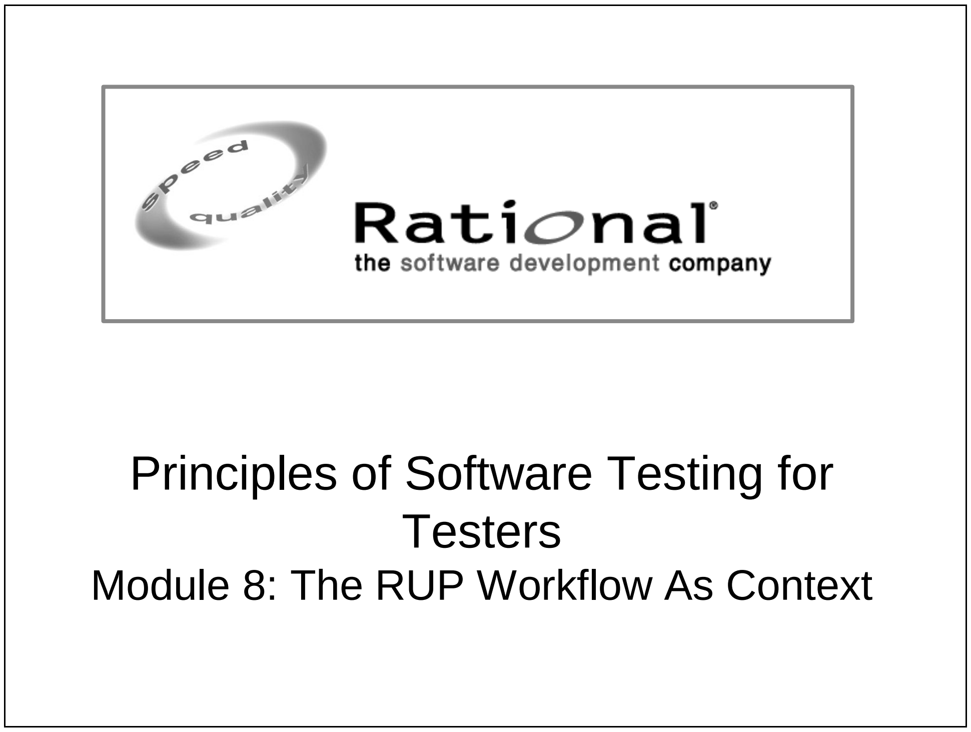
# Module 8
# The RUP Workflow As Context

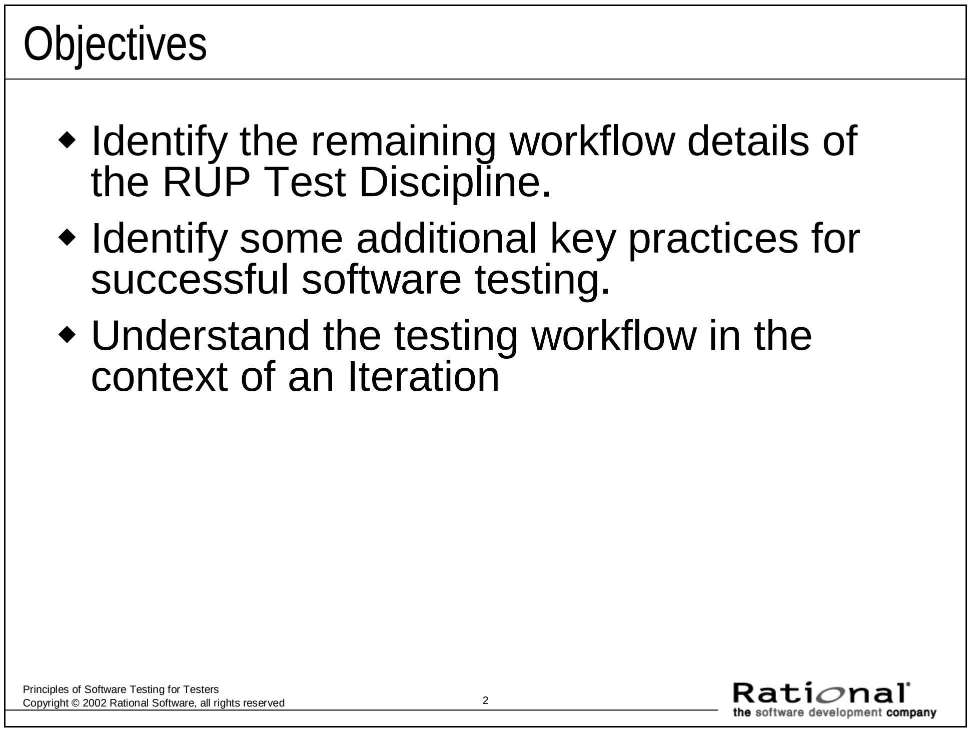


Principles of Software Testing for Testers

Module 8: The RUP Workflow As Context

## Topics

## Objectives

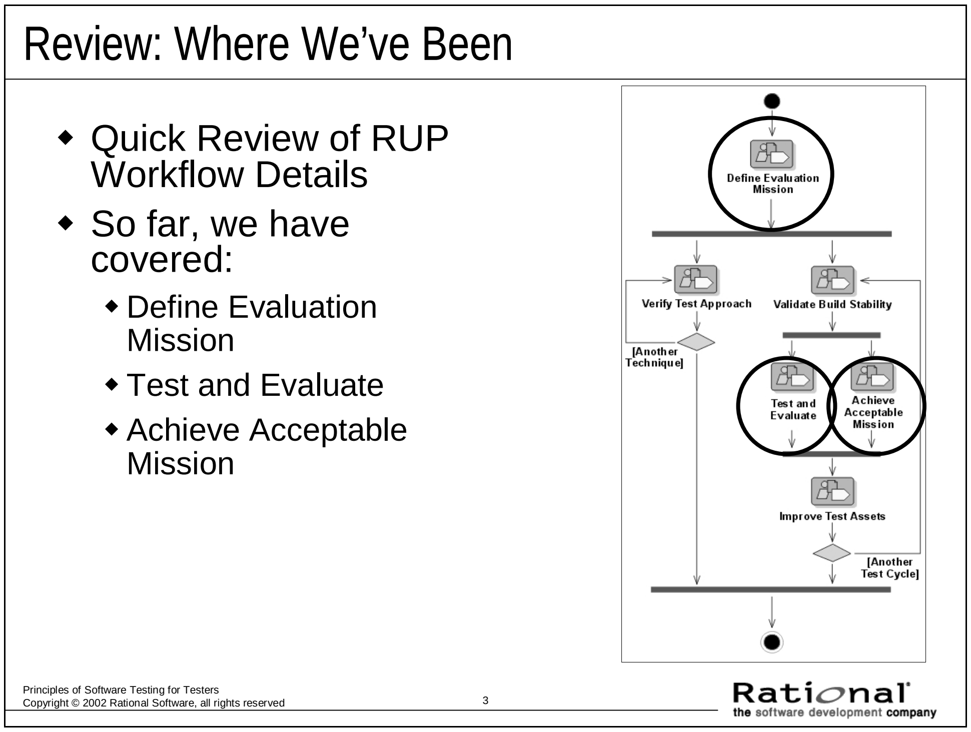


### Objectives

- Identify the remaining workflow details of the RUP Test Discipline.
- Identify some additional key practices for successful software testing.
- Understand the testing workflow in the context of an Iteration

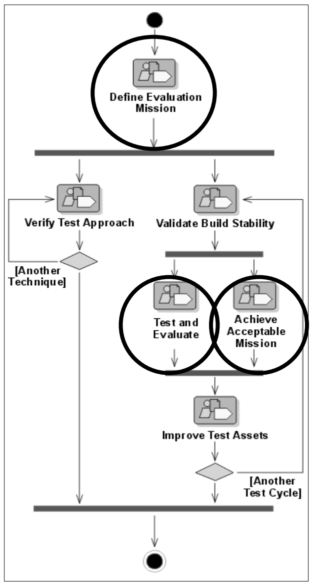
2

## Review: Where We've Been



### Review: Where We've Been

- Quick Review of RUP Workflow Details
- So far, we have covered:
  - Define Evaluation Mission
  - Test and Evaluate
  - Achieve Acceptable Mission



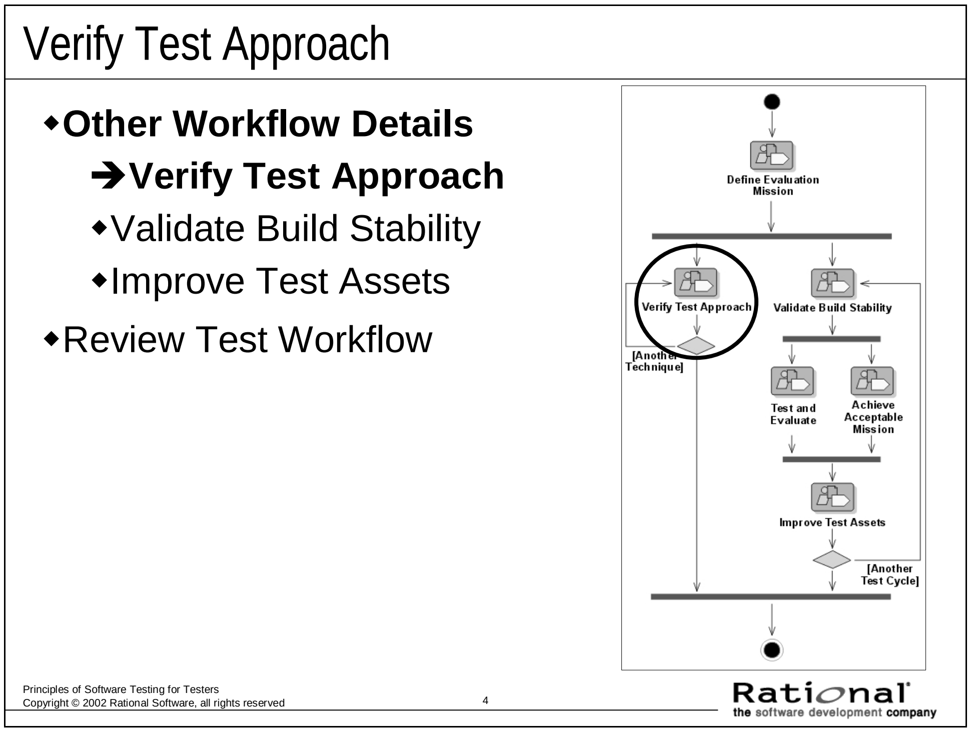
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 3

Rational® the software development company

In the previous modules, we discussed many of the core activities that test teams undertake.

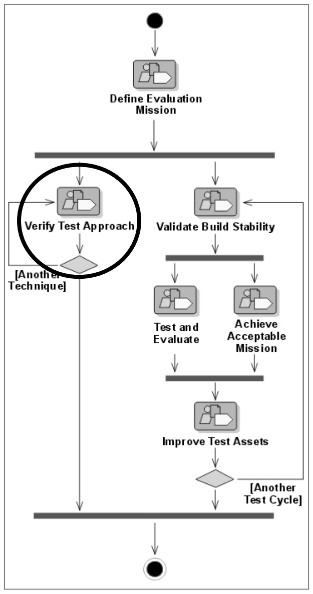
In this module, we'll talk about some additional practices to help make your iterative testing effort more successful.

