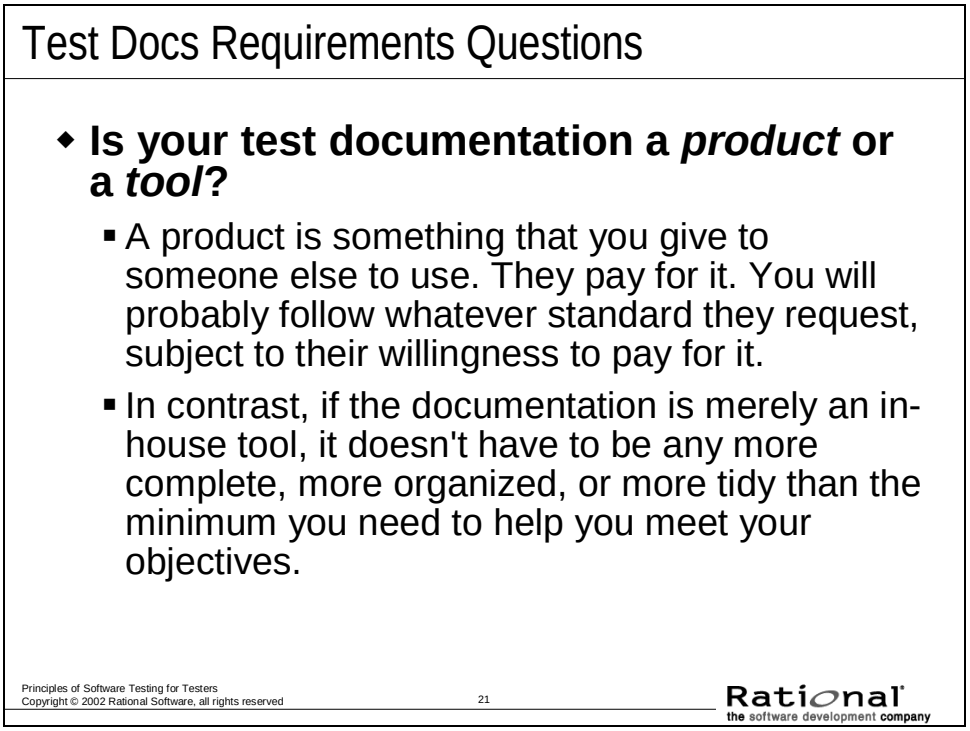
Requirements for Test Documentation



Lessons Learned in Software Testing provides 18 questions (page 136-140) that you can use to guide your analysis of your test documentation requirements.

We'll consider one example on the next slide.

Test Docs Requirements Questions



Here are some additional examples:

- Is software quality driven by legal issues or by market forces?
- How quickly is the design changing?
- How quickly does the design 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?
- If the documentation controls 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 documents (specifications or requirements) are you tracing back to and who controls them?

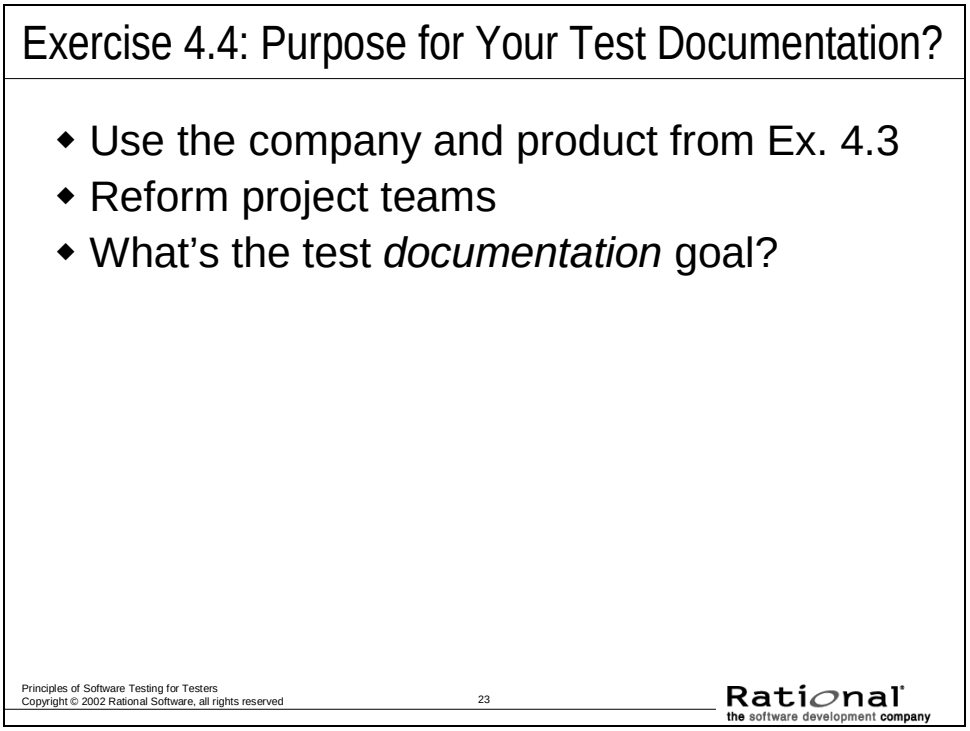
Write a Purpose Statement for Test Documentation

Write a Purpose Statement for Test Documentation

- 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.

22
Rational
the software development company

Exercise 4.4: Purpose for Your Test Documentation?



Optional Exercise

Regroup into your project teams and take ten minutes to discuss the exercise. Write down the answer, so you can share it with the group.

Review

Review: Define Evaluation Mission

- What is a Test Mission?
- What is your Test Mission?
- What makes a good Test Approach (Test Strategy)?
- What is a Test Documentation Mission?
- What is your Test Documentation Goal?