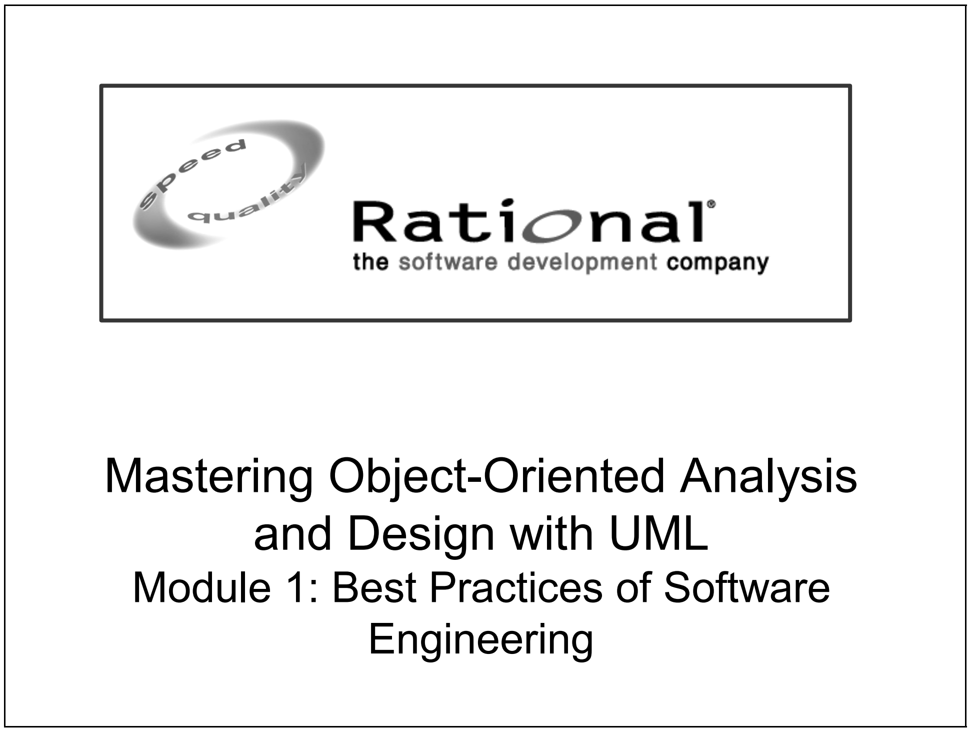
▸ ▸ ▸ Module 1
Best Practices of Software Engineering



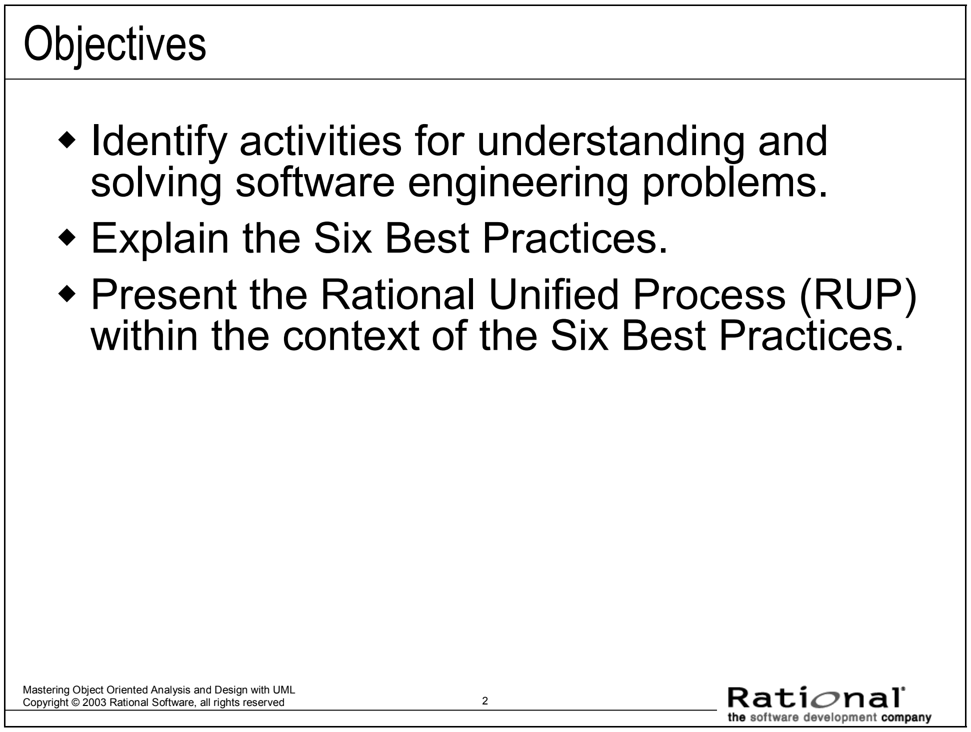
Mastering Object-Oriented Analysis and Design with UML

Module 1: Best Practices of Software Engineering

Topics

Practice 1: Develop Iteratively 1-7
Practice 2: Manage Requirements 1-11
Practice 3: Use Component Architectures 1-15
Practice 4: Model Visually (UML) 1-18
Practice 5: Continuously Verify Quality 1-22
Practice 6: Manage Change 1-27

Objectives



Objectives

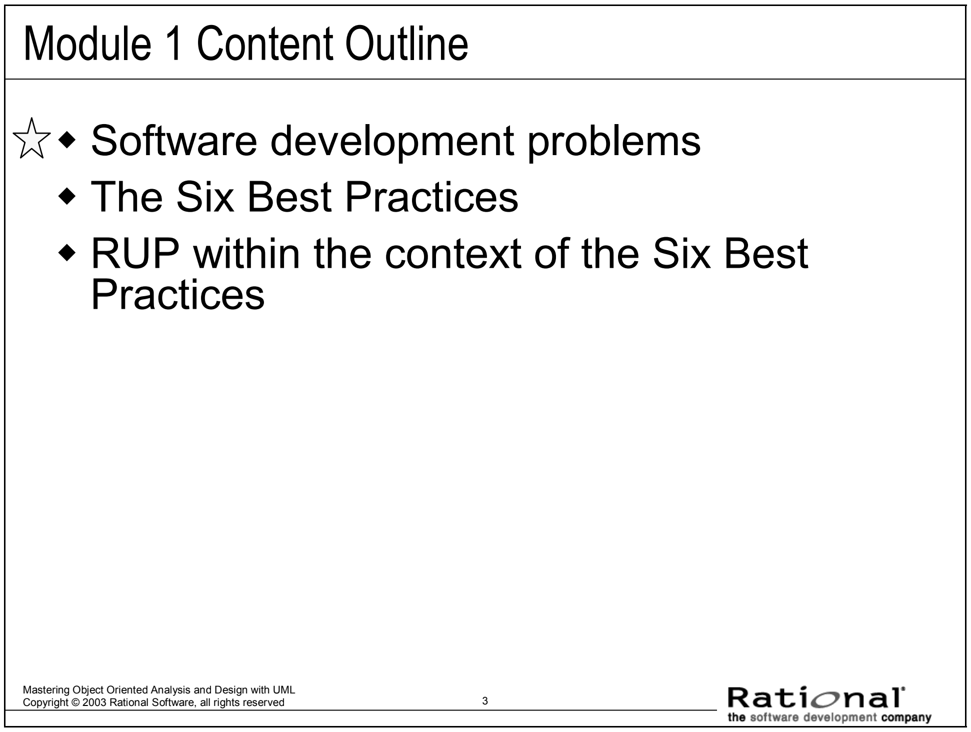
- Identify activities for understanding and solving software engineering problems.
- Explain the Six Best Practices.
- Present the Rational Unified Process (RUP) within the context of the Six Best Practices.

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 2

Rational
the software development company

In this module, you learn about recommended software development Best Practices and the reasons for these recommendations. Then you see how the Rational Unified Process (RUP) is designed to help you implement the Six Best Practices.