## Verify Test Approach



### Verify Test Approach

- ◆**Other Workflow Details**
  - ➔**Verify Test Approach**
  - ◆Validate Build Stability
  - ◆Improve Test Assets
- ◆Review Test Workflow



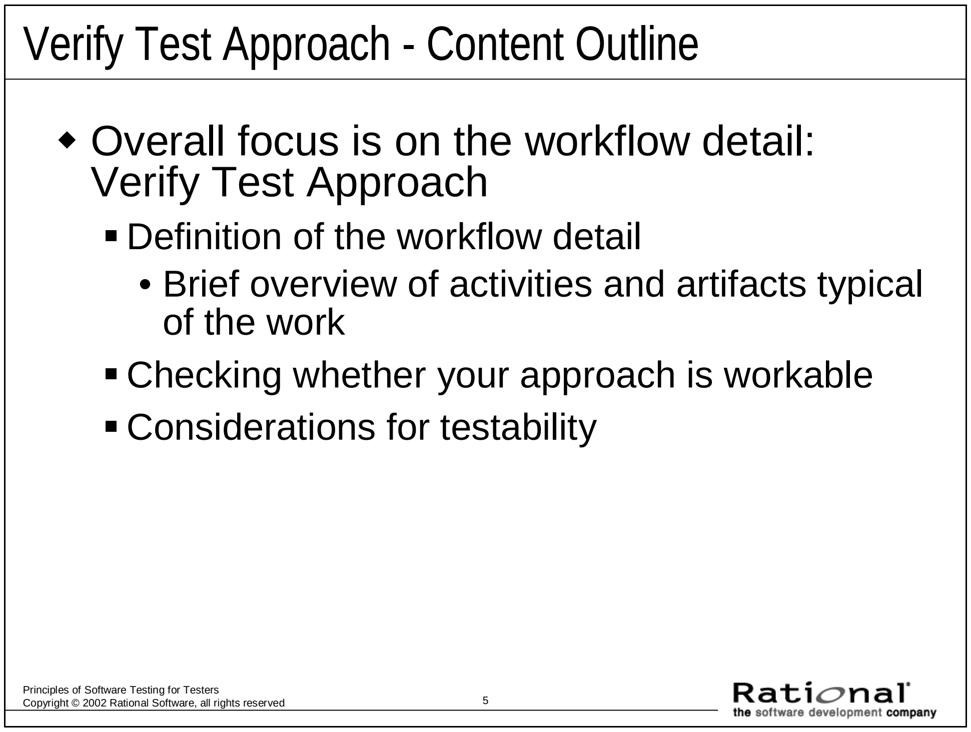
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved
4
Rational the software development company

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 iteration, this work focuses on:

- Early verification that Test Approach will work
- Establishing the supporting infrastructure
- Obtaining the required testability
- Identifying the scope, boundaries, limitations and constraints of each technique

## Verify Test Approach - Content Outline



### Verify Test Approach - Content Outline

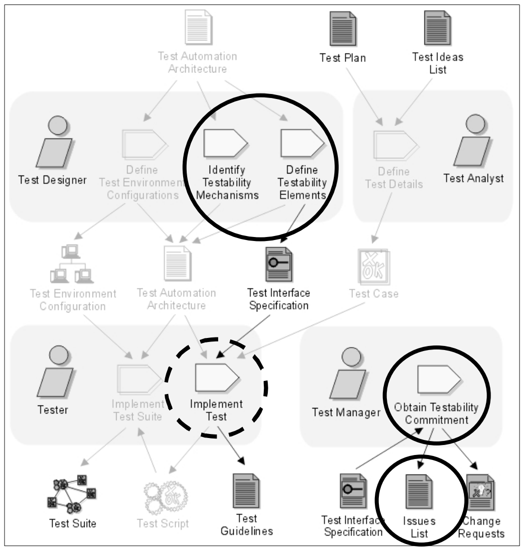
- Overall focus is on the workflow detail: Verify Test Approach
  - Definition of the workflow detail
    - Brief overview of activities and artifacts typical of the work
  - Checking whether your approach is workable
  - Considerations for testability

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved
5

## Verify Test Approach – Activities and Artifacts

### Verify Test Approach – Activities and Artifacts

- This section focuses on proving that the approach you plan to use is workable.
- We will spend time looking at activities that support the concept of *Testability.*
- Note that it is a good practice to implement tests as concrete proof that the approach will work.


Test Automation Architecture
Test Plan
Test Ideas List
Test Designer
Define Test Environment Configurations
Identify Testability Mechanisms
Define Testability Elements
Define Test Details
Test Analyst
Test Environment Configuration
Test Automation Architecture
Test Interface Specification
Test Case
Tester
Implement Test Suite
Implement Test
Test Manager
Obtain Testability Commitment
Test Suite
Test Script
Test Guidelines
Test Interface Specification
Issues List
Change Requests

6

Rational
the software development company

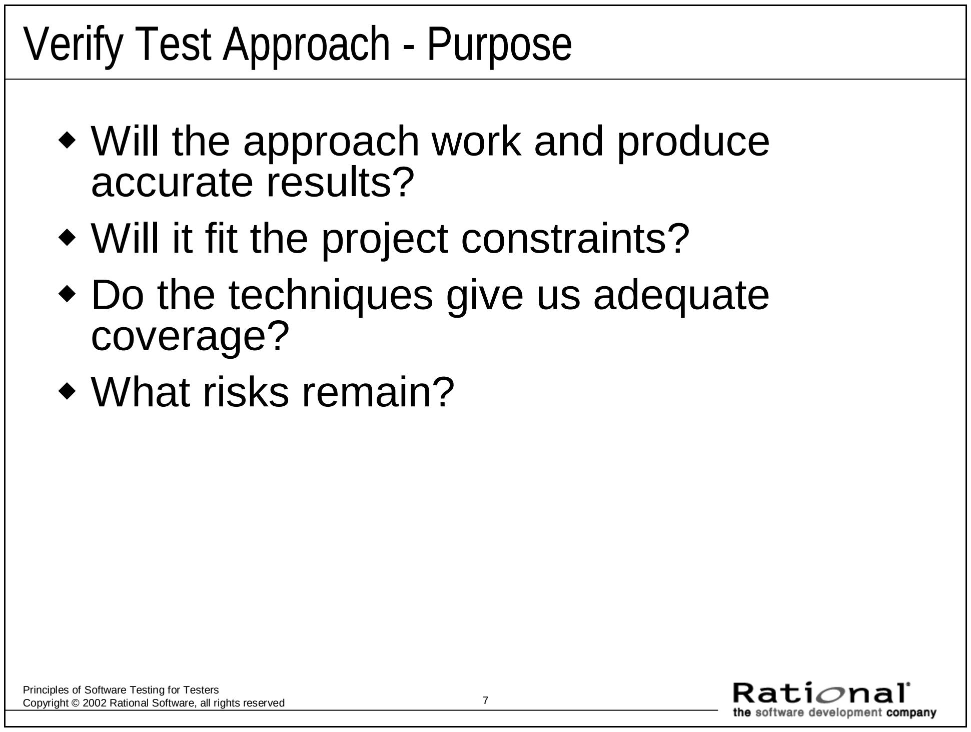
Here are the roles, activities and artifacts RUP focuses on in this work.

In the previous modules, we discussed many of the core activities that test teams undertake.

In this module, we'll talk about considerations for ensuring your test approach is appropriate and workable.

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

## Verify Test Approach - Purpose



### Verify Test Approach - Purpose

- Will the approach work and produce accurate results?
- Will it fit the project constraints?
- Do the techniques give us adequate coverage?
- What risks remain?

Principles of Software Testing for Testers

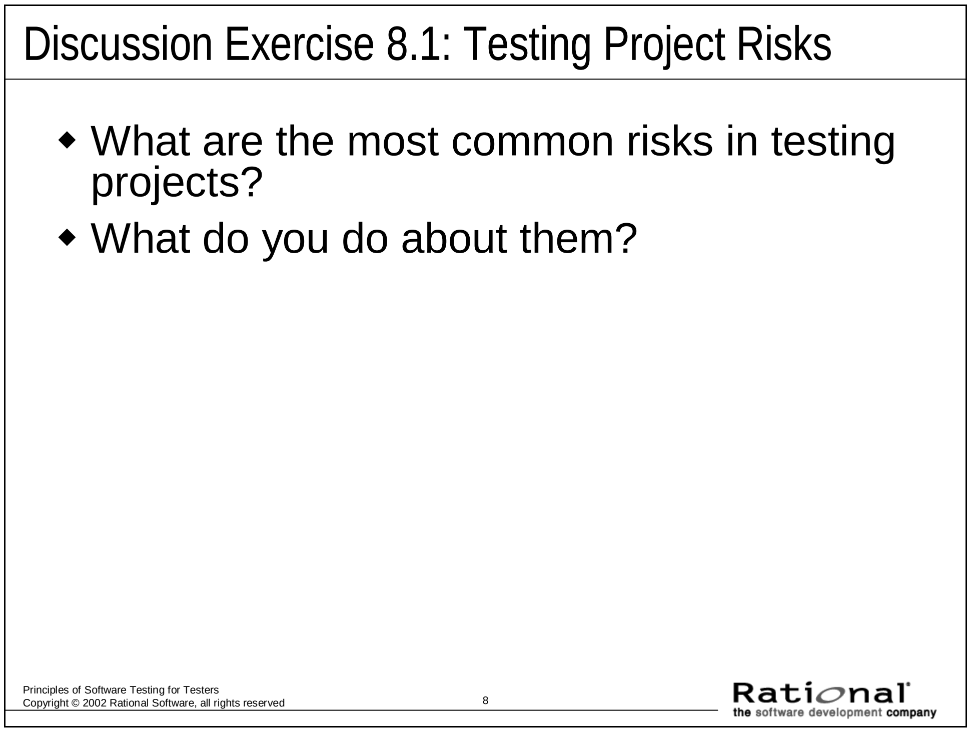
7

Rational the software development company

Show that the various techniques outlined in the Test Approach will support the required testing. It is useful to verify by demonstration that the approach will work, produce accurate results and is appropriate for the available resources.

The objective is to gain an understanding of the constraints and limitations of each technique, and to either find an appropriate implementation solution for each technique or find alternative techniques that can be implemented. This helps to mitigate the risk of discovering too late in the project life-cycle that the test approach is unworkable.

## Discussion Exercise 8.1: Testing Project Risks



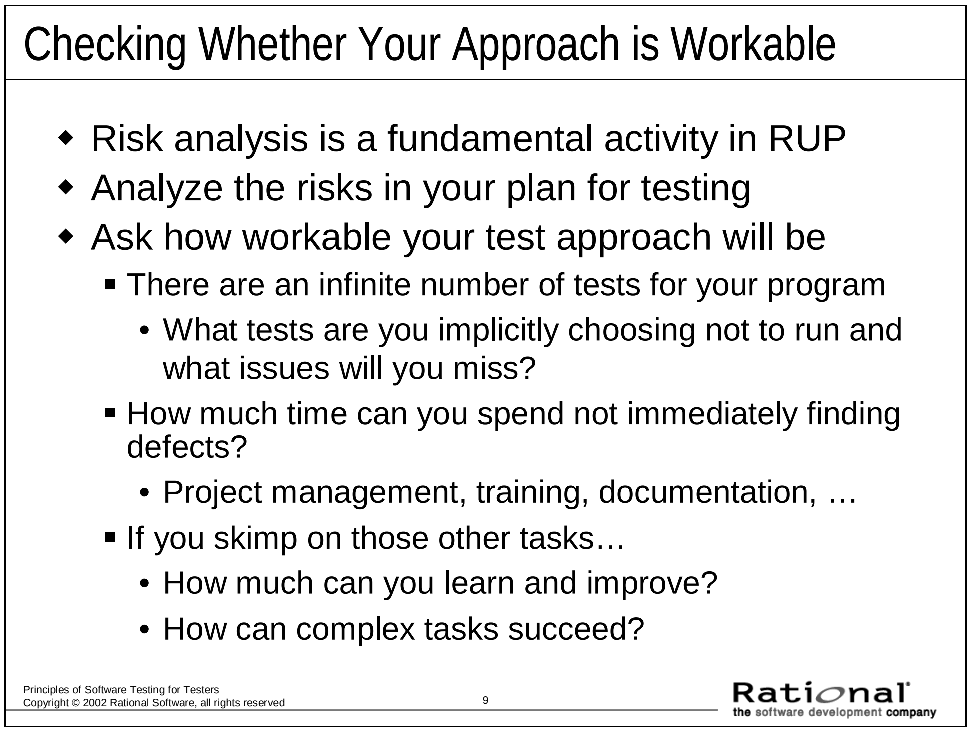
### Discussion Exercise 8.1: Testing Project Risks

- What are the most common risks in testing projects?
- What do you do about them?

