

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:




Principles of Software Testing for Testers

Module 1: Software Engineering Practices

(Some things Testers should know about them)

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

 The important thing for you to teach your students in this module is how each of these practices relate to software testing.

Your goal should be to help your students understand enough about each practice that they can work cohesively as productive members of a software project, especially one that follows RUP.

Highlight how each practice will have an effect on the way in which testing is undertaken in terms of planning (i.e. iterations), the role that testing will play (i.e. objective assessment of iteration objectives) and in terms of the information that is available to base testing on (e.g. risks, use cases, architecture).

Note that for many slides the title offers a good cue to talk to. You can also use the animation callouts to cue your delivery.

Objectives

- ◆ Identify some common software development problems.
- ◆ Identify six software engineering practices for addressing common software development problems.
- ◆ Discuss how a software engineering process provides supporting context for software engineering practices.

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

2

Rational
the software development company

In this module, we explore a number of software engineering practices and explain why these are considered to be good practices to follow. We will also look at how a software engineering process helps you to implement these and many other engineering practices.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

⌚ Transition Slide. Don't spend any time here.

Module 1 - Content Outline (Agenda)

- ➔ Software development problems
 - ♦ Six software engineering practices
 - ♦ Supporting software engineering practices with process

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


3

Rational
the software development company

In this section, we describe some common software development problems and their root causes.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

 It can be effective to present this slide as a group discussion, asking the students to give examples of where they have seen these symptoms.

This acts as an “ice breaker”, encouraging early participation and gives the students the opportunity to share some of their experiences.

Alternatively, you might pick one or two to talk about from your own experience. Avoid reciting the entire list - it doesn't add a lot of value.

Be careful to limit any discussion to an appropriate amount of time.

Symptoms of Software Development Problems

- × **User or business needs not met**
- × **Requirements churn**
- × **Modules don't integrate**
- × **Hard to maintain**
- × **Late discovery of flaws**
- × **Poor quality or poor user experience**
- × **Poor performance under load**
- × **No coordinated team effort**
- × **Build-and-release issues**

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

4

Rational
the software development company

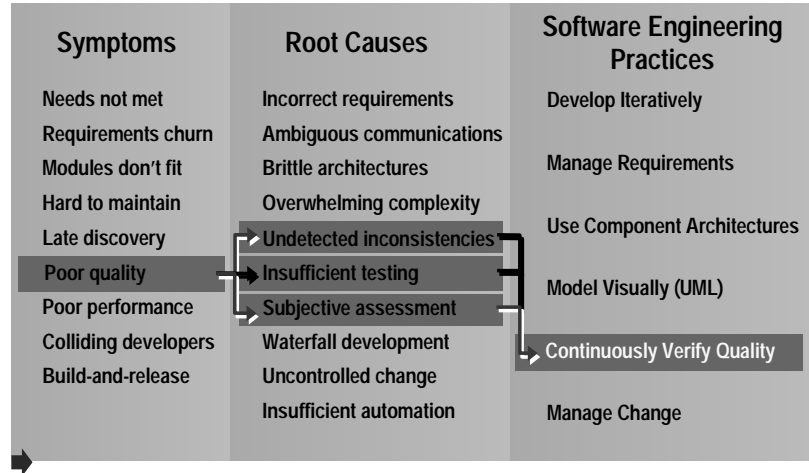
Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

🔄 Animation note:
Automatic – timed over ~ 5 seconds

- 1.symptom highlighted
- 2.root causes highlighted
- 3.engineering practice highlighted

Trace Symptoms to Root Causes



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

5

Rational
the software development company

Treat these root causes, and you'll eliminate the symptoms. Eliminate the symptoms, and you'll be in a much better position to develop quality software in a repeatable and predictable fashion.

The software engineering practices listed here are approaches to developing software that have been commercially-proven. When used in combination, they strike at many of the common root causes of software development problems. These are also referred to as "best practices," not so much because we can precisely quantify their value, but rather because they are observed to be the common practices adopted by successful organizations.

These software engineering practices have been identified by observing thousands of customers on thousands of projects and they align with similar observations made by independent industry experts*.

*(CHAOS Report ©1999, The Standish Group International).

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

⌚ Transition Slide. Don't spend any time here.

Module 1 - Content Outline (Agenda)

- ◆ Software development problems
- ➔ Six software engineering practices
- ◆ Supporting software engineering practices with process

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

6

Rational
the software development company

In this section, we describe some commonly recommended software engineering practices.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

⌚ Introduction Slide. Don't go into too much detail here, or you will have little to say on the subsequent slides.

💡 You will need to be sensitive to testers who have no control over this aspect of their software development project: don't leave them with a sense that they will fail without it. You should try to convey the concept & benefits of iterative development at a very high level in this module. Avoid getting bogged down in detailed discussion.

In the subsequent slides, focus on how this practice effects software testing. Some main points you should highlight for software testing are:

- that each iteration should result in an executable release. This allows the actual execution of the tests to start much earlier than in a waterfall lifecycle. This will have an effect on the way testing is planned and resourced and the amount of up-front documentation that is done.
- that Iterations are usually planned to address a set of key risks, most often technical ones. Testing needs to help assess whether those risks have been adequately addressed.

Practice 1: Develop Iteratively

Software Engineering Practices

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

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

7

Rational
the software development company

Developing iteratively is a technique that is used to deliver the functionality of a system in a successive series of releases of increasing completeness. Each release is developed in a specific, fixed time period called an "iteration."

Each iteration is focused on identifying, defining and analyzing some set of requirements, and designing, building and testing software based on the understanding of those requirements.

If you want to learn more about how software can be developed iteratively, you can take the *Rational Unified Process Fundamentals* course.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

👤 Sometimes it can be effective to present this slide as a group discussion. Here's an example:

To illustrate a problem with the waterfall model:

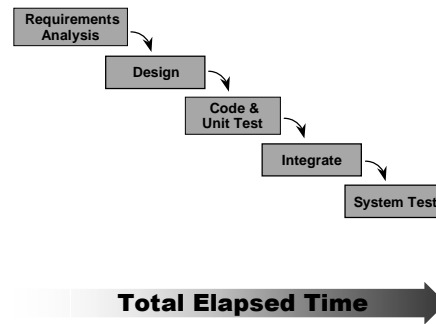
Context: Suppose you estimate that a project will take 2 years, and as you get closer to the end of that time you realize it will actually take closer to 3 years. At the end of 2 years, the project is assessed to consider whether it should be continued.