Module 1 Content Outline



Module 1 Content Outline

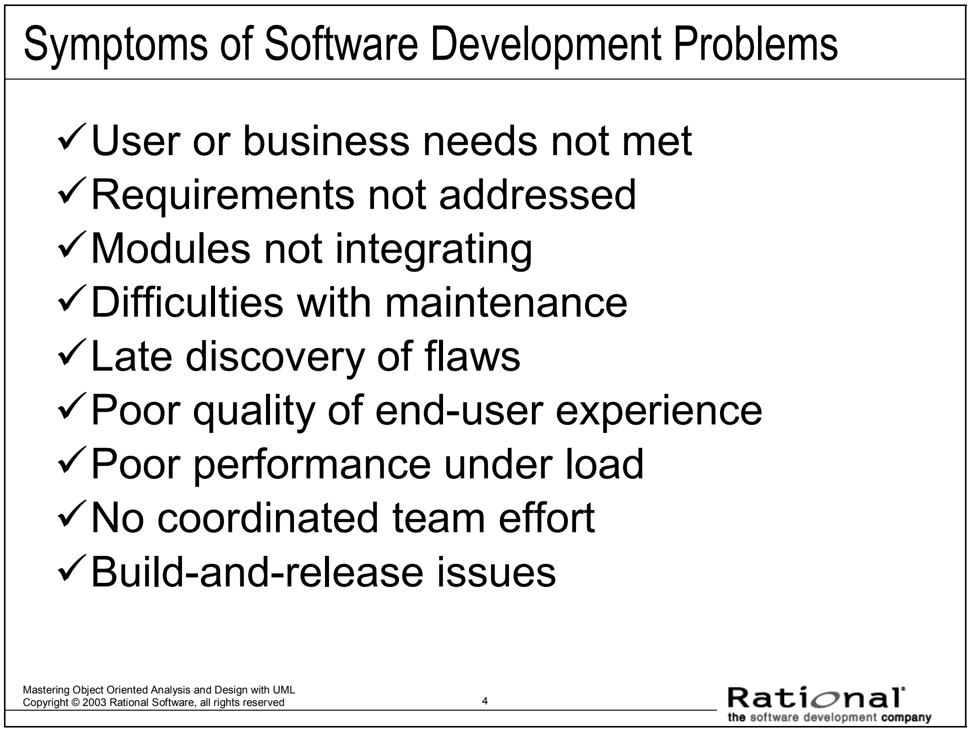
☆ ◆ Software development problems

◆ The Six Best Practices

◆ RUP within the context of the Six Best Practices

3

Symptoms of Software Development Problems

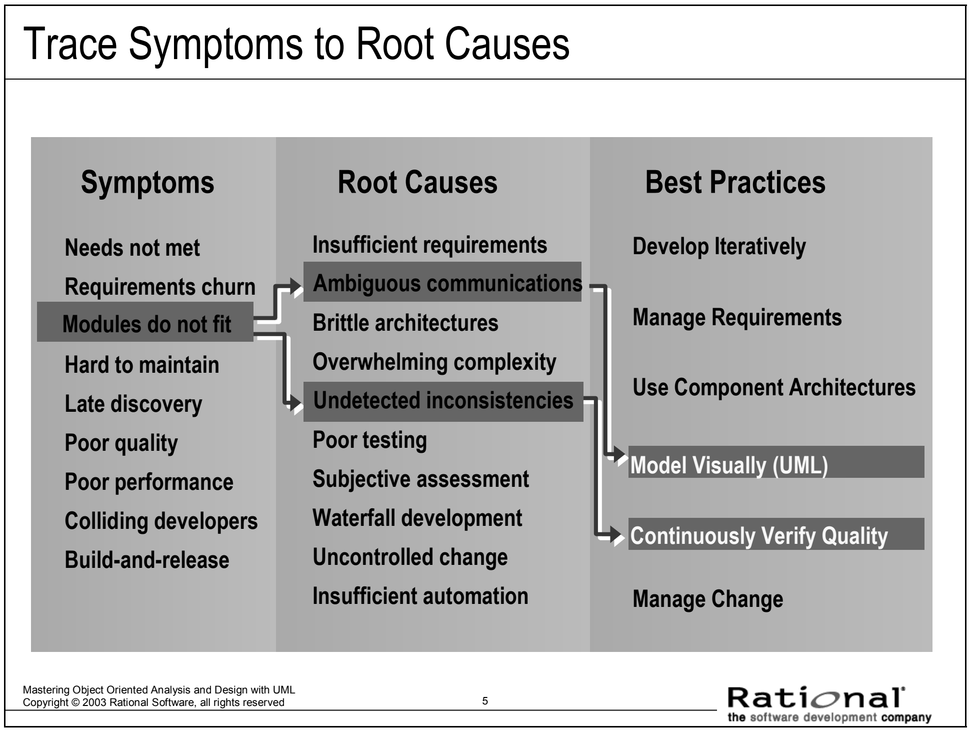


Symptoms of Software Development Problems

- ✓User or business needs not met
- ✓Requirements not addressed
- ✓Modules not integrating
- ✓Difficulties with maintenance
- ✓Late discovery of flaws
- ✓Poor quality of end-user experience
- ✓Poor performance under load
- ✓No coordinated team effort
- ✓Build-and-release issues

4

Trace Symptoms to Root Causes



Trace Symptoms to Root Causes

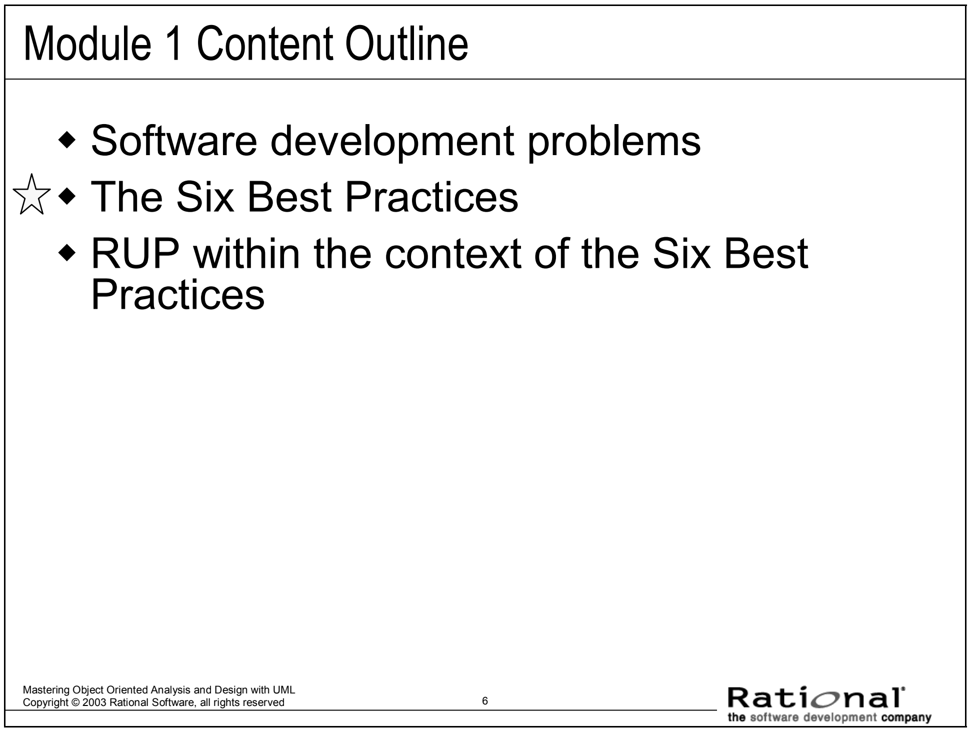
Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 5

Rational
the software development company

Treat these root causes, and you will 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.

Best practices are a set of commercially proven approaches to software development, which, when used in combination, strike at the root causes of software development problems. These are called "Best Practices," not because we can precisely quantify their value, but because they are observed to be commonly used in the industry by successful organizations. The Best Practices are harvested from thousands of customers on thousands of projects and from industry experts.

Module 1 Content Outline

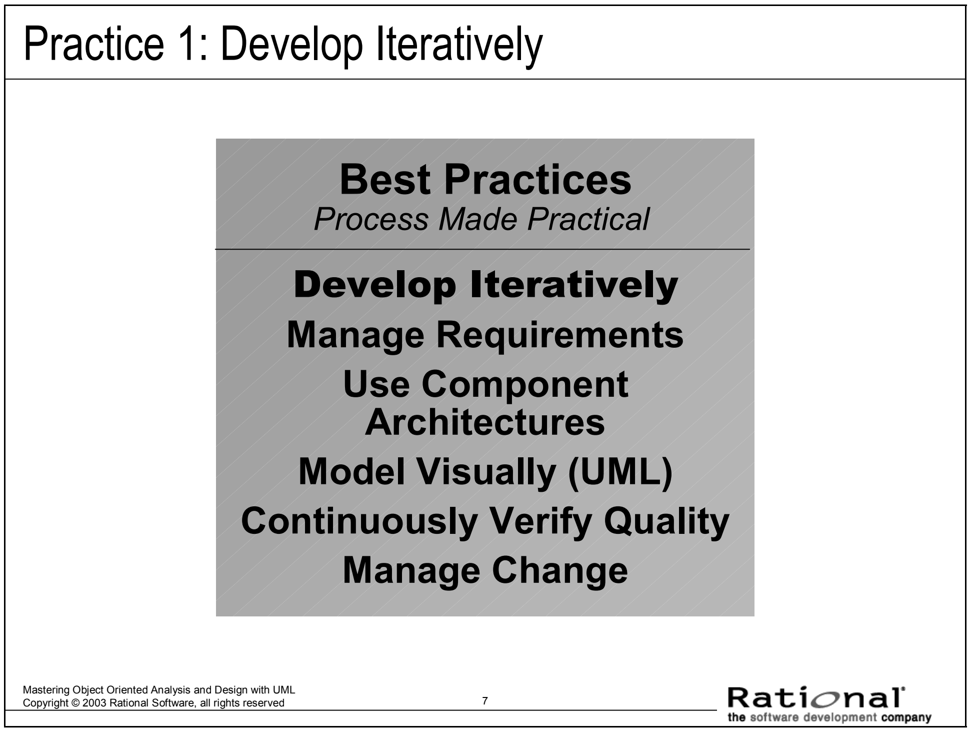


Module 1 Content Outline

- Software development problems
- ☆ The Six Best Practices
- RUP within the context of the Six Best Practices

6

Practice 1: Develop Iteratively



Practice 1: Develop Iteratively

Best Practices
Process Made Practical

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

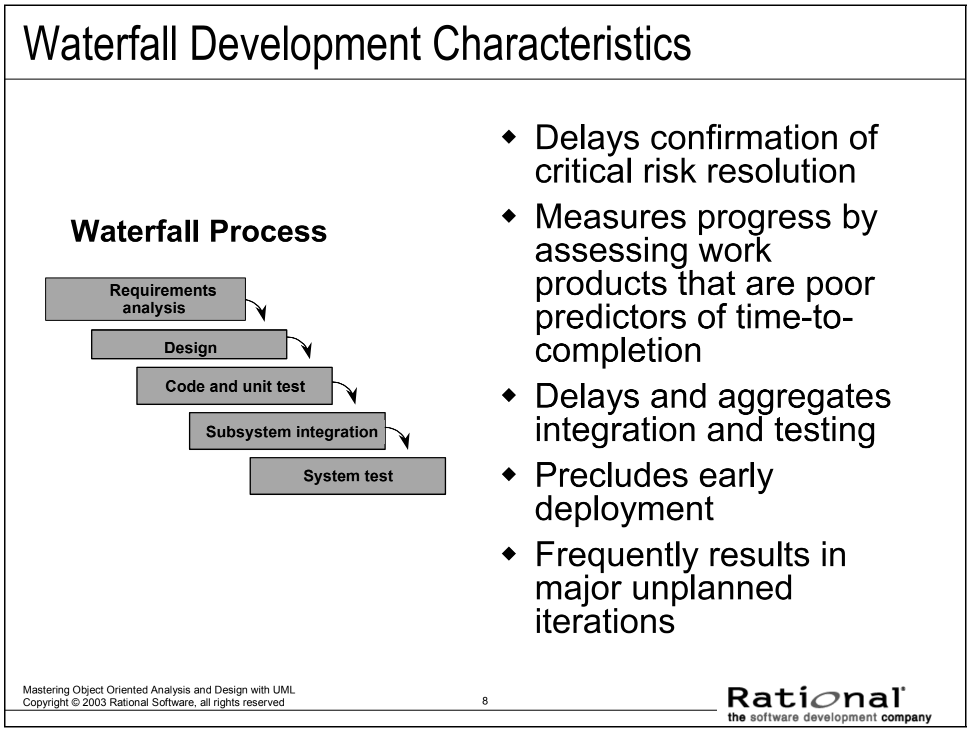
Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 7

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**.

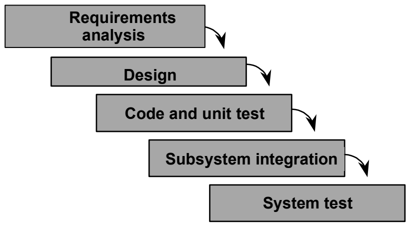
Each iteration is focused on defining, analyzing, designing, building, and testing a set of requirements.