8

Rational
the software development company

## Checking Whether Your Approach is Workable



### Checking Whether Your Approach is Workable

- Risk analysis is a fundamental activity in RUP
- Analyze the risks in your plan for testing
- Ask how workable your test approach will be
  - There are an infinite number of tests for your program
    - What tests are you implicitly choosing not to run and what issues will you miss?
  - How much time can you spend not immediately finding defects?
    - Project management, training, documentation, …
  - If you skimp on those other tasks…
    - How much can you learn and improve?
    - How can complex tasks succeed?

**Equipment availability**: Do you have what you need, for as long as you need it? You need:

- Equipment for compatibility testing
- Equipment capable of running the product
- Equipment of the kinds that have been high risk – resulted in many technical support problems – in the past
- Space (physical lab space, places for your to-be-hired staff to sit)

Do you have the budget for your equipment?

When will you order it?

If your equipment is "only" for compatibility testing, will you test briefly and then surrender it? If so, what if there are defects or relevant code changes? Can you get it back to retest it?

**Staff skills and training**

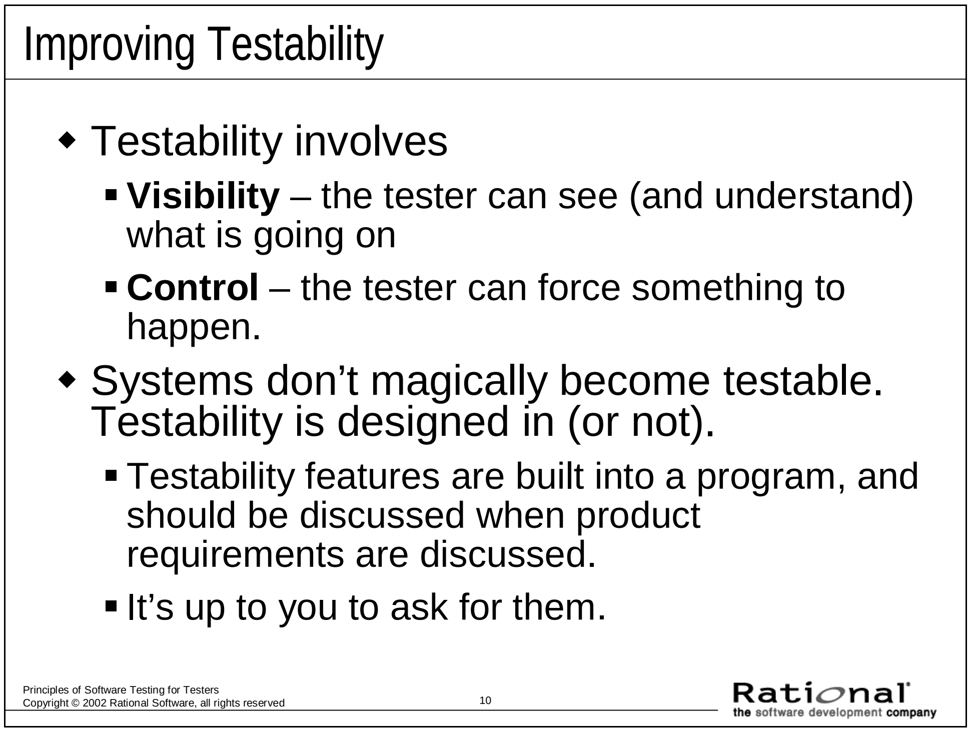
- If you expect them to rely on a tool, have them demonstrate competence with the tool early, in time to send them for training if necessary.
- If they just got back from training, have them demonstrate competence with the tool, on problems of the kind you have to solve.

**Information availability**: If your approach assumes the availability of artifacts from other people (e.g. models or specifications of certain features or activities)

- Do you know whether those artifacts are being created?
- Do you know when they will be available?
- Do you know how good (realistic, detailed, comprehensible) they are likely to be?

If the information is unavailable, what is your contingency plan?

## Improving Testability



### Improving Testability

- Testability involves
  - **Visibility** – the tester can see (and understand) what is going on
  - **Control** – the tester can force something to happen.
- Systems don't magically become testable. Testability is designed in (or not).
  - Testability features are built into a program, and should be discussed when product requirements are discussed.
  - It's up to you to ask for them.

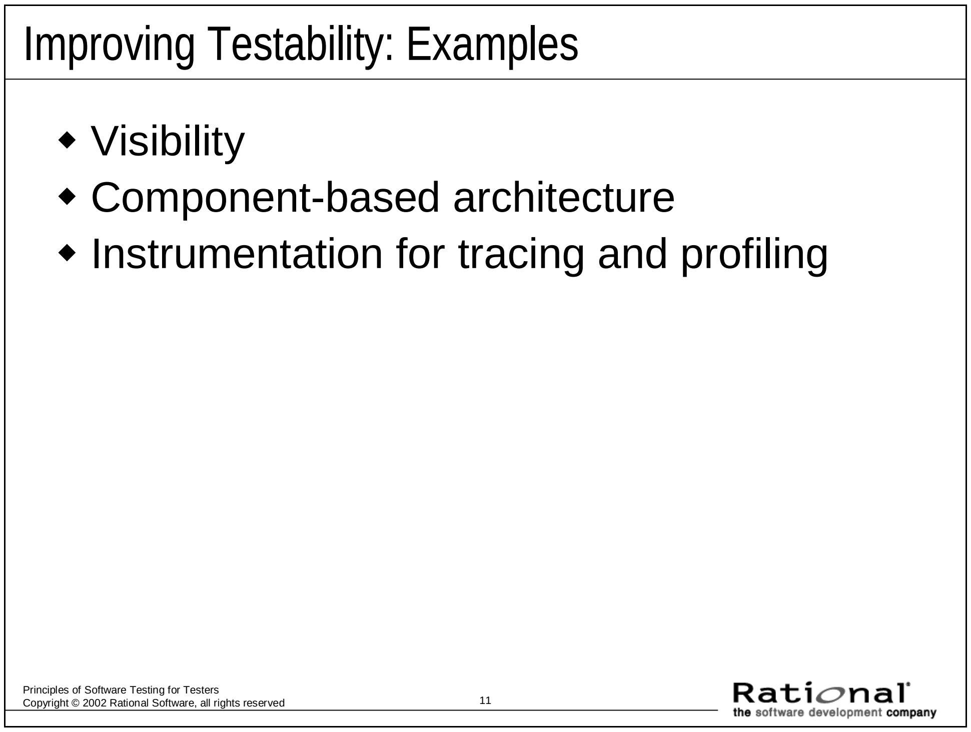
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 10

See *Lessons Learned*, Lesson 137, "**Testability is Visibility and Control**".

The discussion in RUP of testability looks at it from three different angles.

- First, there is the idea of improving testability by asking for testability features, early enough in the project that they can be designed in. Our question should be, what features would help us test this product much more efficiently?
- Second, there are the lesser improvements that you can ask for later in the project, when time is more constrained.
- (Third) Both of the preceding consider testability as an attribute of the product under test. We can also think in terms of our ability to test some aspect of the product, and review, item by item, what information we have and what we can get, what tactics are available to us, for testing that item.

## Improving Testability: Examples



### Improving Testability: Examples

- Visibility
- Component-based architecture
- Instrumentation for tracing and profiling

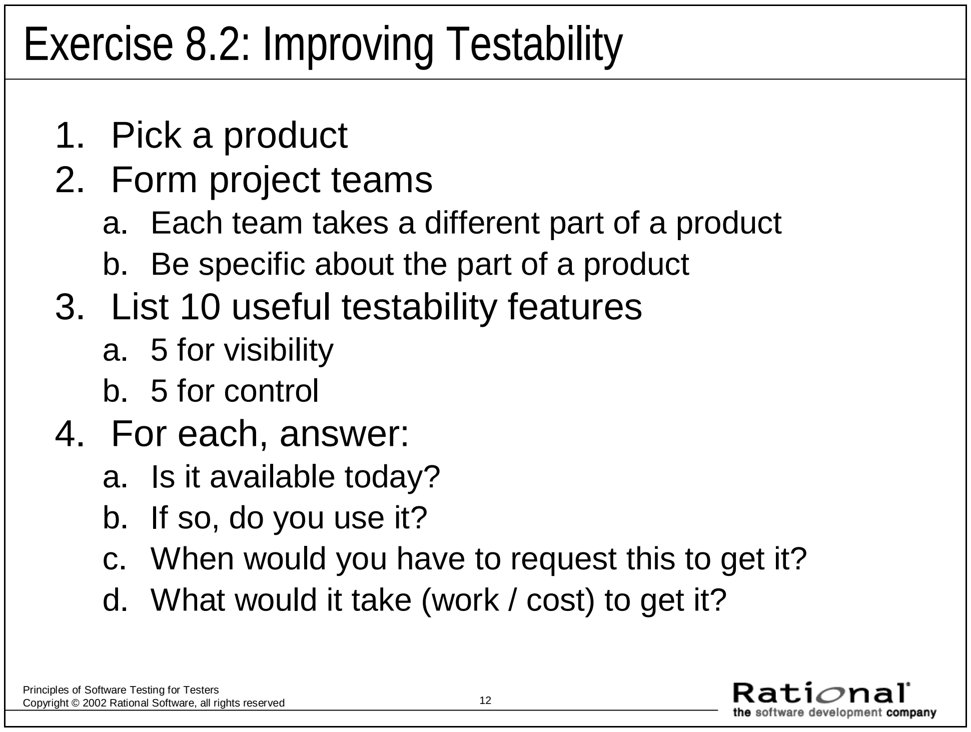
**Visibility:**

- Use standard UI controls, to facilitate GUI-level regression testing
- Unique error information for every error message. Let the tester put the program into a chatty (e.g. debug) mode. An error message reports the function that generated the error, plus additional information about the type of internal test that was failed, the variable that failed it, and any notes from the programmer about what this might mean.
- Trace logs

**Component-based architecture:**

- API layer, beneath the user interface, that gives the tester programmatic control over anything that the UI exposes to the customer. For example, if the UI has a feature that lets the customer add two numbers, the API would give access to the same additional call. The UI may change (location of the dialog, visual design, language, etc.) but the API will change much less often.
- Exposing interface contracts
- This lets the tester create automated tests of the underlying logic of the program, which will need less maintenance, and which will probably survive longer.

## Exercise 8.2: Improving Testability



### Exercise 8.2: Improving Testability

1. Pick a product
2. Form project teams
   a. Each team takes a different part of a product
   b. Be specific about the part of a product
3. List 10 useful testability features
   a. 5 for visibility
   b. 5 for control
4. For each, answer:
   a. Is it available today?
   b. If so, do you use it?
   c. When would you have to request this to get it?
   d. What would it take (work / cost) to get it?