Q: At 2 years, what artifacts have been produced?

A: It is unlikely that there will be any executable, working software, certainly not in terms of end-to-end business value to the customer. Waterfall development doesn't allow for partial delivery: it's generally "all or nothing". Diagrams and models are great interim artifacts, but they can't execute and as such they don't represent anything of significant value to the customer.

Waterfall Development Characteristics

Waterfall Process



- ♦ Delays confirmation of critical risk resolution
- ♦ Measures progress by assessing work-products that are poor predictors of time-to-completion
- ♦ Delays and aggregates integration and testing
- ♦ Precludes early deployment
- ♦ Frequently results in major unplanned project extensions

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

8

Rational
the software development company

Waterfall is conceptually straightforward because it produces a single deliverable. The fundamental problem of this approach is that it pushes risk forward in time, where it's costly to undo mistakes from earlier phases. An initial design will likely be flawed with respect to its key requirements, and furthermore, the late discovery of design defects tends to result in costly overruns and/or project cancellation. The waterfall approach tends to mask the real risks to a project until it is too late to do anything meaningful about them.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

🔄 Animation note:
Automatic – The callout appears .5 second after the main slide appears.

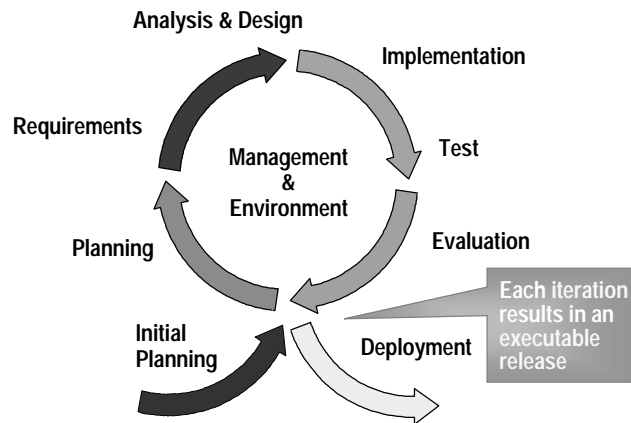
Because each iteration produces an executable, test teams need to think about test planning on an iteration basis. This is in addition to any longer-term lifecycle plans (e.g. “Master” Test Plans) they may want to define.

Having access to an executable release in each iteration also has an effect on how and when the test team is resourced, and on the way tests are defined, implemented, and executed.

Note that while you will typically address each of the waterfall “phases” within an iteration, it won’t usually be in sequence as a “mini waterfall”. In an iterative development cycle, each iteration ideally follows a dynamic structure that reflects a similar arrangement to the macro-structure of the RUP phases: an Iteration has a period of inception, elaboration, construction and transition, where elaboration & construction are based around one or more software builds.

💡 Avoid going into detail about RUP at this stage.

Iterative Development Produces an Executable



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

9

Rational
the software development company

The earliest iterations address greatest risks. Each iteration produces an executable release. Each iteration includes integration and test. Iterations help to:

- resolve major risks before making large investments
- enable early objective feedback
- make testing and integration continuous
- focus the project on achievable short-term objective milestones
- make it possible to deploy partial implementations of the completed final system

Iterative processes were developed to address the problems with the waterfall discussed on the previous slide. With an iterative process, the phase concerns of the waterfall process are addressed in each iteration (although not typically in sequence, usually somewhat more in parallel). Instead of developing the whole system in lock step, an increment (i.e. a subset of system functionality) is selected and developed, then another increment, etc.

The selection of each increment to be developed is based on its potential to address key risks, the highest priority risks being addressed first. To address the selected risk(s), a subset of use cases or use-case instances are selected. The *minimal* set of use-case instances are realized (developed) that will allow objective verification (i.e., through a set of executable tests) of the risks that you have chosen to address. The next increment addresses the next highest risks, and so on.

Principles of Software Testing for Testers Instructor Notes

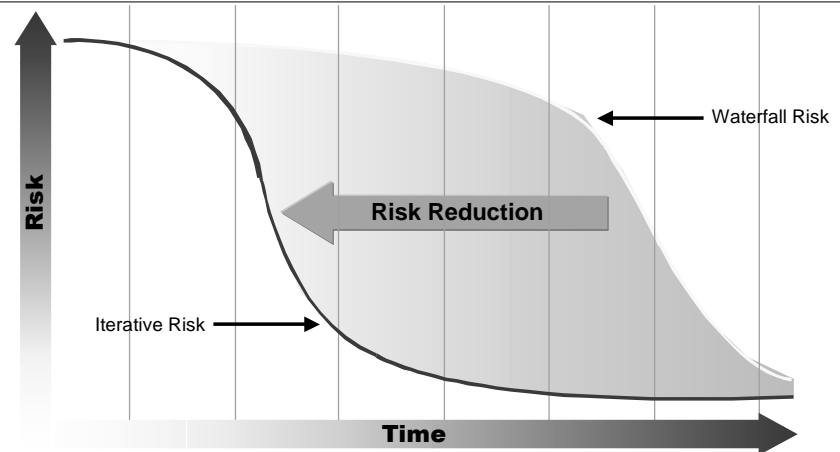
Instructor Notes:

Because addressing risk is a key target for each iteration, test teams need to think about how testing can best help the project assess the likelihood and impact of those risks. This includes assessment of the mitigation strategies and direct solutions that are employed by the project to address those risks.

The test team should strive to provide the project team with objective and timely assessments of risk based on tests conducted against executable software builds.

In assessing the known risks, the test team also needs to act as the “radar” for the project in helping to identify and uncover new potential risks.

Risk Profiles



Iterative development drives risks out early.

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

10

Rational
the software development company

Iterative development produces the architecture first, allowing integration to occur “as the verification activity” of the design phase, and allowing design flaws to be detected and resolved earlier in the lifecycle. Continuous integration throughout the project replaces the big bang integration at the end of a project.

Iterative development also provides much better insight into quality, because system characteristics that are largely inherent in the architecture (e.g., performance, fault tolerance, maintainability) are tangible earlier in the process. Thus, issues are still correctable without jeopardizing target costs and schedules.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

⌚ Introduction Slide. Don't go into too much detail here, or you will have little to say on the subsequent slides.