Waterfall Development Characteristics



Waterfall Development Characteristics

Waterfall Process



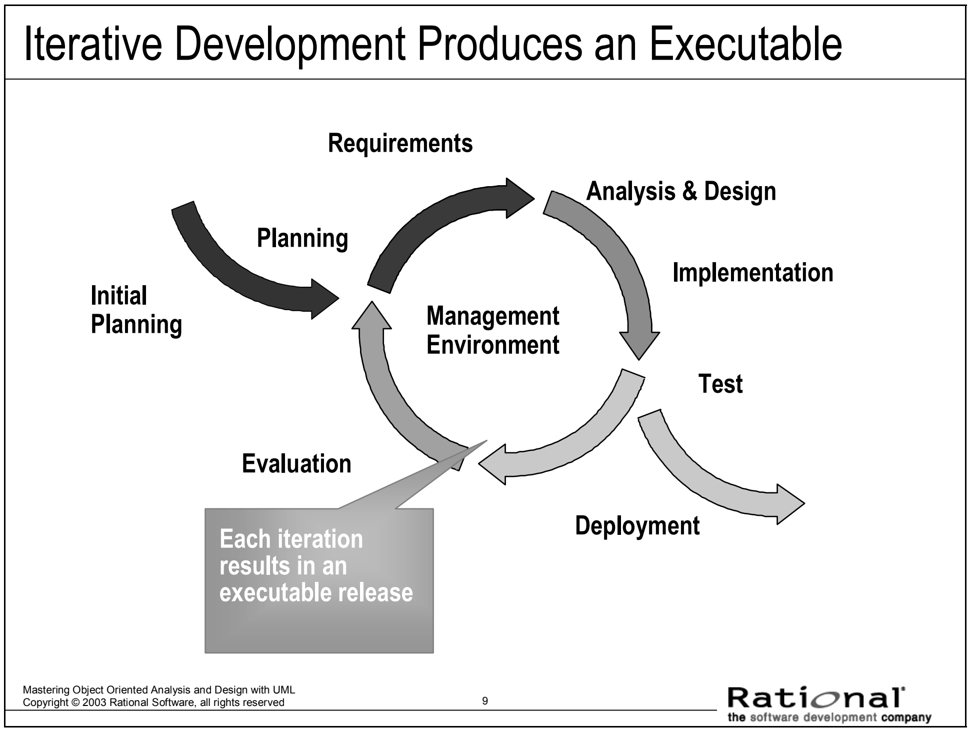
- Delays confirmation of critical risk resolution
- Measures progress by assessing work products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 8

Rational the software development company

Waterfall development is conceptually straightforward because it produces a single deliverable. The fundamental problem of this approach is that it pushes risk forward in time, when it is costly to undo mistakes from earlier phases. An initial design may be flawed with respect to its key requirements, and late discovery of design defects may 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.

Iterative Development Produces an Executable



Iterative Development Produces an Executable

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 9

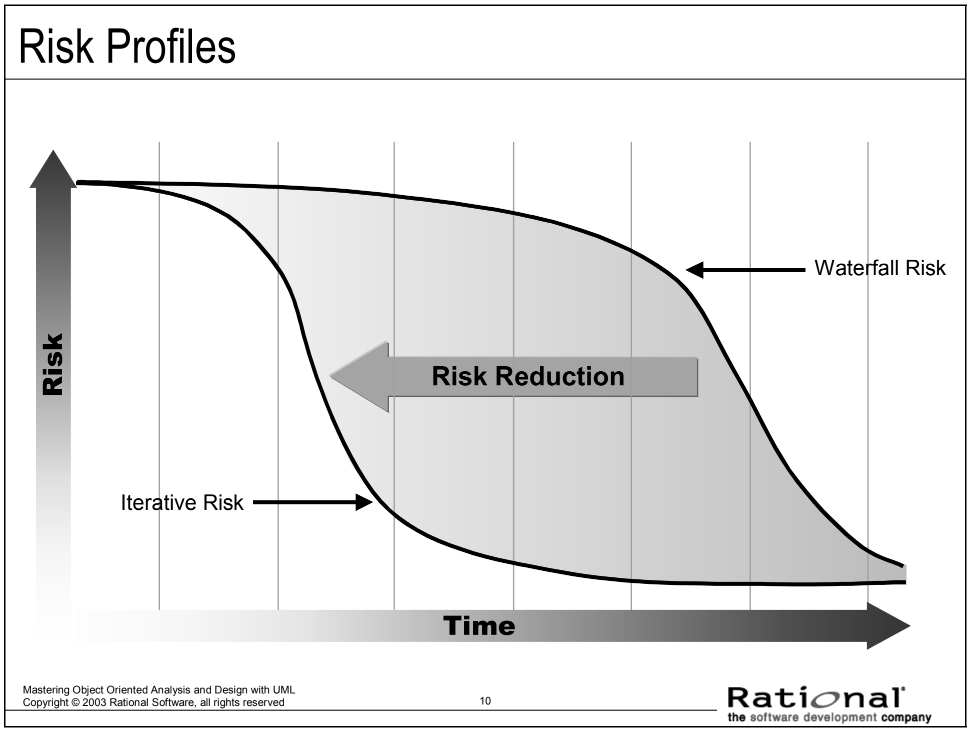
Rational
the software development company

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

- Resolve major risks before making large investments.
- Enable early user feedback.
- Make testing and integration continuous.
- Focus project short-term objective milestones.
- Make possible deployment of partial implementations.

Iterative processes were developed in response to these waterfall characteristics. With an iterative process, the waterfall steps are applied iteratively. Instead of developing the whole system in lock step, an increment (for example, a subset of system functionality) is selected and developed, then another increment, and so on. The selection of the first increment to be developed is based on risk, with the highest priority risks first. To address the selected risk(s), choose a subset of use cases. Develop the *minimal* set of use cases that will allow objective verification (that is, through a set of executable tests) of the risks that you have chosen. Then select the next increment to address the next-highest risk, and so on. Thus you apply the waterfall approach within each iteration, and the system evolves incrementally.

Risk Profiles



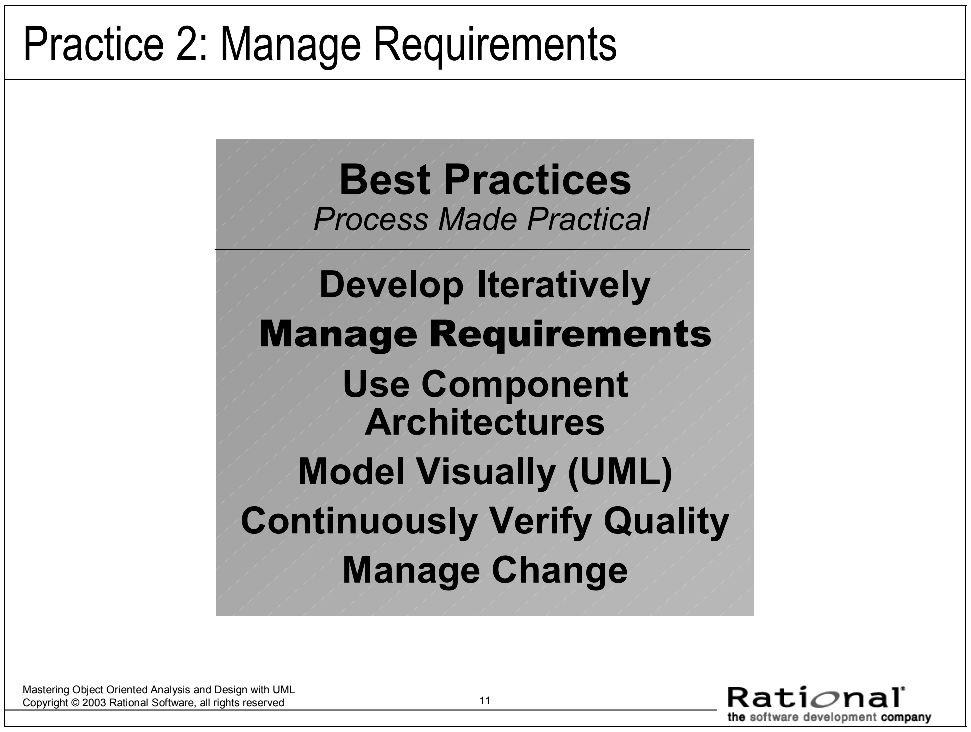
Risk Profiles

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 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 (for example, performance, fault tolerance, and maintainability) are tangible earlier in the process. Thus, issues are still correctable without jeopardizing target costs and schedules.

Practice 2: Manage Requirements



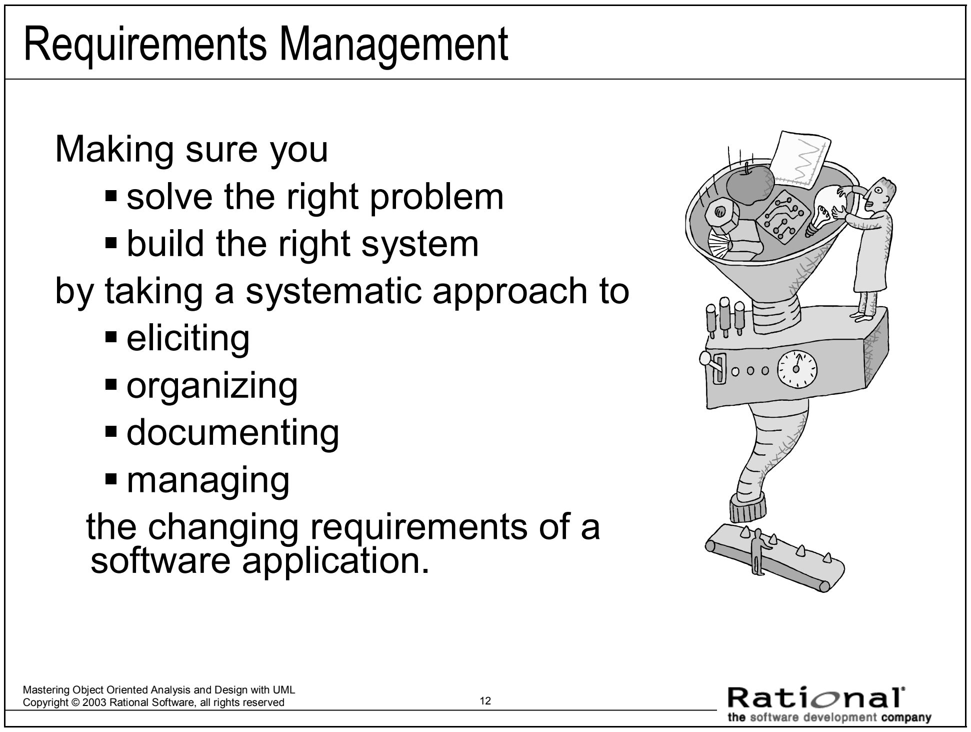
Practice 2: Manage Requirements

Best Practices
Process Made Practical

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

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 (canceled) projects accounted for 31.1%. These failures are attributed to incorrect requirements definition from the start of the project to poor requirements management throughout the development lifecycle. (Source: *Chaos Report*, http://www.standishgroup.com)