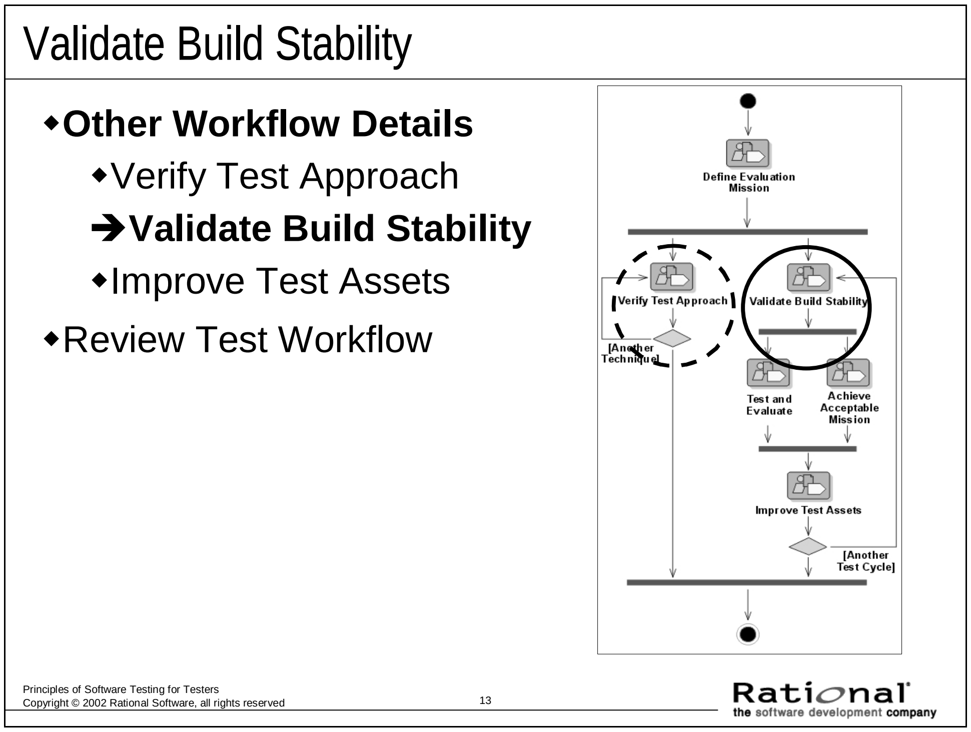
Principles of Software Testing for Testers

12

Rational
the software development company

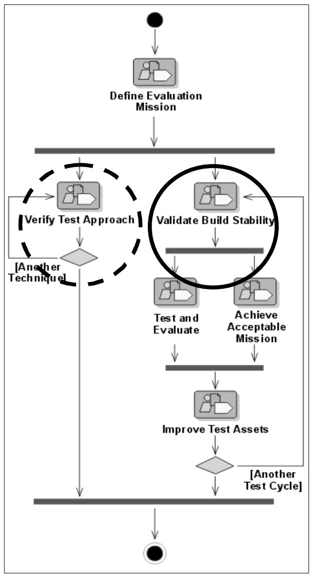
**Optional Exercise**

## Validate Build Stability



### Validate Build Stability

- ◆**Other Workflow Details**
  - ◆Verify Test Approach
  - ➔**Validate Build Stability**
  - ◆Improve Test Assets
- ◆Review Test Workflow



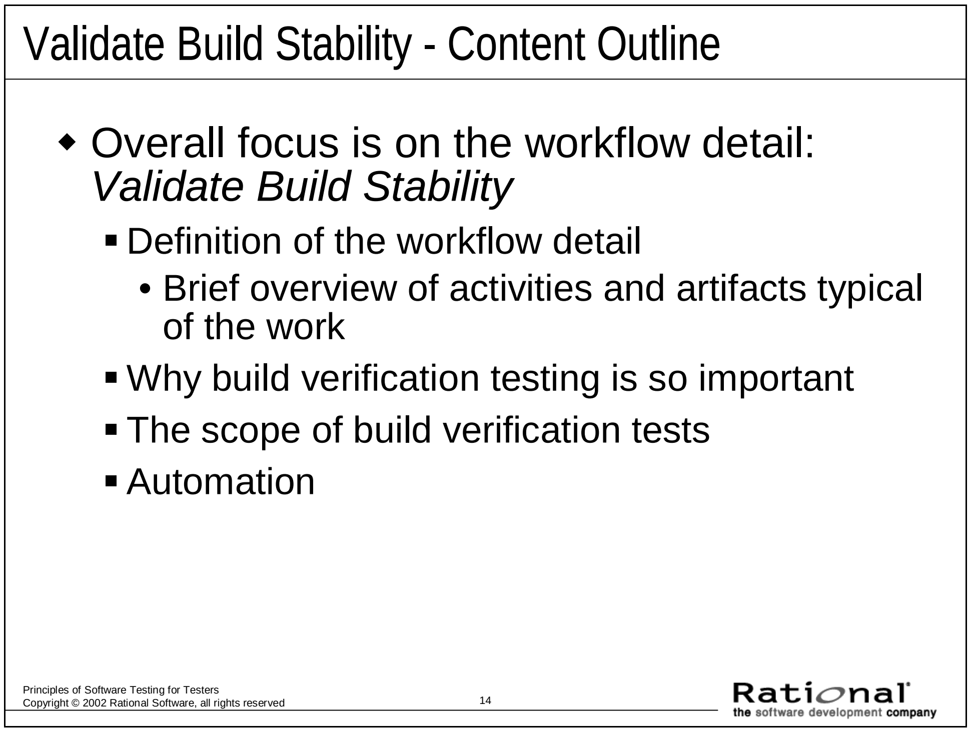
The purpose of this workflow detail is to Validate that the build can be tested / evaluated. This helps to prevent wasted testing effort.

For each build to be tested, this work focuses on:

- Assessing the stability and testability of the build
- Confirming the expectation of the development work delivered in the build
- Deciding to accept the build as worth further testing, guided by the evaluation mission
- If new build is rejected, current build is still used

This work is also referred to as a *smoke test, build verification test, build regression test, sanity check* or *acceptance into testing*.

## Validate Build Stability - Content Outline



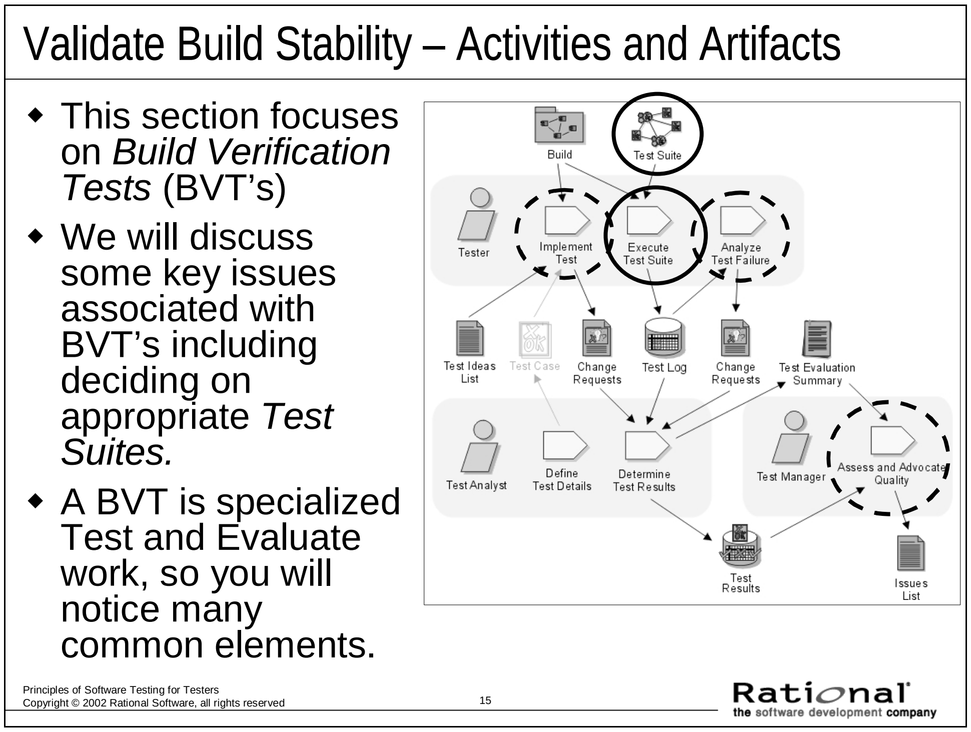
### Validate Build Stability - Content Outline

- Overall focus is on the workflow detail: *Validate Build Stability*
  - Definition of the workflow detail
    - Brief overview of activities and artifacts typical of the work
  - Why build verification testing is so important
  - The scope of build verification tests
  - Automation

14

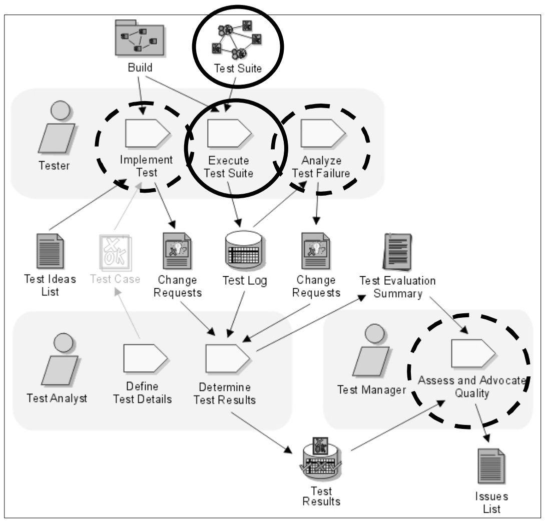
Rational
the software development company

## Validate Build Stability – Activities and Artifacts



### Validate Build Stability – Activities and Artifacts

- This section focuses on *Build Verification Tests* (BVT's)
- We will discuss some key issues associated with BVT's including deciding on appropriate *Test Suites.*
- A BVT is specialized Test and Evaluate work, so you will notice many common elements.



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

Rational the software development company

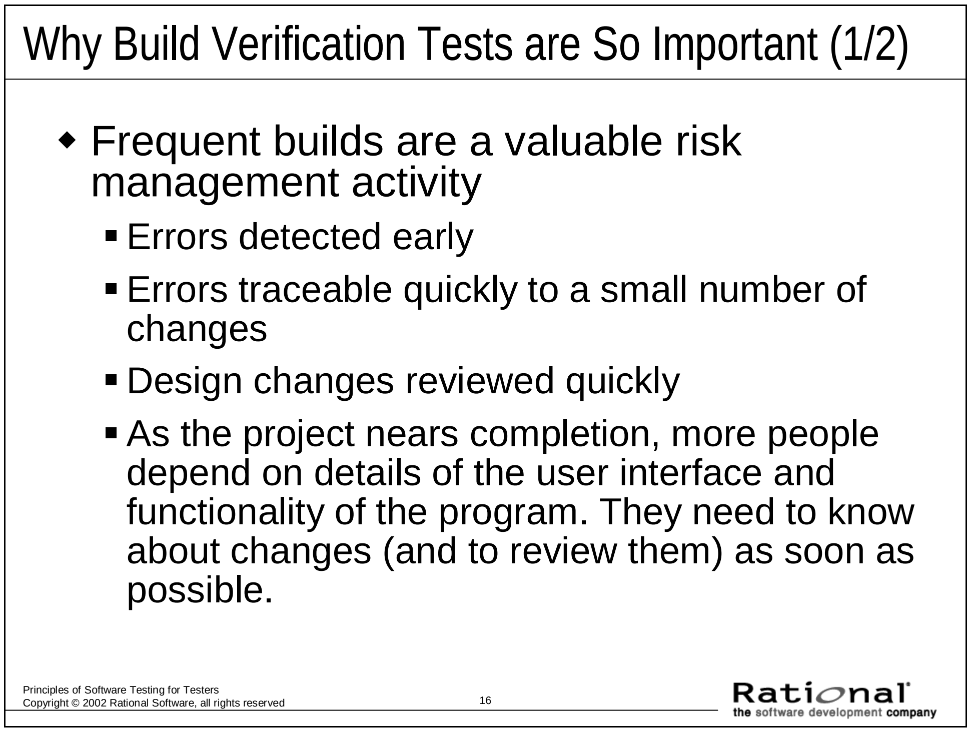
Here are the roles, activities and artifacts RUP focuses on in this work.

In the previous section, we discussed verifying your approach.

In this module, we'll talk about assessing whether a software build is stable enough to conduct detailed testing work against it.

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

## Why Build Verification Tests are So Important (1/2)



### Why Build Verification Tests are So Important (1/2)

- Frequent builds are a valuable risk management activity
  - Errors detected early
  - Errors traceable quickly to a small number of changes
  - Design changes reviewed quickly
  - As the project nears completion, more people depend on details of the user interface and functionality of the program. They need to know about changes (and to review them) as soon as possible.