🗣️ You might facilitate a short discussion with the students concerning how requirements will be elicited on their project and how the project will manage them. Reference this discussion in subsequent modules when you cover quality and test techniques.

In the subsequent slides, focus on how this practice effects software testing. Some main points you should highlight for software testing are:

- that requirements **will** change. This means you will need to strike a careful balance with how much planning and documentation you will do upfront.
- that traceability is a useful technique to employ, but that it needs to be managed at an appropriate level of detail.
- that uses cases are a powerful technique that can be applied to requirements management, and that a good set of use cases are useful input to software testing.

Practice 2: Manage Requirements

Software Engineering Practices

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

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

11

Rational
the software development company

A report from the Standish Group confirms that a distinct minority of software development projects is completed on-time and on-budget. In their report, the success rate was only 16.2%, while challenged projects (operational, but late and over-budget) accounted for 52.7%. Impaired projects (canceled) accounted for 31.1%. These failures are attributed to poor requirements management, incorrect definition of requirements from the start of the project, and poor requirements management throughout the development lifecycle. (Source: Chaos Report, <http://www.standishgroup.com>).

If you want to learn more about how to manage requirements, you can take the *Requirements Management with Use Cases* course.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

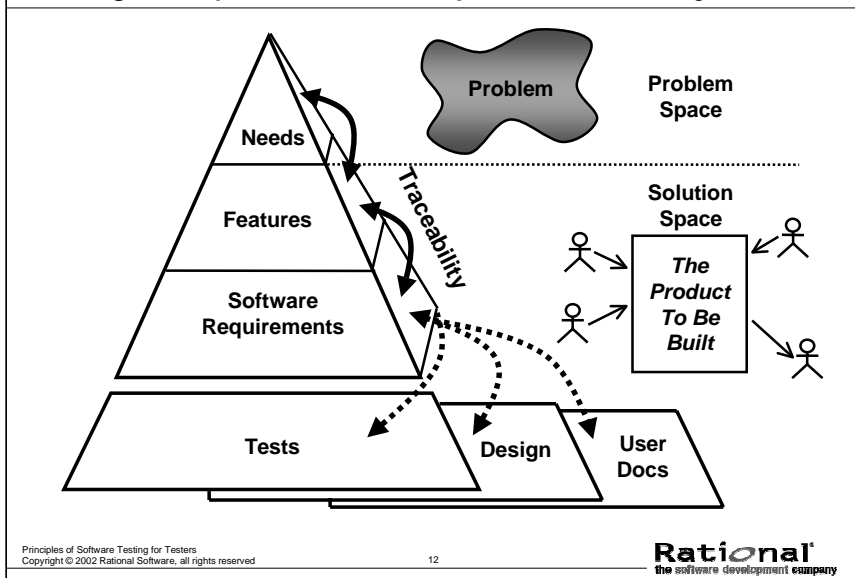
⌚ Avoid going too deep here. Remember that requirements-based testing gets covered in detail in subsequent modules, in addition to traceability strategies.

Requirements **will** change, especially in an iterative lifecycle where the requirements are uncovered over successive iterations and evolve as the resulting software is actually experienced by the stakeholders.

You will need to strike a careful balance with how much test planning and documentation you will do outside of the scope of an iteration.

Traceability is a useful technique to help manage any potential impact to your tests from subsequent changes in requirements. However, as a general rule it will usually be more appropriate to manage traceability at some level of parent element rather than at the level of each "unique" child element.

Manage Requirements - Map of the Territory



Managing requirements involves the translation of stakeholder requests into a set of key stakeholder needs and system features. These in turn are detailed into specifications for functional and non-functional requirements. Detailed specifications are translated into a design, user documentation and tests.

The requirements for the software are a key input to testing. You will often find important problems at the boundary between each section of the pyramid – for example, are the needs appropriately reflected in the features? Does the design appropriately reflect the requirements? Later in this course we will discuss testing based on requirements.

To help manage the relationship between the requirements and the tests derived from those requirements, you can establish traceability relationships between those elements. Traceability assists us to do many things, including:

- Assess the project impact of a change in a requirement
- Assess the impact of a failure of a test on requirements (i.e., if test fails, the requirement may not be satisfied)
- Manage the scope of the project
- Verify that all requirements of the system are fulfilled by the implementation
- Verify that the application does only what it was intended to do
- Manage change

Later in this course we will discuss traceability and assessment needs for testing.

Principles of Software Testing for Testers Instructor Notes

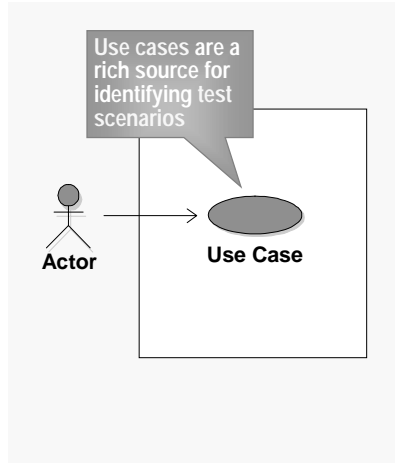
Instructor Notes:

🔄 Animation note:
Automatic – The callout appears .5 second after the main slide appears.

Uses cases are a powerful technique that can be applied to requirements elicitation and management. A good set of use cases are useful input to software testing.

To fully understand the system's purpose you must know who the system is for, that is, who will use it. Different user types are represented as actors.

Manage Requirements - Use-Case Concepts



An **actor** represents a person or another system that interacts with the system.

A **use case** defines a sequence of actions a system performs that yields a result of observable value to an actor.

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

13

Rational
the software development company

Use Cases represent a technique for defining requirements in a way that focuses on the end-user goal. They have been popularized by iterative development processes such as the Rational Unified Process. However, the technique is not specific to iterative development – it can be applied just as well to eliciting and managing requirements in a waterfall development lifecycle.

An Actor:

- is not part of the system. It represents a role that users of the system will play when interacting with it.
- can actively interchange information with the system.
- can be a passive recipient of information.
- can be a giver of information.
- can represent a human, a machine or another system.

A Use Case:

- specifies a dialogue between an actor and the system.
- is initiated by an actor to invoke certain functionality in the system.
- is a collection of meaningful, related flows of events.
- yields a result of observable value.