Requirements Management

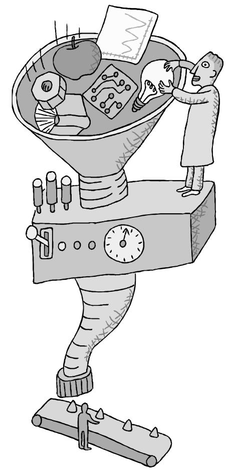


Requirements Management

Making sure you
- solve the right problem
- build the right system

by taking a systematic approach to
- eliciting
- organizing
- documenting
- managing

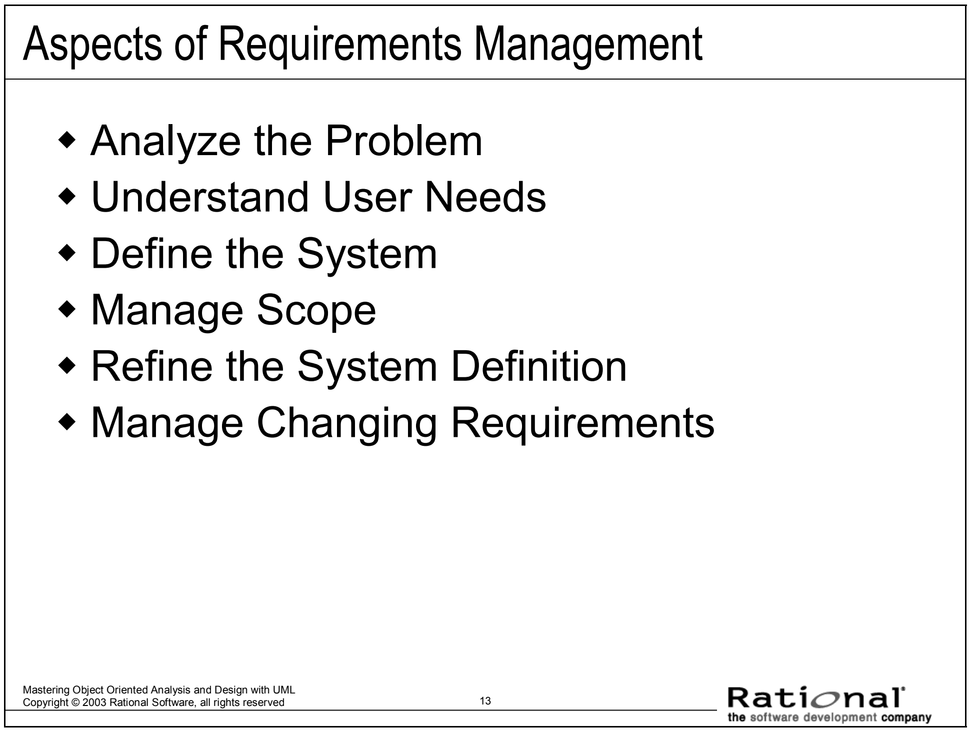
the changing requirements of a software application.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 12

Rational the software development company

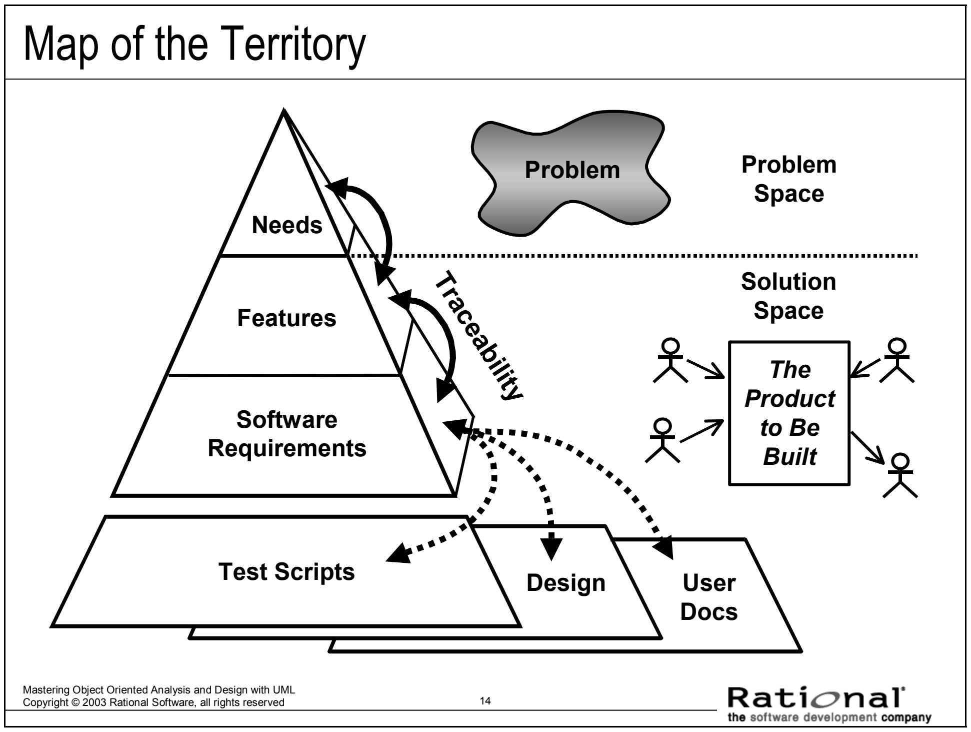
Aspects of Requirements Management



Aspects of Requirements Management

- Analyze the Problem
- Understand User Needs
- Define the System
- Manage Scope
- Refine the System Definition
- Manage Changing Requirements

Map of the Territory



Map of the Territory

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 14

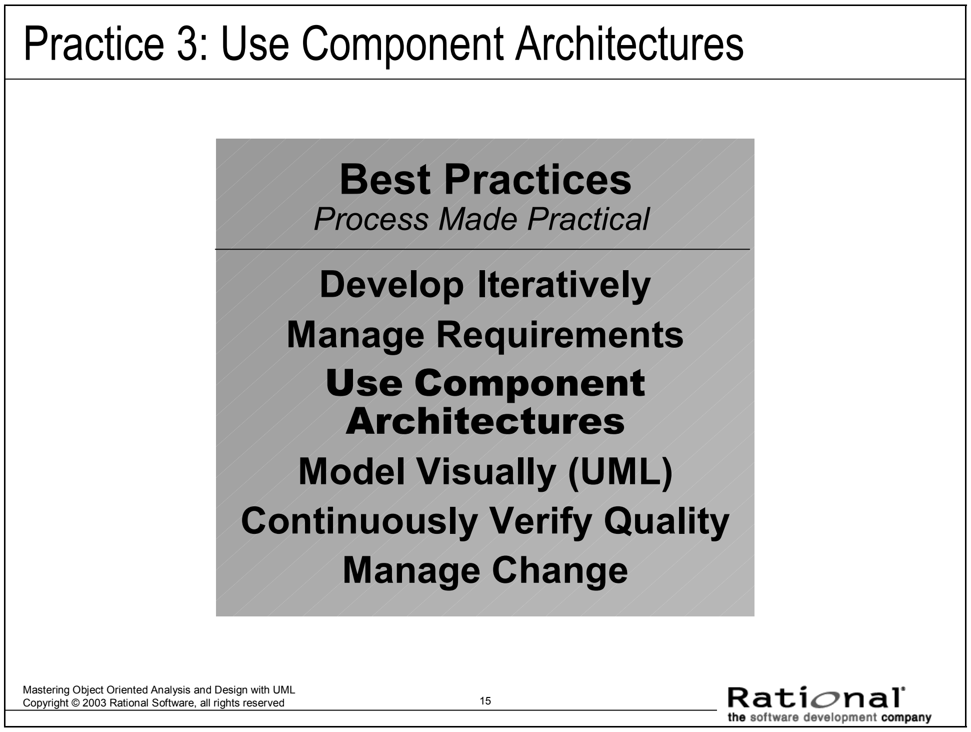
Rational the software development company

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 nonfunctional requirements. Detailed specifications are translated into test procedures, design, and user documentation.

Traceability allows us to:

- Assess the project impact of a change in a requirement.
- Assess the impact of a failure of a test on requirements (that is, if the 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 is intended to do.
- Manage change.

Practice 3: Use Component Architectures



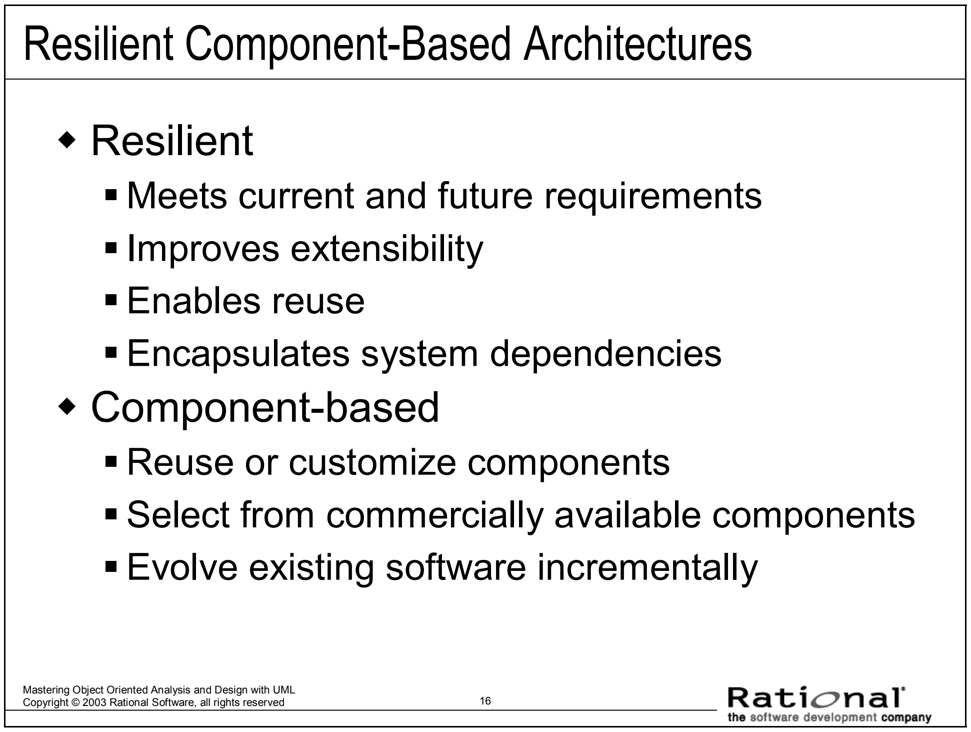
Practice 3: Use Component Architectures

Best Practices
Process Made Practical

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

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 and Rick Kazman [1998] Addison-Wesley). The Software Engineering Institute (SEI) has an effort underway called the Architecture Tradeoff Analysis (ATA) Initiative that focuses 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. As a result of performing architecture evaluations, AT&T reported a 10 percent productivity increase (from news@sei, Vol. 1, No. 2).