Principles of Software Testing for Testers

16

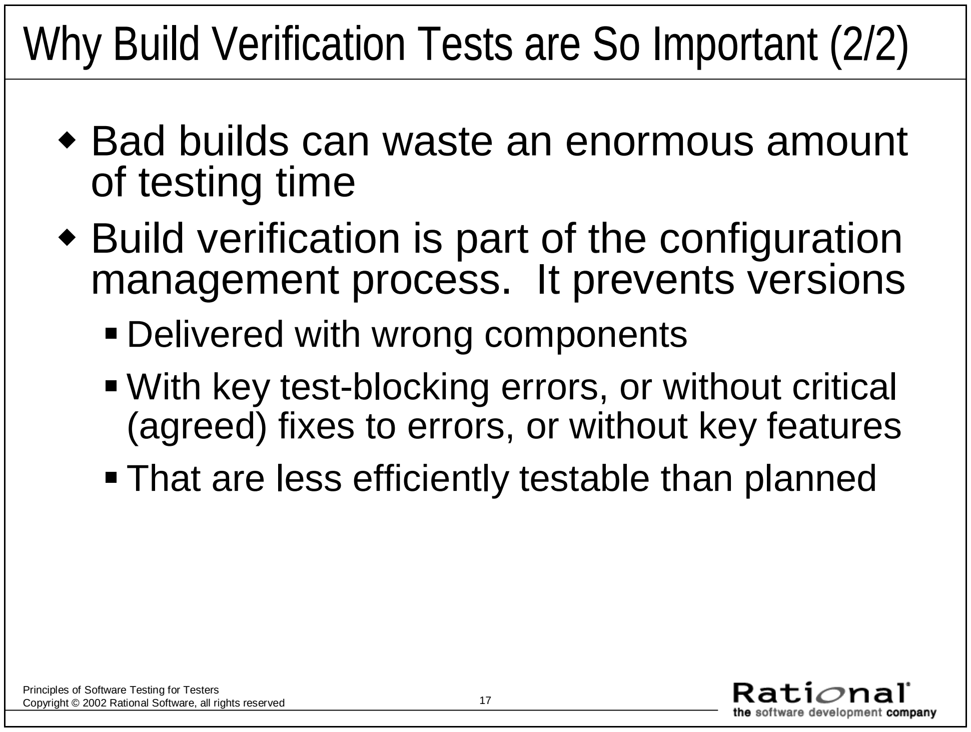
Rational®
the software development company

Costs of inefficient build management:

- New version comes to testing
- Everyone loads the new version, run their initial tests (such as regression tests of allegedly fixed bugs, and tests of other key features of interest to that tester.)
- It has critical problems – result is that the intended tests are invalid (will be ignored), all that time spent was wasted
- Everyone unloads it, gossips about the sloppiness of the programming team, eventually go back to previous version

Rapid building, in companies that don't have efficient BVTs, burns so much testing time that it is called "churning" and programming groups are discouraged from building more than once per week.

## Why Build Verification Tests are So Important (2/2)



Why Build Verification Tests are So Important (2/2)

- Bad builds can waste an enormous amount of testing time
- Build verification is part of the configuration management process. It prevents versions
  - Delivered with wrong components
  - With key test-blocking errors, or without critical (agreed) fixes to errors, or without key features
  - That are less efficiently testable than planned

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

Rational® the software development company

**Who runs the Build Verification Test?**

Consider having the Release Engineering Team run the BVT as part of the build process. (It is still responsibility of the Test Team to create and maintain the BVT.)

**How many tests should be in the Build Verification Test Suite?**

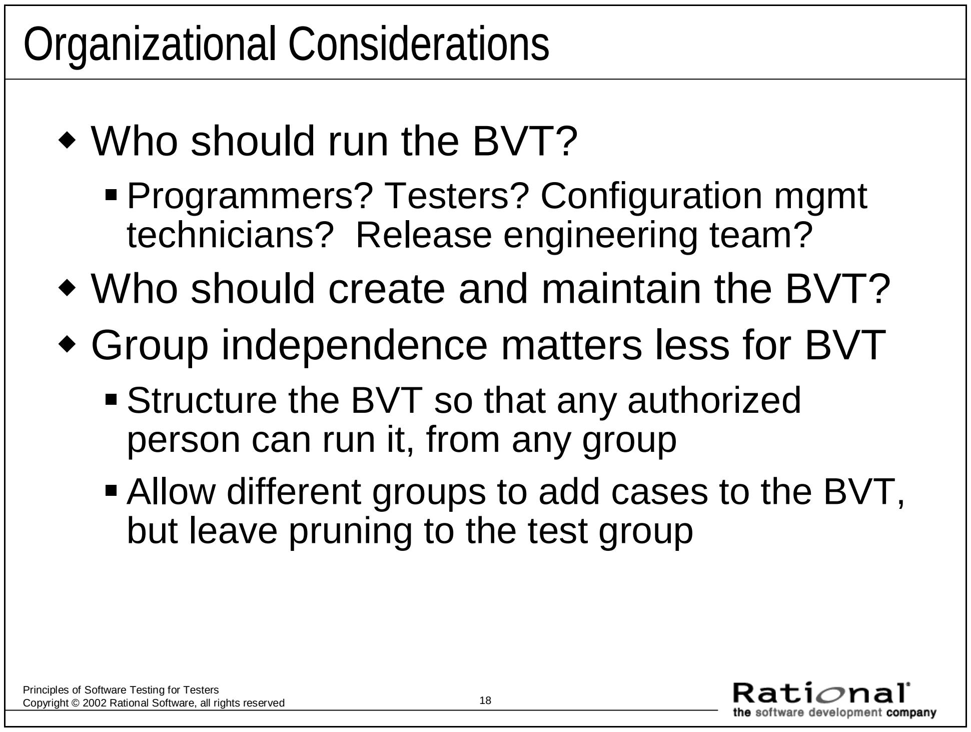
The primary goal is to confirm that this build is worth further testing. The primary testing will be done AFTER the program passes the BVT. Therefore, the BVT is a broad test but not a thorough one.

- Relaxed standards for including a test in the BVT is a risk if there are too many tests in the BVT. In this case, failing the BVT is noticed, but doesn't force a stop in using the build (resulting in immediate fix to the build-breaking bug). These "BVT" suites are more like traditional regression test suites. They are useful, but if there are many builds, your group would probably want to qualify an individual build before proceeding to the regressions.

Some groups include in the build test only those tests that would cause them to kick the build out of testing.

- We keep to minimum set if tests are manual, or if we have fragile (easy-to-break, high maintenance cost) automation.
- Some groups add tests to the BVT in an area when there is instability and change in that area. If cleanup of that area is the essential agreed benefit of a series of builds, added BTV tests can make sense. But these groups often prune their BVT suite regularly, so that tests that were tactically important a month ago are culled.

## Organizational Considerations



### Organizational Considerations

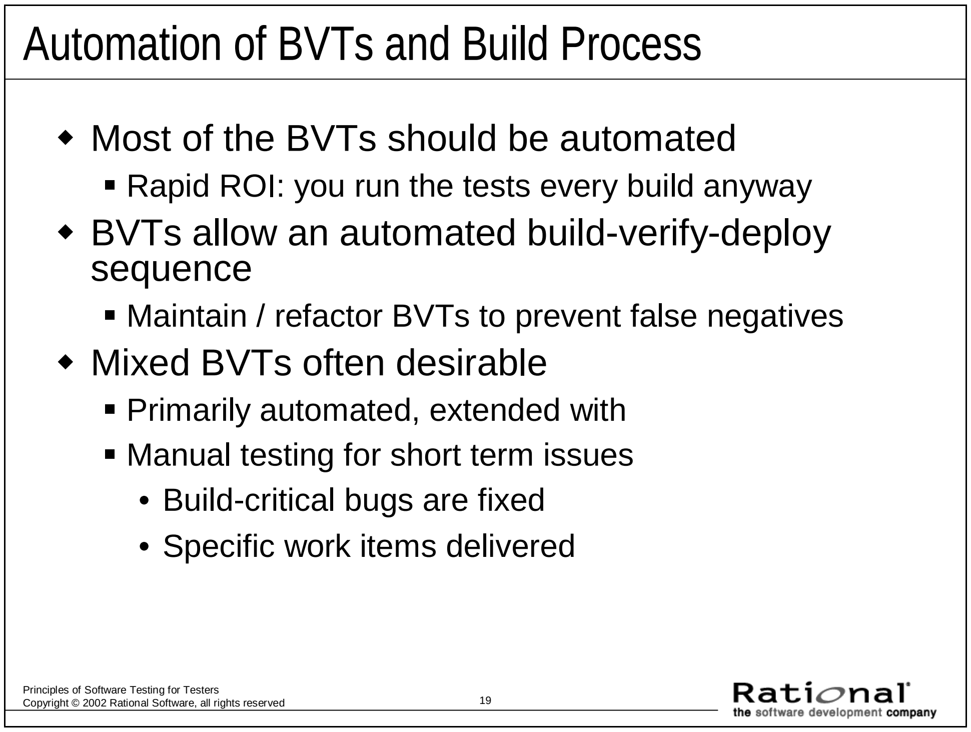
- Who should run the BVT?
  - Programmers? Testers? Configuration mgmt technicians? Release engineering team?
- Who should create and maintain the BVT?
- Group independence matters less for BVT
  - Structure the BVT so that any authorized person can run it, from any group
  - Allow different groups to add cases to the BVT, but leave pruning to the test group

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

An important concern with this approach is:

- We hinted at the group independence and redundancy problem earlier, when we talked about private vs. public bugs. The functional tester looks for different problems, in different ways from the programmer.
- Some test groups create large automated test suites that they pass to the programming team, *who then use these as primary tests for the code they are writing and maintaining*. **This can be a serious mistake**.
- White box methods are more efficient, in the hands of the programmer, than black box methods. If the programmer relies *primarily* on these functional tests, we lose the benefit of the white box testing that would otherwise be done.
- The extent to which this is a problem varies across companies. In some companies, even the build verification test has become a prop—to decide whether a build is ready for testing, these programmers decide to rely exclusively on the BVT instead of using it to supplement their own work. Their work is essential, because they understand the risks associated with the changes they have just made. The BVT is standard, and is blind to build-to-build risks. It is not effective at exposing most problems. Problems that should have been found in programmer-test, that make it past the BVT, will take much more time and work to find in functional testing.

## Automation of BVTs and Build Process



### Automation of BVTs and Build Process

- Most of the BVTs should be automated
  - Rapid ROI: you run the tests every build anyway
- BVTs allow an automated build-verify-deploy sequence
  - Maintain / refactor BVTs to prevent false negatives
- Mixed BVTs often desirable
  - Primarily automated, extended with
  - Manual testing for short term issues
    - Build-critical bugs are fixed
    - Specific work items delivered

Principles of Software Testing for Testers

19

If the program goes through many builds, most or all of the BVT tests should be automated.

It doesn't take much time to realize a return on this investment, because you would run the tests every build even if you were testing manually.

To the extent that the BVT is automated, it can be included in an automated build-verify-deploy sequence.