Taken together, all use cases provide a high-level, external view of all possible ways of using the system.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

⌚ Introduction Slide. Don't go into too much detail here, or you will have little to say on the subsequent slides.

👤 You might facilitate a short discussion with the students about architecture – either common software architecture issues (e.g. Thin vs. Fat Client, n-tier), or use the analogy of building architectures (e.g. contrast a fast food establishment vs. a church, a dog house vs. a sky scraper).

In the subsequent slides, focus on how this practice effects software testing. Some main points you should highlight for software testing are:

- that testers will likely be asked to conduct tests that assess the architecture on various dimensions of quality: reliability, performance, portability etc.
- finding architectural weakness in an application early provides immense value to the project team, therefore it is worth taking time to assess the software architecture early.

Practice 3: Use Component Architectures

Software Engineering Practices

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

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

14


Rational
the software development company

Software architecture is the development product that gives the highest return on investment with respect to quality, schedule, and cost, according to the authors of *Software Architecture in Practice* (Len Bass, Paul Clements & Rick Kazman [1998] Addison-Wesley). The Software Engineering Institute (SEI) has an effort underway called the Architecture Tradeoff Analysis (ATA) Initiative to focus on software architecture, a discipline much misunderstood in the software industry. The SEI has been evaluating software architectures for some time and would like to see architecture evaluation in wider use. By performing architecture evaluations, AT&T reports a 10% productivity increase (from news@sei, Vol. 1, No. 2).

If you want to learn more about the use of component architectures in software development, you can take the *Object Oriented Design with UML* or *Principles of Architecting Software* courses.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

 You might encourage discussion at this point. A single requirement, such as throughput or fault tolerance, affects almost every design decision made on a system. For example, if building a transportation system (such as for trains), it is likely that they will have a “no single point of failure” requirement that must be in every developer’s mind every step of the way. Many architectural mechanisms will be developed to accommodate that single requirement.

If you can get some students to describe their architectural challenges, this point is more effectively driven home.

Resilient Component-Based Architectures

- ◆ **Resilient**
 - Meets current and future requirements
 - Improves extensibility
 - Enables reuse
 - Encapsulates system dependencies
- ◆ **Component-based**
 - Reuse or customize components
 - Select from commercially available components
 - Evolve existing software incrementally

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

15

Rational
the software development company

Architecture is an aspect of design. It is about making decisions on how the system will be built. But it is not all of the design. It stops at the major abstractions, or in other words, the elements that have some pervasive and long-lasting effect on the system’s performance and ability to evolve.

A software system’s architecture is perhaps the most important aspect that can be used to control the iterative and incremental development of a system throughout its lifecycle.

The most important property of an architecture is resilience -- flexibility in the face of change. To achieve it, architects must anticipate evolution in both the problem domain and implementation technologies to produce a design that can gracefully accommodate such changes. Key techniques are abstraction, encapsulation, and object-oriented analysis and design. The result is that applications are fundamentally more maintainable and extensible.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

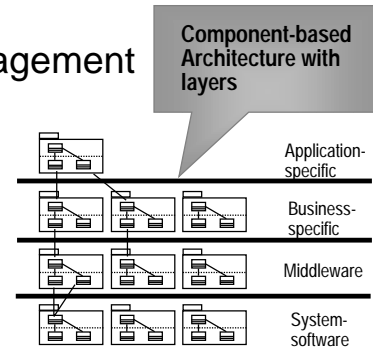
Because students vary in how familiar they are with the concept of architecture applied to software, it is best to get a sense of this from the students before beginning this section. If they are fairly unfamiliar, it helps to use the analogy of buildings or civil engineering. The more complex the building, the more critical a good architecture is. The longer you want the building to be useful, the more effort and expense you will put into the architecture. And in both of these cases, the choice of architect is critical.

Regarding the last paragraph in the student notes:

- One challenge is that third-party component developers may not have provided adequate testability features, making testing – esp. automation – difficult to implement.
- One problem is that if you find defects in third-party components, they may be difficult to get resolved.
- One opportunity is that either the third-party developer or other testers in the community may already have developed test assets you can make use of.

Purpose of a Component-Based Architecture

- ◆ Basis for reuse
 - Component reuse
 - Architecture reuse
- ◆ Basis for project management
 - Planning
 - Staffing
 - Delivery
- ◆ Intellectual control
 - Manage complexity
 - Maintain integrity



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

16

Rational
the software development company

Definition of a (Software) Component:

Process Definition: A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces.

UML Definition: A physical, replaceable part of a system that packages implementation, and conforms to and provides the realization of a set of interfaces. A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files.

Your testing will typically be constrained by and dependent on the delivery and availability of software components. As noted on the slide, planning and staffing of the project will often be based around components, so your test plans will most likely need to reflect this as well.

Components also help to manage complexity by hiding (or encapsulating) unnecessary detail, making it easier to discuss how basic and fundamental interaction occurs between components at different levels of detail (or abstraction). This will assist in gaining an understanding of how the software is designed to work, and will help you to reason about useful tests to conduct.

In some cases components will be acquired from third-party suppliers or reused from other projects within your organization. This poses some interesting potential problems and challenges (as well as opportunities) for testing software systems built using previously developed components.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

You may wish to lead a discussion about models and why we build models in general.

We build models because the thing we are studying is so complex that no one can understand and remember all of the details. Typically a model will be looked at from different perspectives or views. Each view typically leaves out the details that are unimportant in the context of that view.

A good model facilitates our understanding of the larger issues by hiding complexity.

Practice 4: Model Visually (UML)

Software Engineering Practices

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

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

17

Rational
the software development company

A **model** is a simplification of reality that provides a complete description of a system from a particular perspective. We build models so that we can better understand the system we are modeling. We build models of complex systems because we cannot comprehend any such system in its entirety.

If you want to learn more about visual modeling in software development, you can take the *Fundamentals of Visual Modeling with UML* course.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

Visual modeling for software is analogous to blueprints for construction.

You might want to use the blueprint analogy with your students. For example:

- before building a bridge, the architect builds a model and has it reviewed.
- blueprints for an office building contain different levels of detail: foundations, structural beams, electrical wiring, heating and air-conditioning, plumbing etc. Each aspect typically has a separate view or “overlay layer” in the complete blueprint “model”.