Resilient Component-Based Architectures



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

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 16

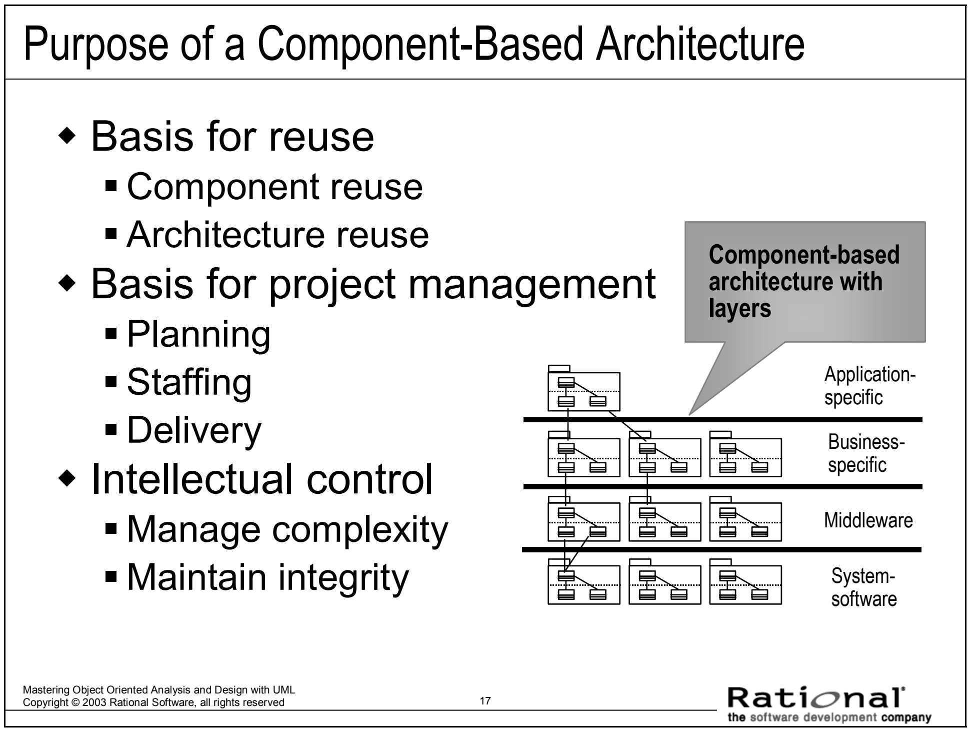
Rational the software development company

Architecture is a part 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 system 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 the 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.

Purpose of a Component-Based Architecture



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

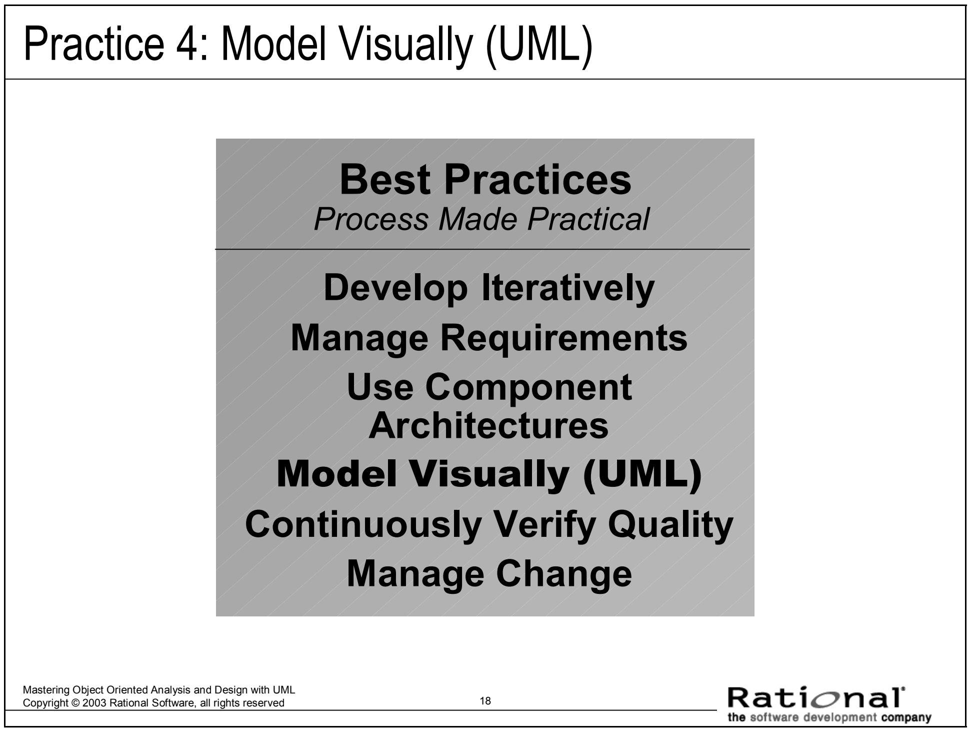
Component-based architecture with layers
Application-specific
Business-specific
Middleware
System-software

Definition of a (software) component:

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

Practice 4: Model Visually



Practice 4: Model Visually (UML)

Best Practices
Process Made Practical

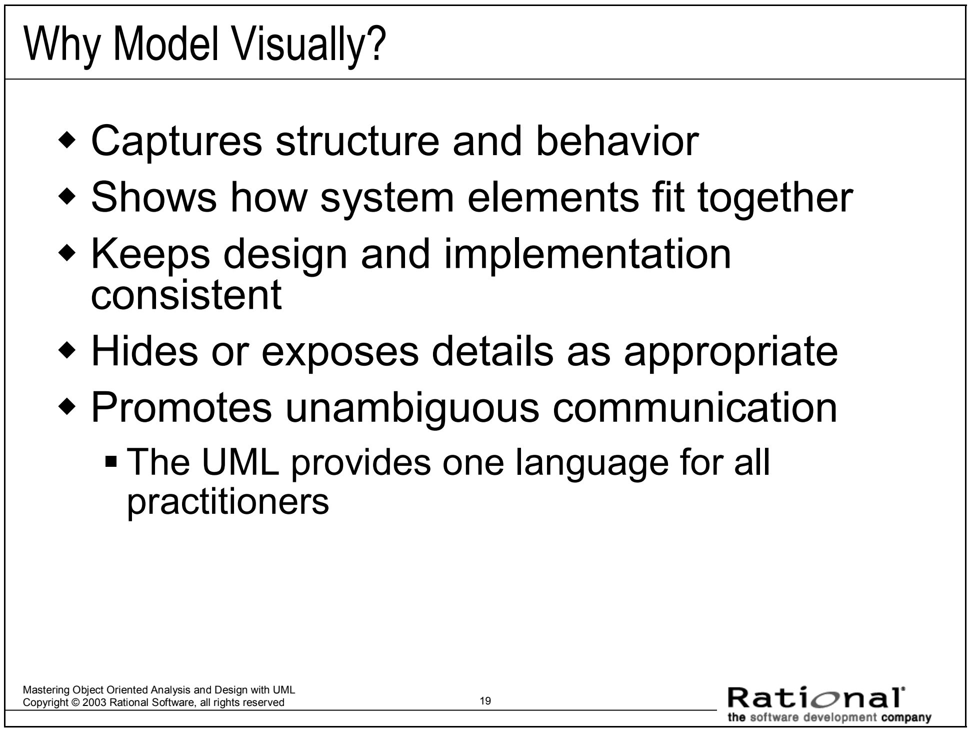
Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 18

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 building. We build models of complex systems because we cannot comprehend any such system in its entirety.

Why Model Visually?



Why Model Visually?

- Captures structure and behavior
- Shows how system elements fit together
- Keeps design and implementation consistent
- Hides or exposes details as appropriate
- Promotes unambiguous communication
  - The UML provides one language for all practitioners

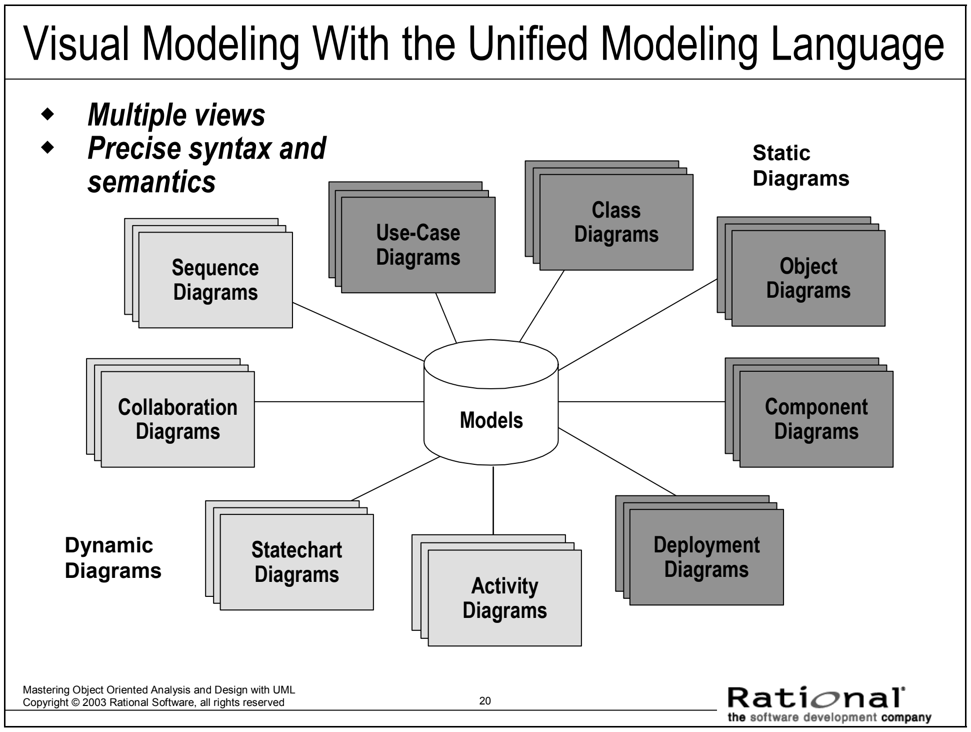
Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved
19

Rational
the software development company

Modeling is important because it helps the development team visualize, specify, construct, and document the structure and behavior of system 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 system artifacts - its requirements, designs, and implementations. In short, visual modeling helps improve a team's ability to manage software complexity.