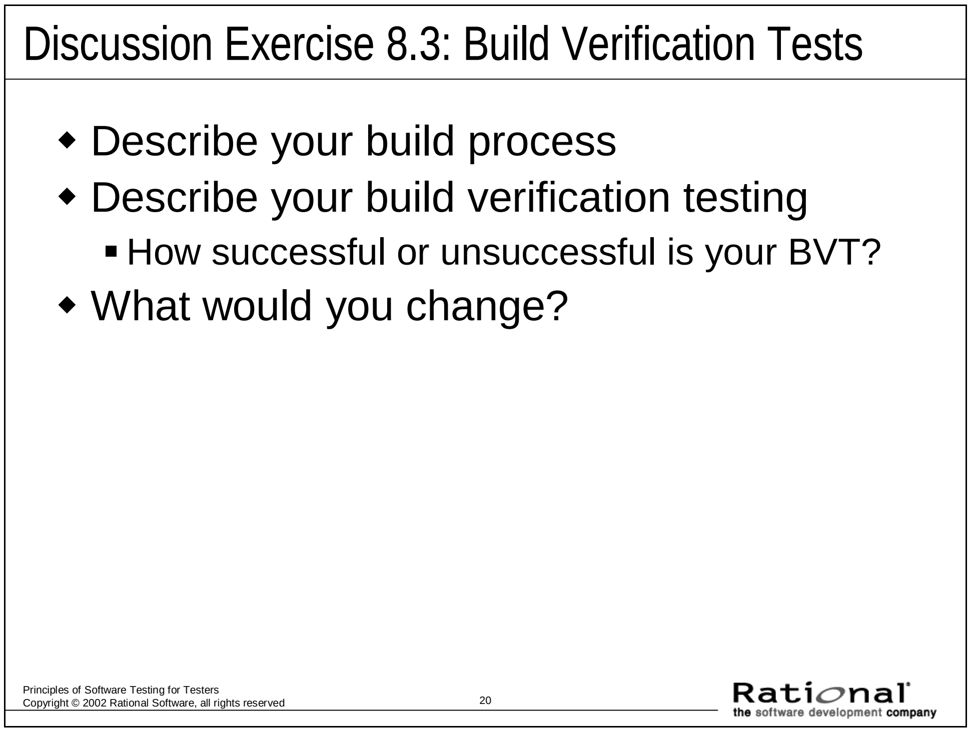
Companies that build frequently (e.g. nightly builds) need to automate the full build process (including the BVT as part of it) as much as possible.

Companies with a purely manual BVT probably cannot afford to run nightly builds.

Some groups use primarily automated BVTs but allow manual testing for issues that are short term (e.g. check that build-critical bugs were fixed—these specific tests will come and go, and may or may not be efficient to automate.)

Be sure to review the BVT at each iteration exit and consider pruning unnecessary tests from it.

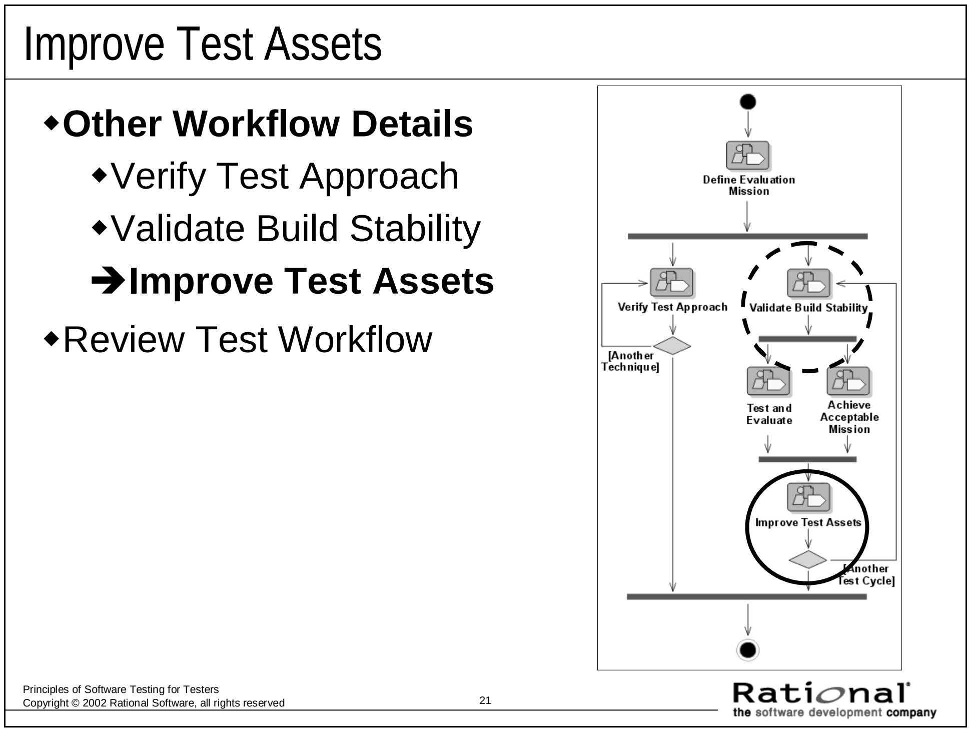
## Discussion Exercise 8.3: Build Verification Tests



### Discussion Exercise 8.3: Build Verification Tests

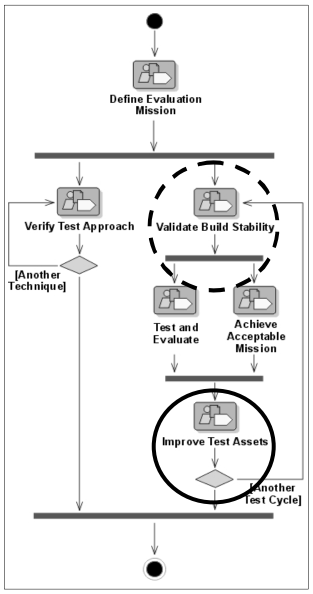
- Describe your build process
- Describe your build verification testing
  - How successful or unsuccessful is your BVT?
- What would you change?

## Improve Test Assets



### Improve Test Assets

- Other Workflow Details
  - Verify Test Approach
  - Validate Build Stability
  - ➔**Improve Test Assets**
- Review Test Workflow



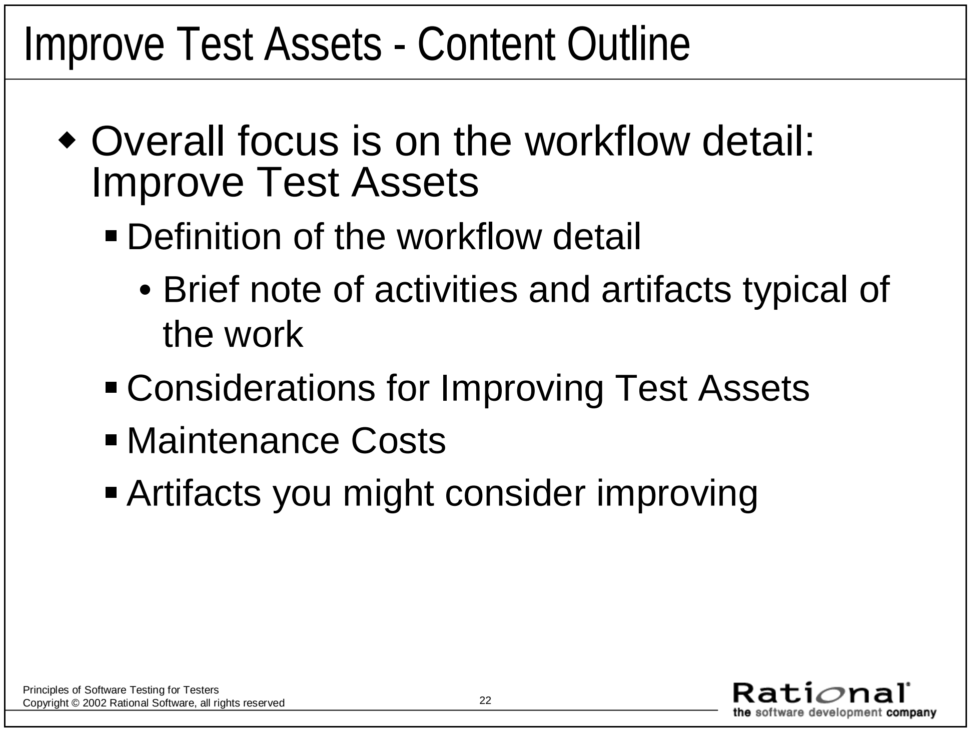
Rational the software development company

The purpose of this workflow detail is to maintain and improve the test assets. This is important especially if the intention is to reuse the assets developed in the current test cycle in subsequent test cycles.

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

- Assembling Test Scripts into additional appropriate Test Suites
- Removing test assets that no longer serve a useful purpose or have become uneconomic to maintain
- Maintaining Test Environment Configurations and Test Data sets
- Exploring opportunities for reuse and productivity improvements
- Conducting general maintenance of and making improvements to the maintainability of test automation assets
- Documenting lessons learned—both good and bad practices discovered during the test cycle

## Improve Test Assets - Content Outline



### Improve Test Assets - Content Outline

- Overall focus is on the workflow detail: Improve Test Assets
  - Definition of the workflow detail
    - Brief note of activities and artifacts typical of the work
  - Considerations for Improving Test Assets
  - Maintenance Costs
  - Artifacts you might consider improving

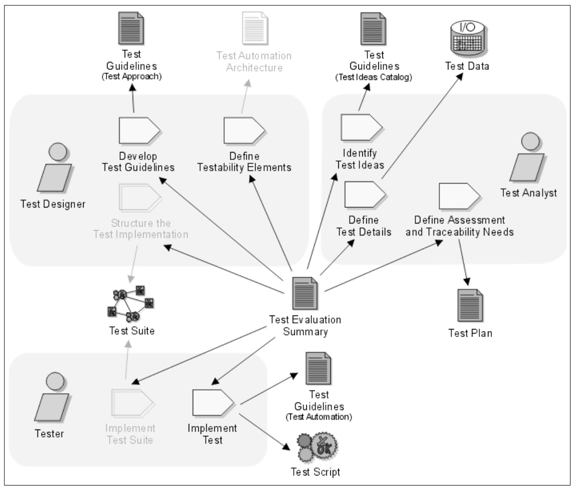
Principles of Software Testing for Testers

22
Rational the software development company

## Improve Test Assets – Activities and Artifacts

### Improve Test Assets – Activities and Artifacts

- This section focuses on making sure regular improvements are made to the test effort.
- We will discuss some of the key concerns in making ongoing improvements to test assets and activities.


Test Guidelines (Test Approach)
Test Automation Architecture
Test Guidelines (Test Ideas Catalog)
I/O
Test Data
Develop Test Guidelines
Define Testability Elements
Identify Test Ideas
Test Analyst
Test Designer
Structure the Test Implementation
Define Test Details
Define Assessment and Traceability Needs
Test Suite
Test Evaluation Summary
Test Plan
Tester
Implement Test Suite
Implement Test
Test Guidelines (Test Automation)
Test Script

Rational the software development company

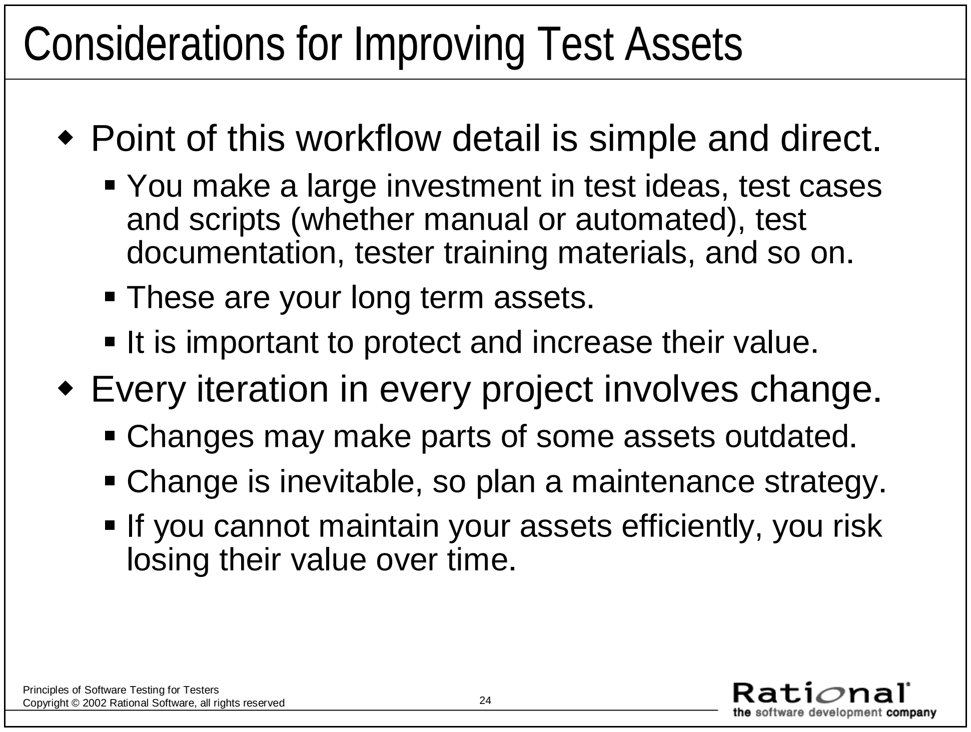
Here are the roles, activities and artifacts RUP focuses on in this work.