Why Model Visually?

- ◆ To help manage complexity
 - To capture both structure and behavior
 - To show how system elements fit together
 - To hide or expose details as appropriate
- ◆ To keep design and implementation consistent
- ◆ To promote unambiguous communication
 - UML provides one language for all practitioners

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

18

Rational
the software development company

Modeling is important because it helps the development team visualize, specify, construct, and document the structure and behavior of a system’s architecture. Using a standard modeling language such as the UML (the Unified Modeling Language), different members of the development team can communicate their decisions unambiguously to one another.

Using visual modeling tools facilitates the management of these models, letting you hide or expose details as necessary. Visual modeling also helps you maintain consistency among a system’s artifacts: its requirements, designs, implementations and tests. In short, visual modeling helps improve a team’s ability to manage software complexity.

Different techniques can be used to verify aspects of a model prior to the physical implementation of program code associated with the model. The UML itself provides rules to establish whether a model is “well-formed”, and various software is commercially available to “walk the model” looking for anomalies. That software can typically be extended with user-defined rules.

Tools are also becoming available that take UML models as input and allow the generation of test assets based on those models. For more information about these tools, you might want to look at the *Rational Quality Architect* product.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

🔄 Animation note:
Automatic – The callouts appear .5 second after the main slide appears.

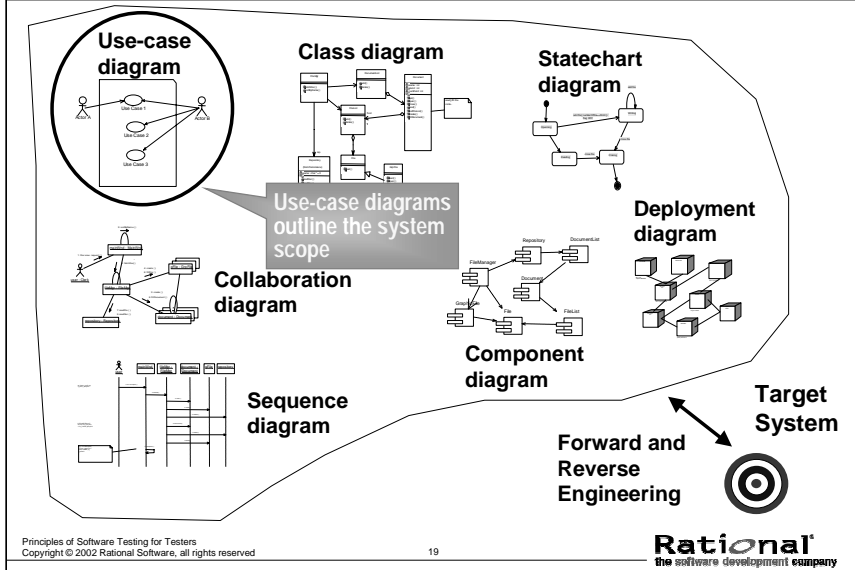
The point to be made is that the UML is the language we use to visually model. Since it is a widely-adopted standard, it facilitates the understanding and communication of the visual models we create.

Activity diagrams are not shown on the slide. Activity diagrams can be used to model workflows in business process engineering.

Discuss the diagrams that testers might typically be expected to read, review, and discuss:

- use-case diagrams
- activity and state-chart diagrams
- interaction diagrams (sequence, collaboration)
- simple class diagrams (entity-class domain models, analysis classes)
- deployment diagrams.

Visual Modeling Using UML Diagrams



Visual modeling with the UML makes an application's architecture tangible, permitting us to assess it in multiple dimensions. How portable is it? Can it exploit expected advances in parallel processing? How might we modify it to support a family of applications? We've discussed the importance of architectural resilience and quality. The UML enables us to evaluate these key characteristics during early iterations -- at a point when design defects can be corrected before threatening project success.

If your software development team will be making use of visual models, it is worth taking some time to learn how to read, interpret and discuss these models. You will find this assists your communication with the developers, and offers you new and unique insight into what the software is designed to do. In turn, this will help you reason more completely about the appropriate tests that you should conduct.

Advances in forward and reverse engineering techniques permit changes to an application's model to be automatically reflected in its source code, and changes to its source code to be automatically reflected in its model. This is critical when using an iterative process, where we expect such changes with each iteration.

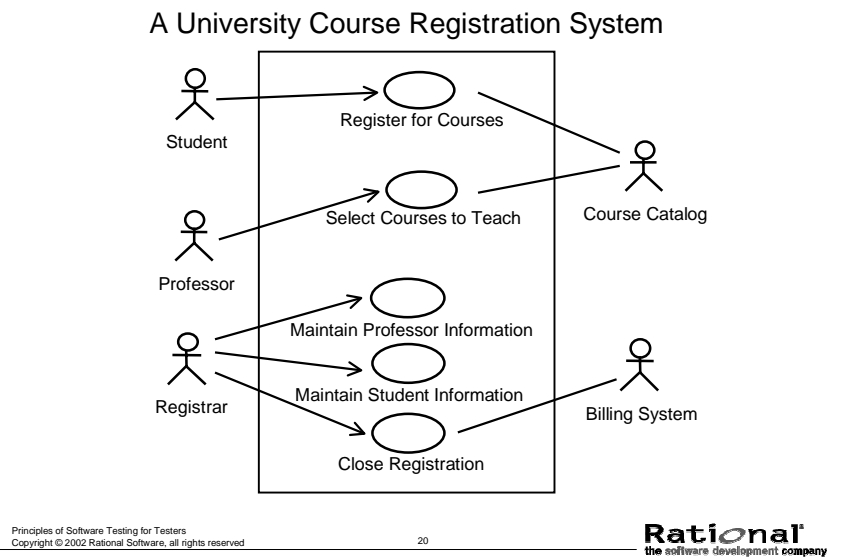
Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

The purpose of this diagram is to familiarize the student with the rich language of the UML which is available to represent software artifacts. You should NOT try to explain all the notation, just the purpose of the diagram and how to read it (superficially).

A use case has a set of properties: brief description, flow of events, special requirements, etc. Use cases are enclosed in the use-case-model artifact.

Workbook Page: A Sample UML Diagram – Use Cases



Often a global use-case diagram will be included in the Use-Case-Model Survey to give a graphical overview of the system.