Visual Modeling With the Unified Modeling Language



Visual Modeling With the Unified Modeling Language

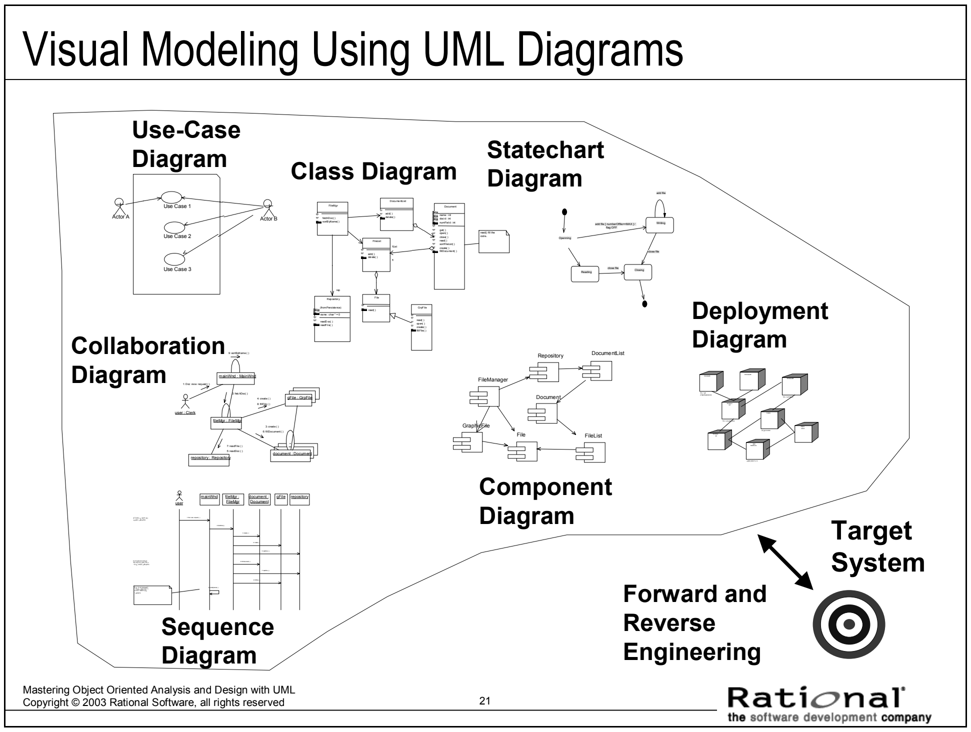
- ***Multiple views***
- ***Precise syntax and semantics***

20
Rational the software development company

In building a visual model of a system, many different diagrams are needed to represent different views of the system. The UML provides a rich notation for visualizing models. This includes the following key diagrams:

- Use-Case diagrams to illustrate user interactions with the system
- Class diagrams to illustrate logical structure
- Object diagrams to illustrate objects and links
- Component diagrams to illustrate physical structure of the software
- Deployment diagrams to show the mapping of software to hardware configurations
- Activity diagrams to illustrate flows of events
- Statechart diagrams to illustrate behavior
- Interaction diagrams (that is, collaboration and sequence diagrams) to illustrate behavior

Visual Modeling Using UML Diagrams



Visual Modeling Using UML Diagrams

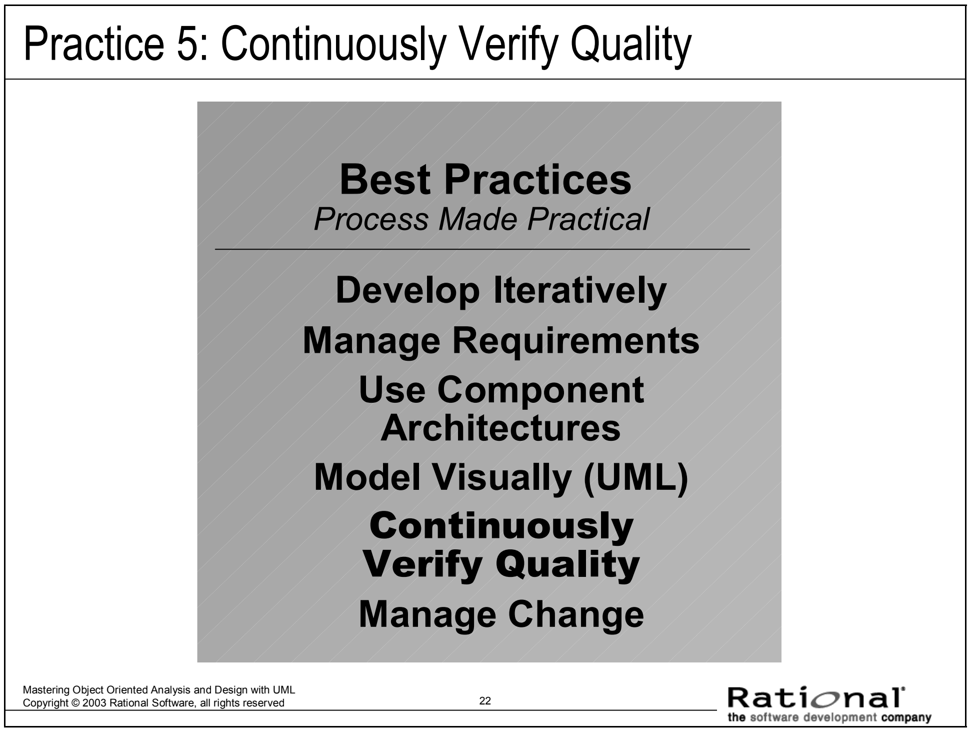
Rational the software development company

Visual modeling with the UML makes application architecture tangible, permitting us to assess it in multiple dimensions. How portable is it? Can it exploit expected advances in parallel processing? How can you modify it to support a family of applications?

We have 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.

Advances in forward and reverse engineering techniques permit changes to an application's model to be reflected automatically 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, in which we expect such changes with each iteration.

Practice 5: Continuously Verify Quality


Practice 5: Continuously Verify Quality

Best Practices
Process Made Practical

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

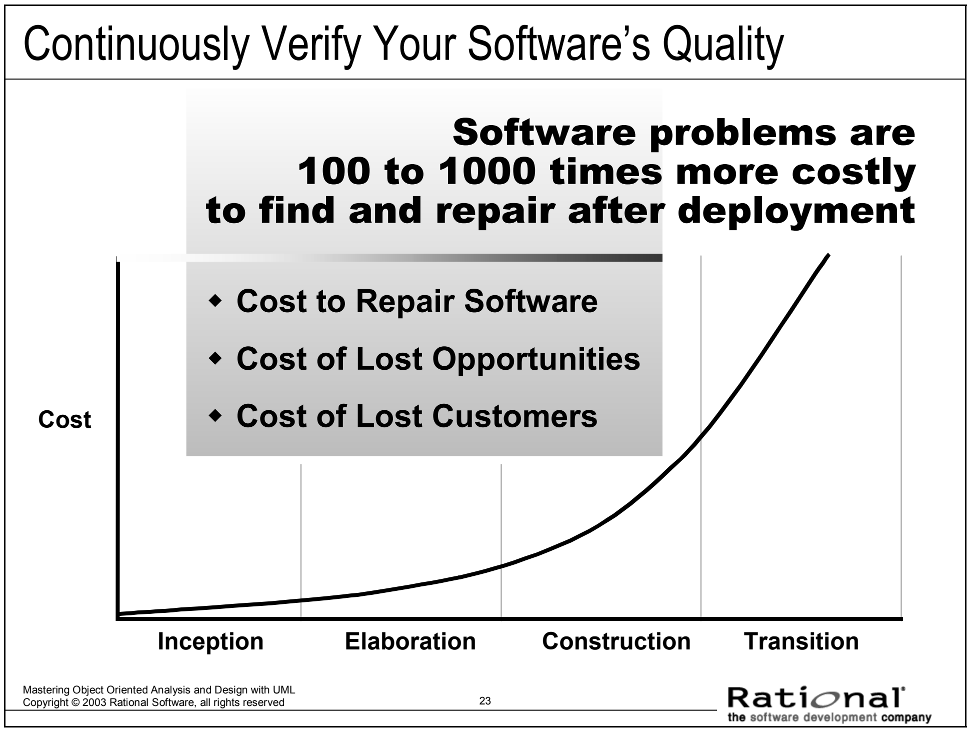
Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 22

Rational the software development company

Quality, as used within RUP, is defined as "The characteristic of having demonstrated the achievement of producing a product which meets or exceeds agreed-upon requirements, as measured by agreed-upon measures and criteria, and is produced by an agreed-upon process." Given this definition, achieving quality is not simply "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 the desired degree of quality (and can be repeated and managed).

In many organizations, software testing accounts for 30% to 50% of software development costs. Yet most people believe that software is not well-tested before it is delivered. This contradiction is rooted in two clear facts. First, testing software is enormously difficult. The different ways a particular program can behave are almost infinite. Second, testing is typically done without a clear methodology and without the required automation or tool support. While the complexity of software makes complete testing an impossible goal, a well-conceived methodology and use of state-of-the-art tools can greatly improve the productivity and effectiveness of the software testing.

Continuously Verify Your Software's Quality



Continuously Verify Your Software's Quality

Software problems are 100 to 1000 times more costly to find and repair after deployment

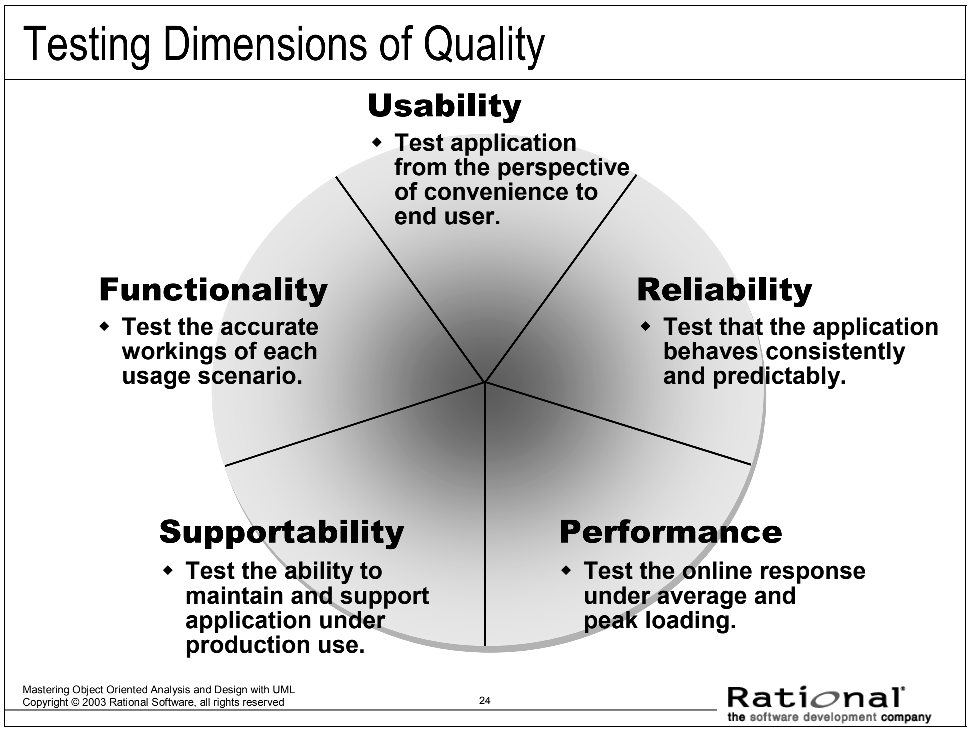
- Cost to Repair Software
- Cost of Lost Opportunities
- Cost of Lost Customers

Cost
Inception
Elaboration
Construction
Transition

This principle is driven by a fundamental and well-known property of software development: It is a lot less expensive to correct defects during development than to correct them after deployment.