In the previous module, we discussed Build Verification Testing.

In this module, we'll talk about ongoing improvement of test assets over the course of the iteration, and the life of the project.

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

## Considerations for Improving Test Assets



### Considerations for Improving Test Assets

- Point of this workflow detail is simple and direct.
  - You make a large investment in test ideas, test cases and scripts (whether manual or automated), test documentation, tester training materials, and so on.
  - These are your long term assets.
  - It is important to protect and increase their value.
- Every iteration in every project involves change.
  - Changes may make parts of some assets outdated.
  - Change is inevitable, so plan a maintenance strategy.
  - If you cannot maintain your assets efficiently, you risk losing their value over time.

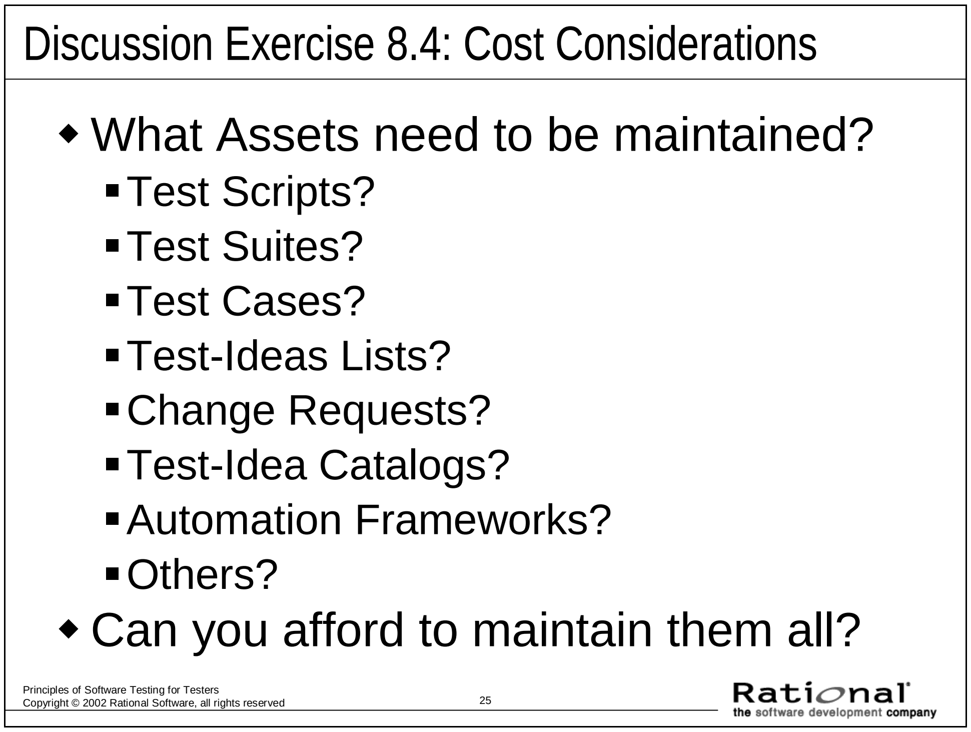
Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 24

Rational® the software development company

The point is to maintain and improve the test assets.

This is important if you will reuse test assets, especially where considerable time or expense has been expended developing the assets in the first place.

## Discussion Exercise 8.4: Cost Considerations



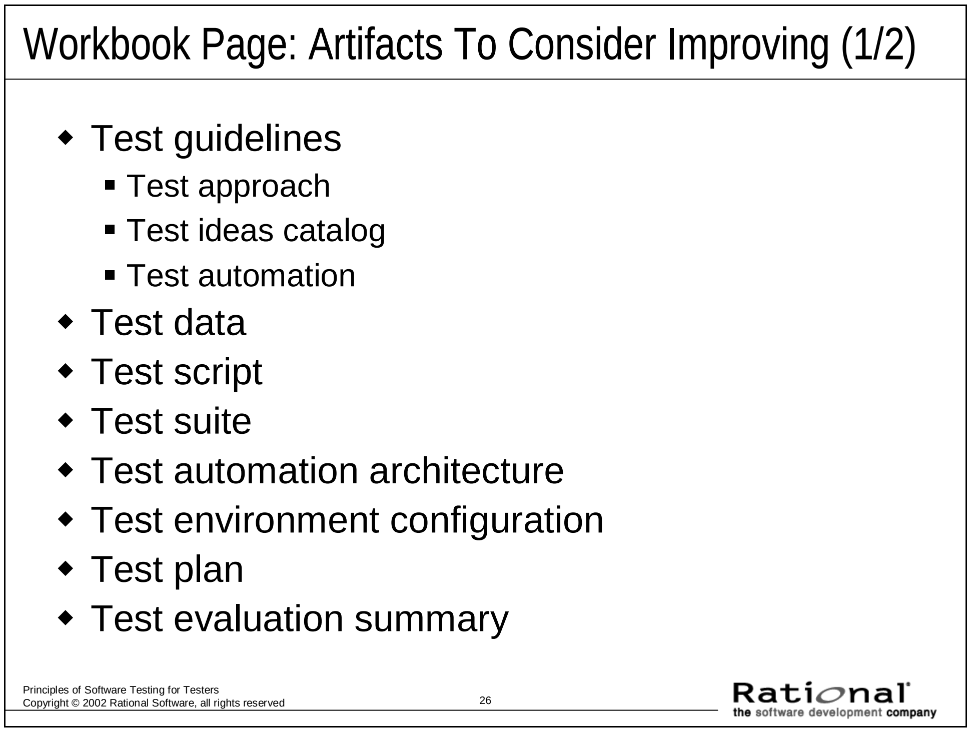
### Discussion Exercise 8.4: Cost Considerations

- What Assets need to be maintained?
  - Test Scripts?
  - Test Suites?
  - Test Cases?
  - Test-Ideas Lists?
  - Change Requests?
  - Test-Idea Catalogs?
  - Automation Frameworks?
  - Others?
- Can you afford to maintain them all?

You won't have time or resources to maintain and improve all of your test assets to the level you would like. You will need to make trade-offs and decide which assets are the most beneficial and economical to maintain.

Over time, the effort spent on maintenance activity will indicate which assets are of more or less value to maintain. Take the time to understand this important feedback so you can make the best use of your limited resources in each subsequent test cycle or iteration.

## Workbook Page: Artifacts To Consider Improving (1/2)



### Workbook Page: Artifacts To Consider Improving (1/2)

- Test guidelines
  - Test approach
  - Test ideas catalog
  - Test automation
- Test data
- Test script
- Test suite
- Test automation architecture
- Test environment configuration
- Test plan
- Test evaluation summary

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

Rational the software development company

Test guidelines

- A documented record of any of the following: process control and enactment decisions, standards to be adhered to, or good-practice guidance generally to be followed by the practitioners on a given project.

Test data

- The definition (usually formal) of a collection of test input values that are consumed during the execution of a test, and expected results referenced for comparative purposes during the execution of a test.

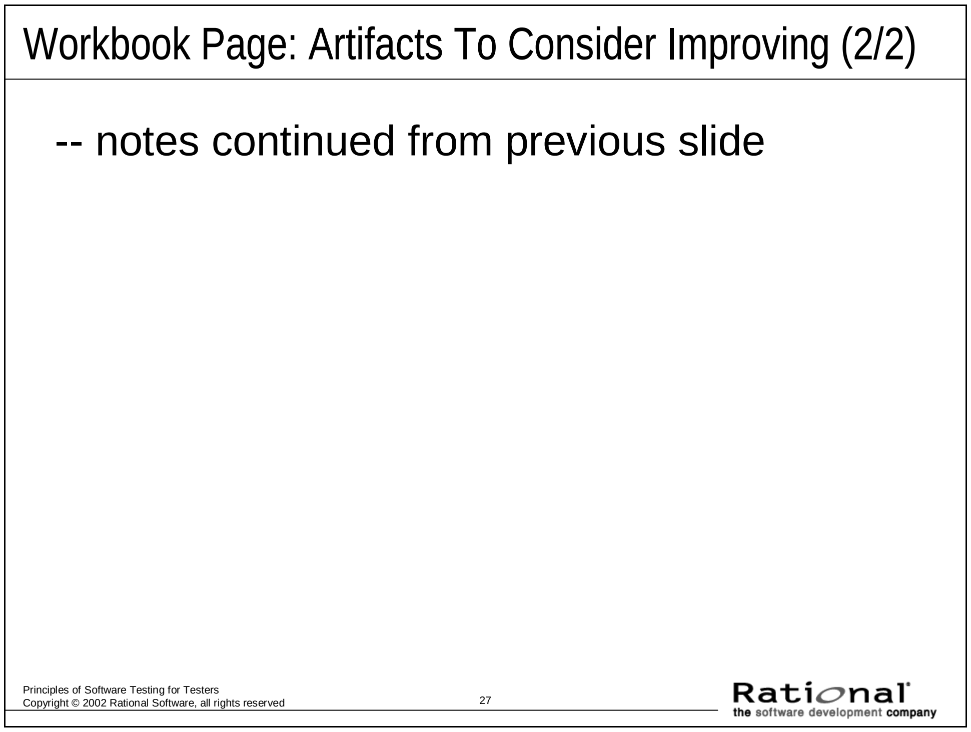
Test script

- The step-by-step instructions that realize a test, enabling its execution. Test Scripts may take the form of either documented textual instructions that are executed manually or computer readable instructions that enable automated test execution.

Test suite

- A package-like artifact used to group collections of Test Scripts, both to sequence the execution of the tests and to provide a useful and related set of Test Log information from which Test Results can be determined.

## Workbook Page: Artifacts To Consider Improving (2/2)


Workbook Page: Artifacts To Consider Improving (2/2)

-- notes continued from previous slide

Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 27

Rational the software development company

Test automation architecture

- A composition of various test automation elements and their specifications that embody the fundamental characteristics of the test automation software system. The Test Automation Architecture provides a comprehensive architectural overview of the test automation system, using a number of different architectural views to depict different aspects of the system.

Test environment configuration

- A specific arrangement of hardware, software, and the associated environment settings required to conduct accurate tests that enable the evaluation of the Target Test Items.