This should include all use cases, actors, and their relationships that cover the scope of the system being built.

Use case diagrams are used to show the existence of use cases and their relationships, both to each other and to actors. An actor is something external to the system that has an interface with the system, such as end users. A use case models a dialogue between actors and the system. A use case is initiated by an actor to invoke a certain functionality in the system. For example, in the diagram above, one class of user of the system is student. In this system, students have a goal to use the system to register for courses. Hence, Register for Courses is a use case.

The arrow (which is optional) indicates the direction in which messages are invoked in the interaction. Here, the Student actor sends messages to the Register for Courses use case.

A use case is a complete and meaningful flow of events. The flow of events supplements the use case diagram and is usually provided in text format.

Taken together, all use cases constitute all possible ways of using the system.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

Quality and Testing are not synonymous, although testing is an important aspect of assessing quality. While software testing is most often the main technique used to assess software quality, there are other techniques that we will mention during the course.

☛ You may be drawn into arguments here about Quality Assurance vs. Testing/ Quality Assessment. Avoid this trap, if possible. If you cannot, there are additional instructor materials with the course that may help you manage the discussion to closure (see the section entitled “*common controversies*”).

Practice 5: Continuously Verify Quality

Software Engineering Practices

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

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

21

Rational
the software development company

Quality has many definitions. We'll discuss some of the more commonly held opinions about quality in a subsequent module. However, it is fair to state that achieving quality is not simply about "meeting requirements" or producing a product that meets user needs and expectations. Quality also includes identifying the measures and criteria (to demonstrate the achievement of quality), and the implementation of a process to ensure that the resulting product has achieved an appropriate degree of quality.

Software testing is an important aspect of Software Quality process. Software testing accounts for 30% to 50% of software development costs in many organizations, yet most people believe that software is not well-tested before it is delivered. This contradiction is arguably rooted in two interesting observations. First, testing software is enormously difficult. The different ways a given program can behave are almost infinite. Second, testing is typically done without a clear methodology and without adequate supporting tools. While the complexity of software makes "complete" testing an impossible goal, an appropriate methodology for the project context, and use of appropriate supporting tools, can help to improve the productivity and effectiveness of the software testing effort.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

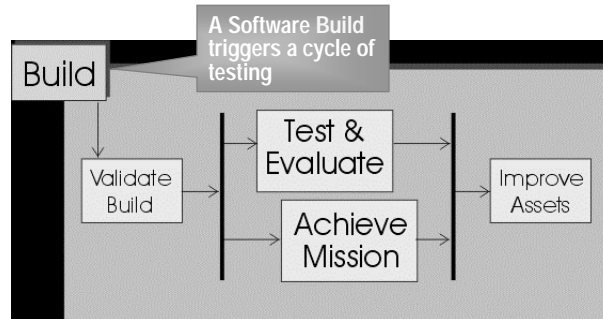
🔄 Animation note:
Automatic – The callouts appears .5 second after the main slide appears.

Point out that the independent test effort fits within the iterative product development lifecycle. Contrast this early start with the typical late start in waterfall development.

Stress the key role that test plays as part of the assessment activities. Without the objective proof that test provides by validating the executable software, there can be little confidence that the goals of the iteration have been realized.

Note that Assessment activities are broader than just “testing”. Assessment also occurs in the form of reviews and other types of evaluation that don’t execute the software product itself. Instead, these assessment activities help to verify that the software development process itself is being followed and is appropriate.

Continuously Verify Quality – in each Iteration



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

22

Rational
the software development company

In traditional software development, there is a tendency to delay certain types of assessment—such as black-box testing—until late in the development cycle. In the case of black-box testing, delaying these tests will delay the discovery of potentially important problems until late in the development cycle. In many cases this will mean the problems are too expensive to correct.

In iterative development, assessment activities are an integral part of the effort in each iteration: they are needed to provide objective proof that the goals of the iteration have been met. Without iteration-based assessment, it isn’t possible to objectively evaluate whether an iteration achieved its goals: Were the key risks addressed satisfactorily? Were the planned features delivered? Did the software exhibit the required quality attributes?

The design and development of tests can be as complex and arduous as developing the software product itself. You can mitigate the risk of expensive problems derailing the testing process by starting early. In general it is best to start testing activities in the same iteration as the first executable software release is planned.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

Animation note:

Automatic – The callouts appears .5 second after the main slide appears.

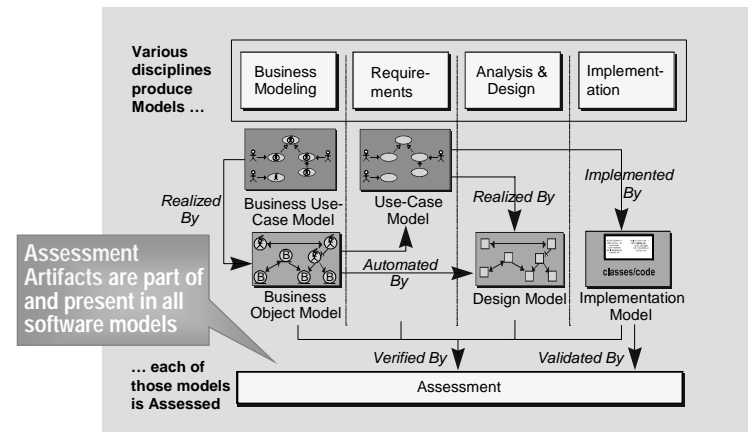
You can consider all of the models listed here, taken together, to be “the system model.”

The business model is a little different in that it describes the business as a whole, not just the automated part. The other models describe various aspects of an information system that will support the business model.

Point out that each of these models is incrementally developed across many iterations.

As such, assessment activities can make use of these models from the earliest iterations. However, there needs to be acknowledgement of the incomplete nature of these models in the earlier parts of the lifecycle, and acceptance that the models will change and evolve over time.

Continuously Verify Quality – Software Models



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

23

Rational
the software development company

The UML can be used to produce a number of models that represent various perspectives or views on a software system as it evolves. Several models are useful to fully describe the evolving system, with different software disciplines producing those models. Each model is developed incrementally over multiple iterations.