- Tests for key scenarios ensure that all requirements are properly implemented.
- Poor application performance hurts as much as poor reliability.
- Verify software reliability — memory leaks, bottlenecks.
- Test every iteration — automate test.

Testing Dimensions of Quality



Testing Dimensions of Quality

Usability
- **Test application from the perspective of convenience to end user.**

Functionality
- **Test the accurate workings of each usage scenario.**

Reliability
- **Test that the application behaves consistently and predictably.**

Supportability
- **Test the ability to maintain and support application under production use.**

Performance
- **Test the online response under average and peak loading.**

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 24

Rational the software development company

Functional testing verifies that a system executes the required use-case scenarios as intended. Functional tests may include the testing of features, usage scenarios, and security.

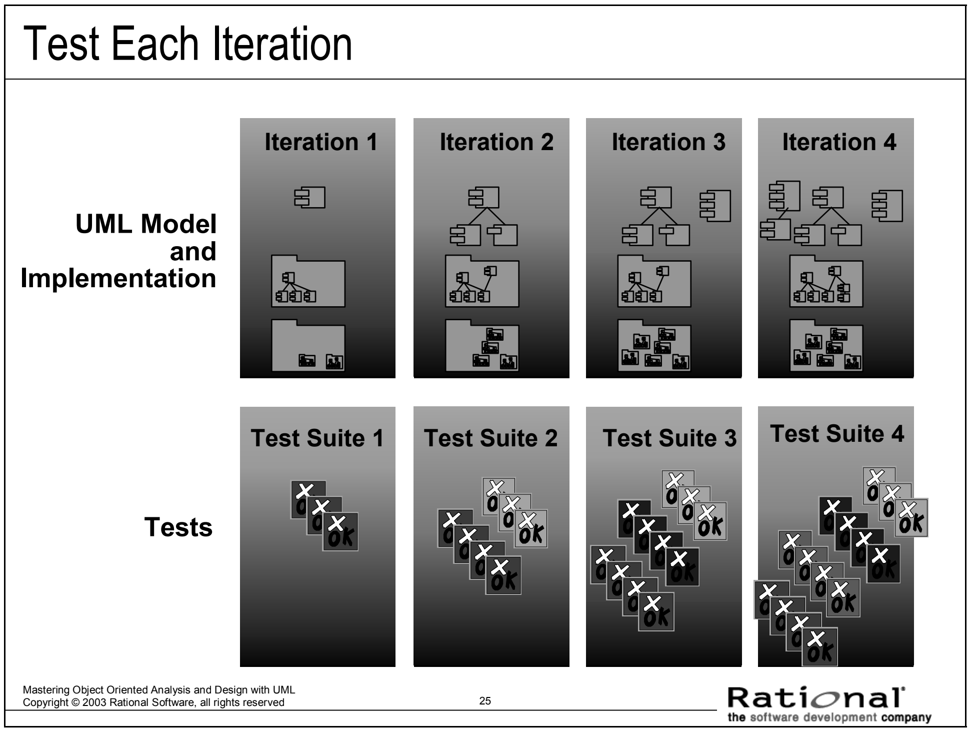
Usability testing evaluates the application from the user's perspective. Usability tests focus on human factors, aesthetics, consistency in the user interface, online and context-sensitive Help, wizards and agents, user documentation, and training materials.

Reliability testing verifies that the application performs reliably and is not prone to failures during execution (crashes, hangs, and memory leaks). Effective reliability testing requires specialized tools. Reliability tests include tests of integrity, structure, stress, contention, and volume.

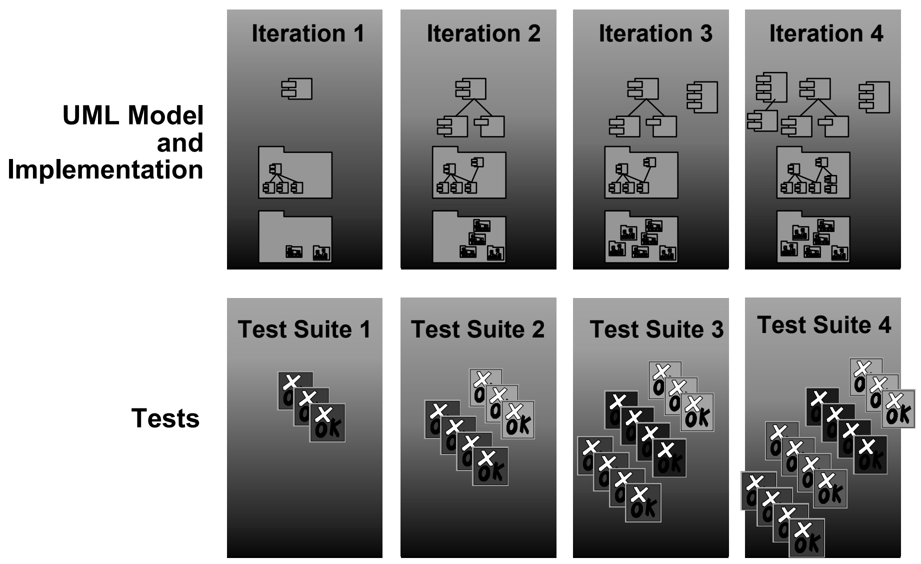
Performance testing checks that the target system works functionally and reliably under production load. Performance tests include benchmark tests, load tests, and performance profile tests.

Supportability testing verifies that the application can be deployed as intended. Supportability tests include installation and configuration tests.

Test Each Iteration



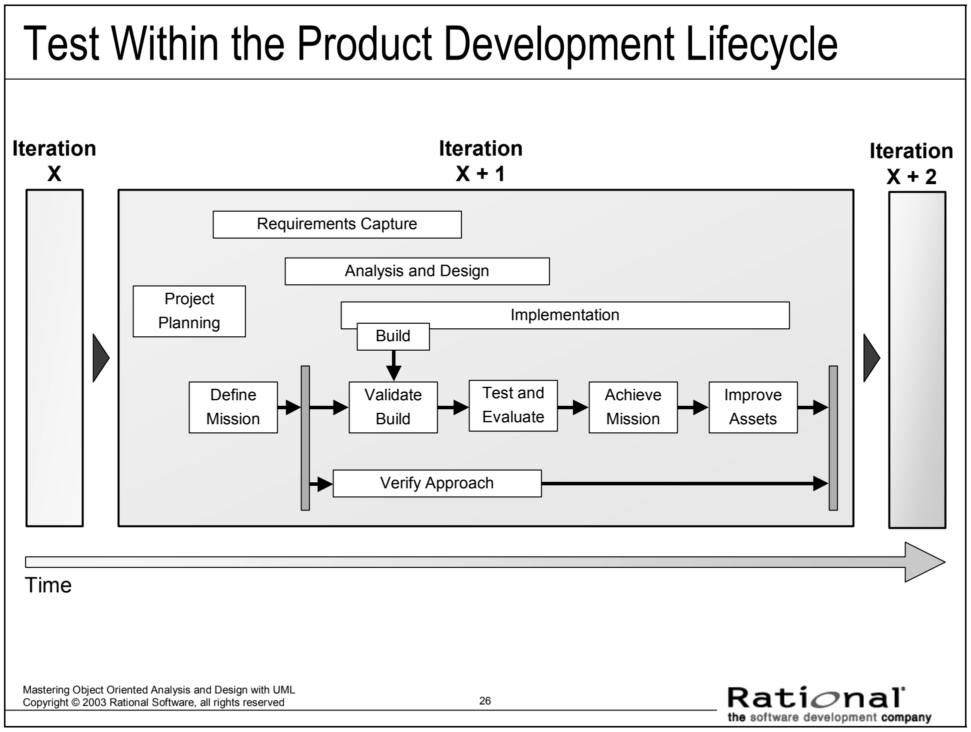
Test Each Iteration



25
Rational the software development company

In each iteration, automated tests are created that test the requirements addressed in that iteration. As new requirements are added in subsequent iterations, new tests are generated and run. At times, a requirement may be changed in a later iteration. In that case, the tests associated with the changed requirement may be modified or simply regenerated by an automated tool.

Test Within the Product Development Lifecycle



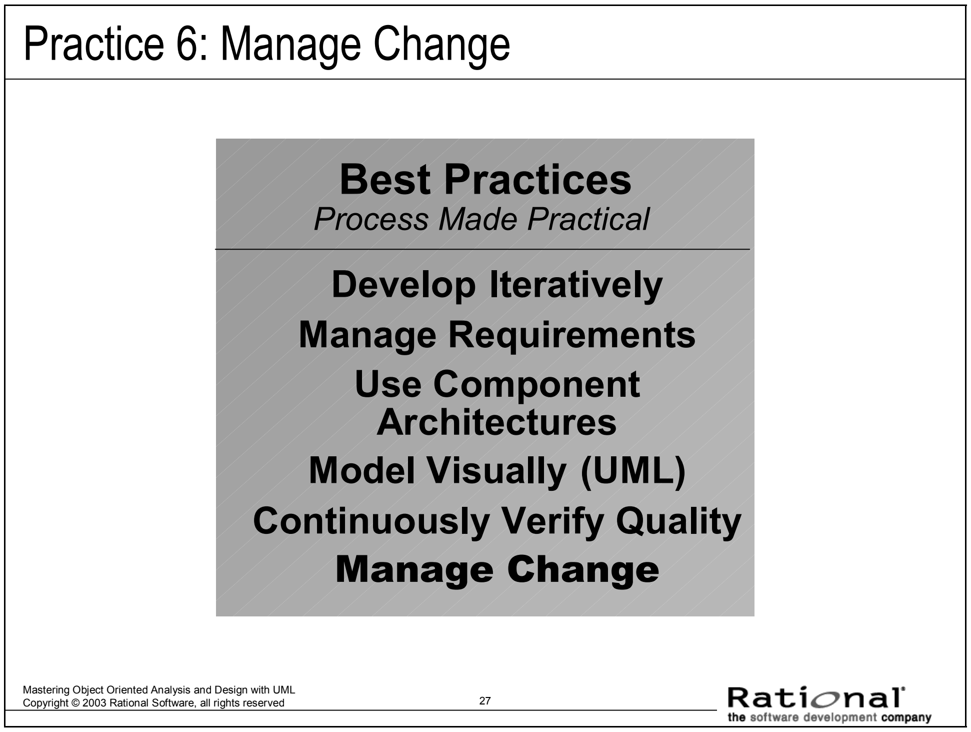
Test Within the Product Development Lifecycle

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 26

Rational the software development company

The testing lifecycle is a part of the software lifecycle; both should start in an equivalent time frame. The design and development process for tests can be as complex and arduous as the process for developing the software product itself. If tests do not start in line with the first executable software releases, the test effort will back load the discovery of potentially important problems until late in the development cycle.