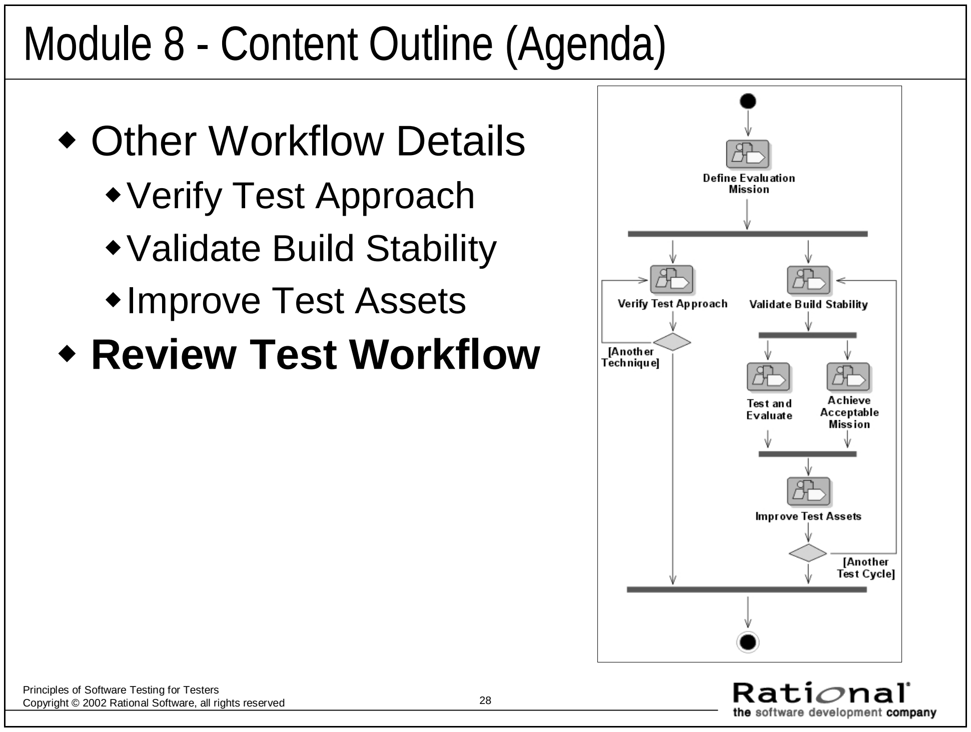
Test plan

- The definition of the goals and objectives of testing within the scope of the iteration (or project), the items being targeted, the approach to be taken, the resources required and the deliverables to be produced. Test evaluation summary

Test Evaluation Summary

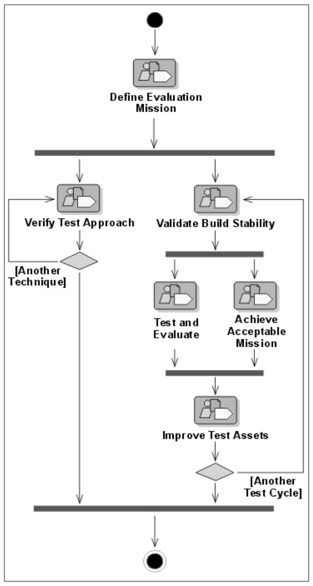
- Organizes and presents a summary analysis of the Test Results and key measures of test for review and assessment, typically by key quality stakeholders. In addition, the Test Evaluation Summary may contain a general statement of relative quality and provide recommendations for future test effort.

## Module 8 - Content Outline (Agenda)



### Module 8 - Content Outline (Agenda)

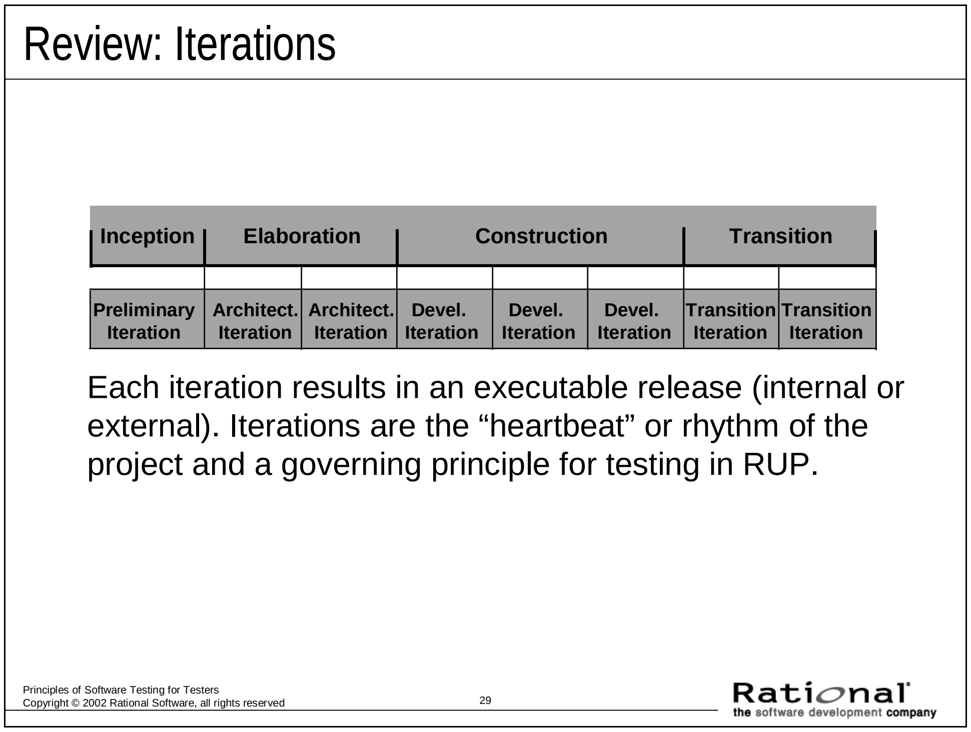
- Other Workflow Details
  - Verify Test Approach
  - Validate Build Stability
  - Improve Test Assets
- **Review Test Workflow**



Principles of Software Testing for Testers
Copyright © 2002 Rational Software, all rights reserved 28

Rational the software development company

## Review: Iterations



### Review: Iterations

| Inception | Elaboration | | Construction | | | Transition | |
|---|---|---|---|---|---|---|---|
| Preliminary Iteration | Architect. Iteration | Architect. Iteration | Devel. Iteration | Devel. Iteration | Devel. Iteration | Transition Iteration | Transition Iteration |

Each iteration results in an executable release (internal or external). Iterations are the “heartbeat” or rhythm of the project and a governing principle for testing in RUP.

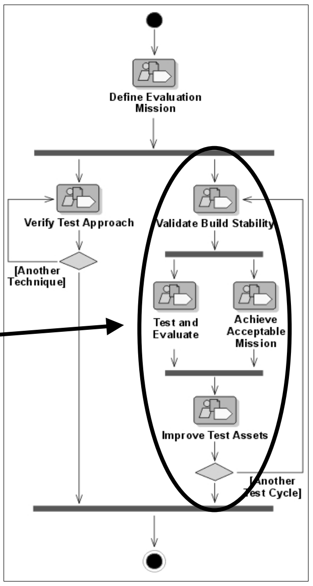
29

In Module 3, we covered iterations. There are often multiple iterations per phase of the lifecycle.

## The Test Workflow Occurs in Each Iteration

### The Test Workflow Occurs in Each Iteration

- This diagram represents testing work within an Iteration.
- All workflow details can be performed in a single Iteration.
- There are usually several Test Cycles within the Iteration.



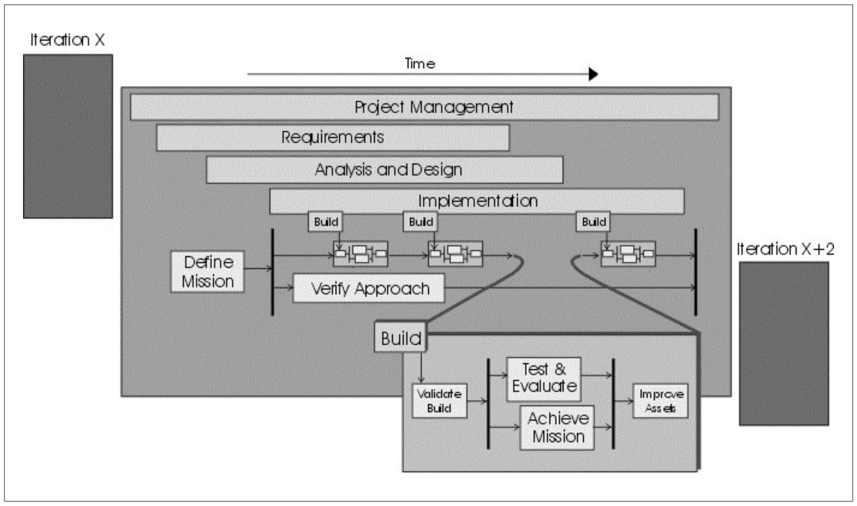
Every iteration will potentially involve all of the workflow details we have discussed during this course. Although you may decide in a given iteration that specific workflow details will not be performed, it is important to have considered each one and arrived at a reasoned conclusion why that work doesn't need to be undertaken.

Within each Iteration, there will be one or more Test Cycles. Each test cycle represents testing work undertaken against a specific software build.

There may also be multiple techniques to verify as being appropriate and useful within the project context. As the project progresses, you may need to verify changes to the existing techniques or introduce new ones. To help focus this work, it can be useful to scope it based on a single technique, or a limited subset of them.

## Each Build Is a Candidate for a Cycle of Testing

### Each Build Is a Candidate for a Cycle of Testing



As discussed at the start of this course, assessment activities – which include testing – form a part of the software lifecycle in each iteration.

Within an iteration, each software build is a potential candidate to be tested. Within each iteration – and for each build to be tested within the iteration – the challenge is to:

- Focus on the most important factors that will motivate the test effort
- Target your approach accordingly
- Conduct appropriate tests to explore each motivating factor
- Provide ongoing objective assessment of your findings in a timely manner

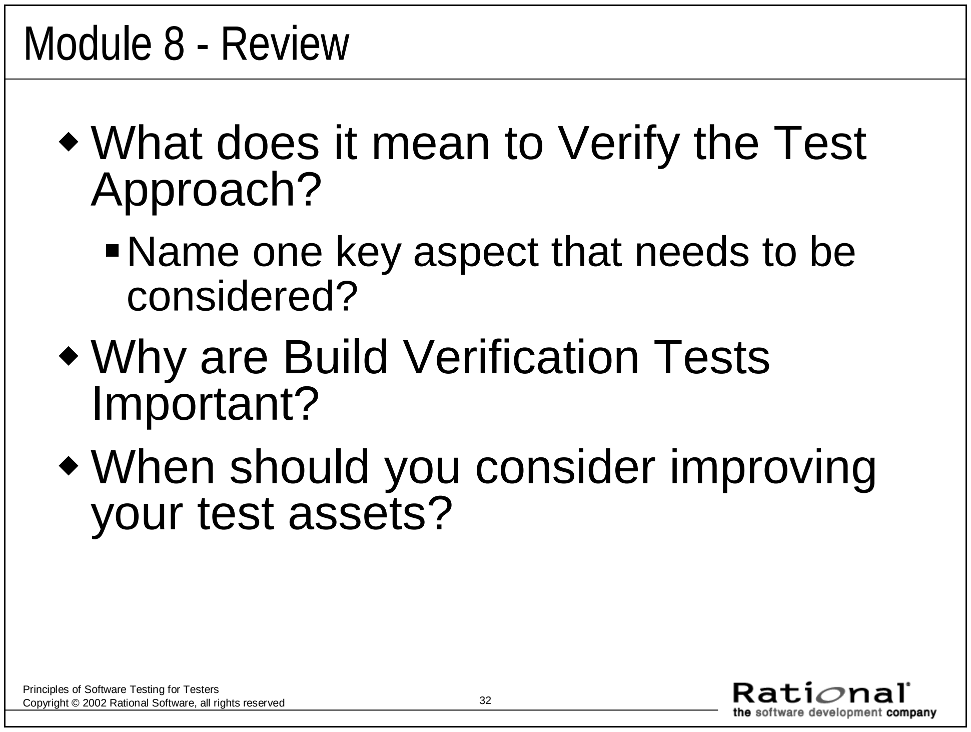
These are all aspects of what is often referred to as "Context-Driven Testing".

Note that there are various considerations why a build may or may not be tested:

- The build cycle is too frequent (e.g. daily), requiring too much overhead to complete a cycle of testing for each build.
- The build is too unstable and/ or incomplete. Again, build verification testing helps to address this risk.

In cases where a specific software build won't be tested, it typical for testing to continue on an earlier build.

## Review



### Module 8 - Review

- What does it mean to Verify the Test Approach?
  - Name one key aspect that needs to be considered?
- Why are Build Verification Tests Important?
- When should you consider improving your test assets?