- The Business Model is a model of what the business processes are and of the business environment. It is primarily used to gain a better understanding of the software requirements in the business context.
- The Use-Case Model is a model of the value the system represents to the external users of the system environment. It describes the “external services” that the system provides.
- The Design Model is a model that describes how the software will “realize” the services described in the use cases. It serves as a conceptual model (or abstraction) of the implementation model and its source code.
- The Implementation Model represents the physical software elements and the implementation subsystems that contain them.

Assessment involves both Verification and Validation activities: verifying that the software product is being built right, and validating that the right software product is being built. This distinction refers to assessing both the appropriateness of the process by which the software product is built (verification) and the appropriateness of the resulting software product that will be delivered to the customer (validation).

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

Control of change is especially important in an iterative project. Artifacts are generally produced incrementally. At the end of each iteration, another increment is put under configuration control. Otherwise, no progress will be made, and iterations will not converge on a complete and consistent system. We are not just talking about changes to source code, but also to requirements, models, documents, plans, and all development artifacts.

Practice 6: Manage Change

Software Engineering Practices

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

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

24

Rational
the software development company

As we indicated earlier, we cannot stop change from being introduced into our project. However, we must control how and when changes are introduced into project artifacts, and who introduces the changes. We also must synchronize change across development teams and locations.

Unified Change Management (UCM) is Rational Software's approach to managing change in software system development, from requirements to release.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

🔄 Animation note:
Automatic – The callouts
appear .5 second after the
main slide appears.

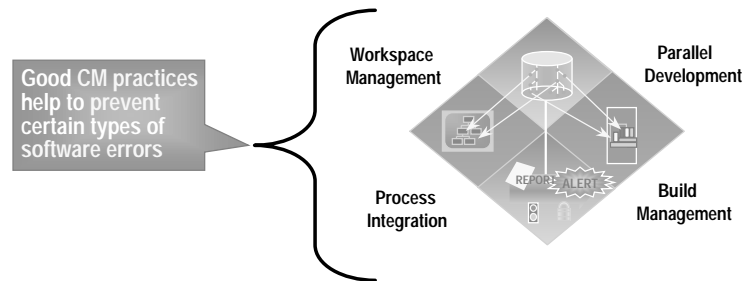
Good CM practices help to
prevent:

- resolved faults from reappearing because the fixed code is lost.
- developers overwriting each others coding changes.

If possible, use an example from your experience to illustrate what can go wrong when multiple staff work on the same set of artifacts without adequate controls. For example, two programmers attempting to make simultaneous updates to the same component, or an entire test suite failing because the wrong version of the test suite was run.

What Do You Want to Control?

- ◆ Changes to enable iterative development
 - Secure workspaces for each worker
 - Parallel development possible
- ◆ Automated integration/build management



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

25

Rational
the software development company

Establishing secure workspaces for each worker on the project provides isolation from changes made in other workspaces and control of all software artifacts -- models, code, docs, tests etc.

A key challenge to developing software-intensive systems is the need to cope with multiple workers, organized into different teams, possibly at different sites, all working together on multiple iterations, releases, products, and platforms. In the absence of disciplined control, the development process rapidly degrades into chaos. Progress can come to a stop.

Three common problems that result are:

- Simultaneous update -- When two or more workers separately modify the same artifact, the last one to make changes destroys the work of the former.
- Limited notification -- When a problem is fixed in shared artifacts, some of the workers are not notified of the change.
- Multiple versions -- It is feasible to have multiple versions of an artifact in different stages of development at the same time. For example, one software release is in use by the customer, one is actively being developed and tested, and yet another one is undergoing early prototyping of future features. If a problem is identified in any one of the versions, the fix may need to be propagated among all of them and change control can lead to chaos and halt progress.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

Discuss briefly some different aspects of change that software projects are exposed to.

Unless your students explicitly ask you to, you shouldn't discuss each one of these aspects in detail. Just pick one or two that you feel comfortable talking about.

Workbook Page: Aspects of a CM System

- ◆ Change Request Management (CRM)
- ◆ Configuration Status Reporting
- ◆ Configuration Management (CM)
- ◆ Change Tracking
- ◆ Version Selection
- ◆ Software Manufacture

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

26

Rational
the software development company

Change Request Management (CRM) addresses the organizational infrastructure required to assess the cost and schedule impacts of a requested change to the existing product. CRM addresses the workings of a Change Review Team or Change Control Board.

Configuration Status Accounting (Measurement) is used to describe the "state" of the product based on the type, number, rate and severity of defects found, and fixed, during the course of product development. Metrics derived under this aspect, either through audits or raw data, are useful in determining the overall completeness status of the project.

Configuration Management (CM) describes the product structure and identifies its constituent configuration items that are treated as single versionable entities in the configuration management process. CM deals with defining configurations, building and labeling, and collecting versioned artifacts into constituent sets and maintaining traceability between these versions.

Change Tracking describes what is done to components for what reason and at what time. It serves as history and rationale of changes. It is quite separate from assessing the impact of proposed changes as described under "Change Request Management."


Version Selection ensures that the right versions of configuration items are selected for change or implementation. Version selection relies on a solid foundation of "configuration identification."

Software Manufacture covers the need to automate the steps to compile, test and package software for distribution.

Principles of Software Testing for Testers Instructor Notes

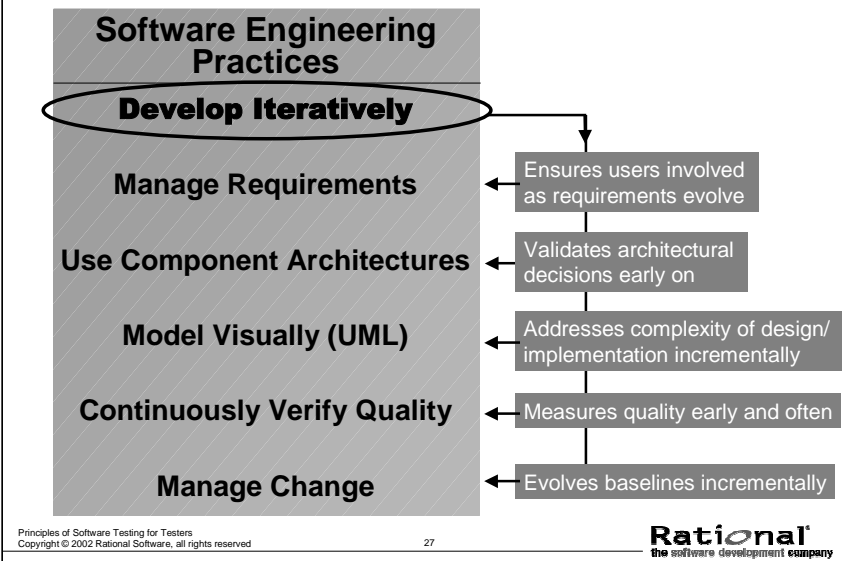
Instructor Notes:

This slide may be confusing unless it is explained properly. We have discussed each Proven Practice individually in this module. This slide is intended to illustrate how these Practices used together provide more benefit than each individually. The slide only illustrates how ONE of the Practices (Develop Iteratively) supports the others.

 As revision, you might ask your students to share their thoughts on how Quality from the Start relates to the other practices. Many comments are made in the preceding slides that will help both you as instructor and the students understand some of the interrelationships.

Be careful not to spend too much time here. If you have already taken enough time on this module, skip the discussion.

Software Engineering Practices Reinforce Each Other



In the case of our six software engineering practices, the whole is much greater than the sum of the parts. Each of the practices reinforces and, in some cases, enables the others. The slide shows just one example: how iterative development leverages the other five software engineering practices. However, each of the other five practices also enhances iterative development.

For example, iterative development done without adequate requirements management typically fails to converge on a solution: requirements change at will, users can't agree, and the iterations never reach closure. When requirements are managed, this is less likely to happen. Changes to requirements are visible, and the impact to the development process assessed before they are accepted. Convergence on a stable set of requirements is assured. Similarly, each pair of practices provides mutual support. Hence, although it is possible to use one practice without the others, additional benefits are realized by combining them.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

⌚ Transition Slide. Don't spend any time here.

The next three slides introduce the basic features of the Rational Unified Process.

Module 1 - Content Outline (Agenda)

- ◆ Software development problems
- ◆ Six software engineering practices
- ➔ Software engineering process and software engineering practices

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

28

Rational
the software development company

In this section, we briefly discuss how a defined process – such as the Rational Unified Process (RUP) – helps you to implement software engineering practices.

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

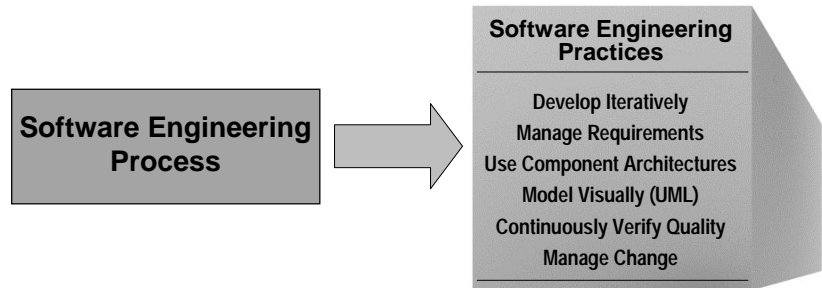
🔄 Animation note:
Automatic – The arrow
appears .5 second after the
main slide appears.

Talk generically about the
value of process guidance to
support the software
engineering practices you
wish to employ.

A documented process
helps to provide
repeatability and improve
predictability.

While you can mention the
RUP here, note that we'll be
discussing the RUP
specifically in a subsequent
module.

An Engineering Process Implements Engineering Practices



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

29

Rational
the software development company

Why have a process?

- Provides guidelines for efficient development of quality software
- Reduces risk and increases predictability
- Promotes a common vision and culture
- Harvests and institutionalizes software engineering practices

A software engineering process should provide a disciplined yet flexible approach to assigning tasks and responsibilities within a software development organization. The goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget.

The UML provides a standard for many of the artifacts of software development (semantic models, syntactic notation, and diagrams): the things that must be controlled and exchanged. But the UML is not a standard for the development *process*.

Despite all of the value that a common modeling language brings, you cannot achieve successful development of today's complex systems solely by the use of the UML. Successful iterative development also requires employing a repeatable engineering process.

Principles of Software Testing for Testers Instructor Notes

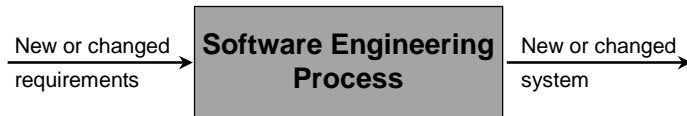
Instructor Notes:

It can be difficult to explain what a process is if people aren't already familiar with one. An informal example most people can relate to is the process of balancing a checkbook or paying bills at the end of the month. Most of us have developed a process we use -- the same steps every month. It shortens the time required to accomplish the task and ensures that we don't forget any steps.

The same applies to a software engineering process. We want it to be repeatable and to ensure that all required tasks are accomplished when required. Of course, a software engineering process is much more complex than balancing a checkbook -- the good news is that there is a tremendous amount of information contained in the RUP to help you follow a good software engineering process.

A Team-Based Definition of Process

A process defines **Who** is doing **What** **When**, and **How**, in order to reach a certain goal.



This course is about the What, When and How of Testers' activities in the process.



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

30

Rational
the software development company

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

Talk briefly here about some of the distinguishing features of a good engineering process.

Stress how important a methodology (RUP or otherwise) is to the success of software development and testing initiatives.

⌚ Avoid going into too much detail about the RUP here: you'll cover the RUP in more detail in the subsequent module that introduces the Test discipline in the RUP.

Workbook Page: Implementing Software Engineering Practices

A modern engineering process ideally:

- ◆ Supports a controlled, iterative approach
- ◆ Supports the use of user-focused requirements to coordinate and drive the work in requirements, design, implementation and test
- ◆ Enables architectural concerns to be addressed early
- ◆ Allows the process to be configured to suit the context of the individual project
- ◆ Provides guidance for conducting work (activities) and producing work products (artifacts)

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

31

Rational
the software development company

Principles of Software Testing for Testers Instructor Notes

Instructor Notes:

Module 1 - Review

- ♦ Software engineering practices guide software development by addressing root causes of problems.
- ♦ Software engineering practices reinforce each other.
- ♦ Process guides a team on who does what when and how.
- ♦ A software engineering process provides context and support for implementing software engineering practices.

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

32

Rational
the software development company