Practice 6: Manage Change



Practice 6: Manage Change

Best Practices
Process Made Practical

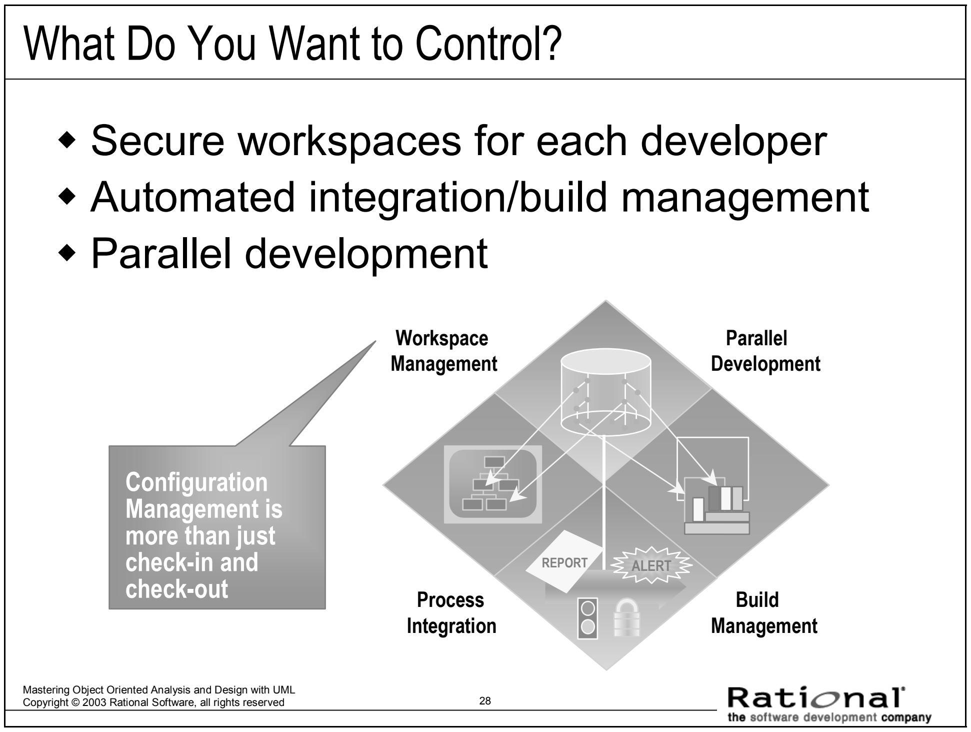
Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

You cannot stop change from being introduced into a project. However, you must control how and when changes are introduced into project artifacts, and who introduces those changes.

You must also synchronize changes across development teams and locations.

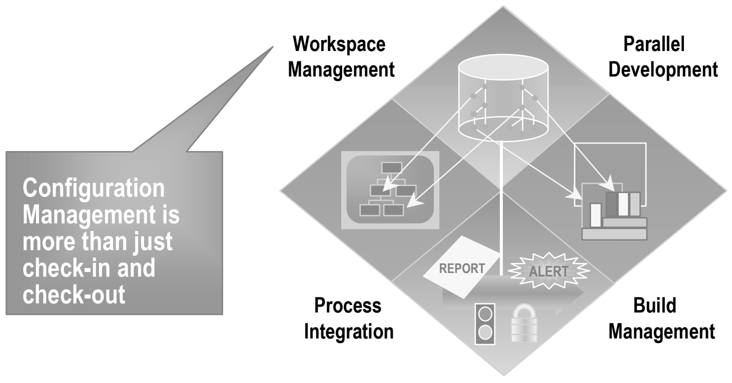
Unified Change Management (UCM) is the Rational Software approach to managing change in software system development, from requirements to release.

What Do You Want to Control?



What Do You Want to Control?

- Secure workspaces for each developer
- Automated integration/build management
- Parallel development



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 28
Rational the software development company

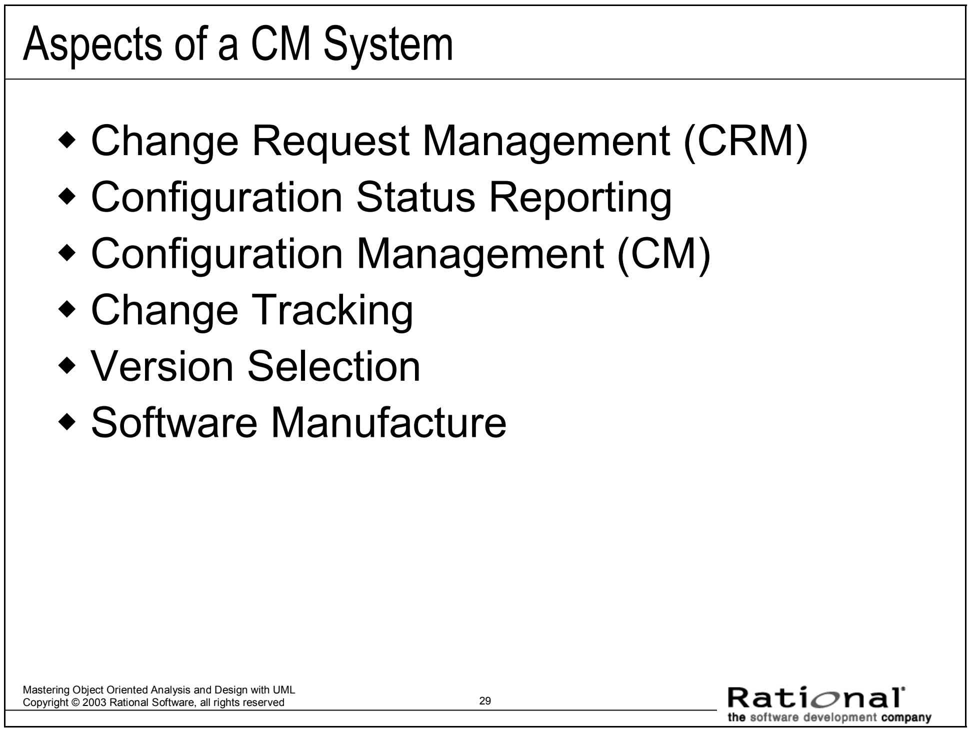
Establishing a secure workspace for each developer provides isolation from changes made in other workspaces and control of all software artifacts — models, code, documents and so forth.

A key challenge to developing software-intensive systems is the need to cope with multiple developers, 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 roles 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 users are not notified of the change.
- **Multiple versions**: With iterative development, it would not be unusual to have multiple versions of an artifact in different stages of development at the same time. For example, one release is in customer use, one is in test, and one is still in development. If a problem is identified in any one of the versions, the fix must be propagated among all of them.

Aspects of a CM System



Aspects of a CM System

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

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 29

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 Reporting (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, through either audits or raw data, are useful in determining the overall completeness of the project.

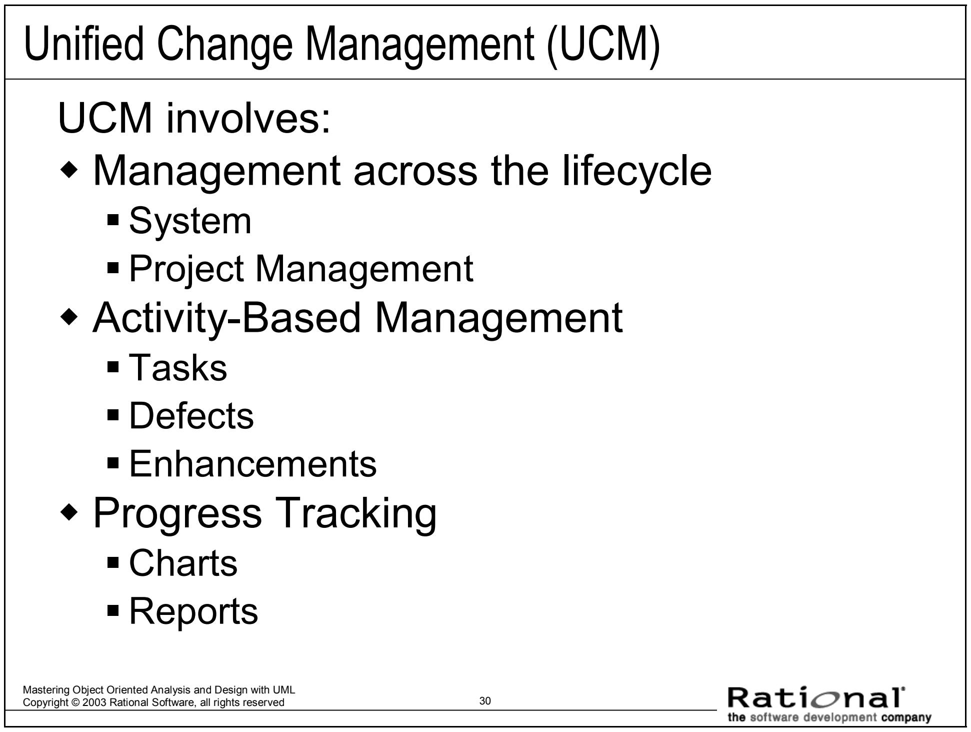
Configuration Management (CM) describes the product structure and identifies its constituent configuration items, which are treated as single versionable entities in the configuration management process. CM deals with defining configurations, building and labeling, and collecting versioned artifacts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

Change Tracking describes what is done to components for what reason and at what time. It serves as the history of and rationale for 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.

Unified Change Management (UCM)



Unified Change Management (UCM)

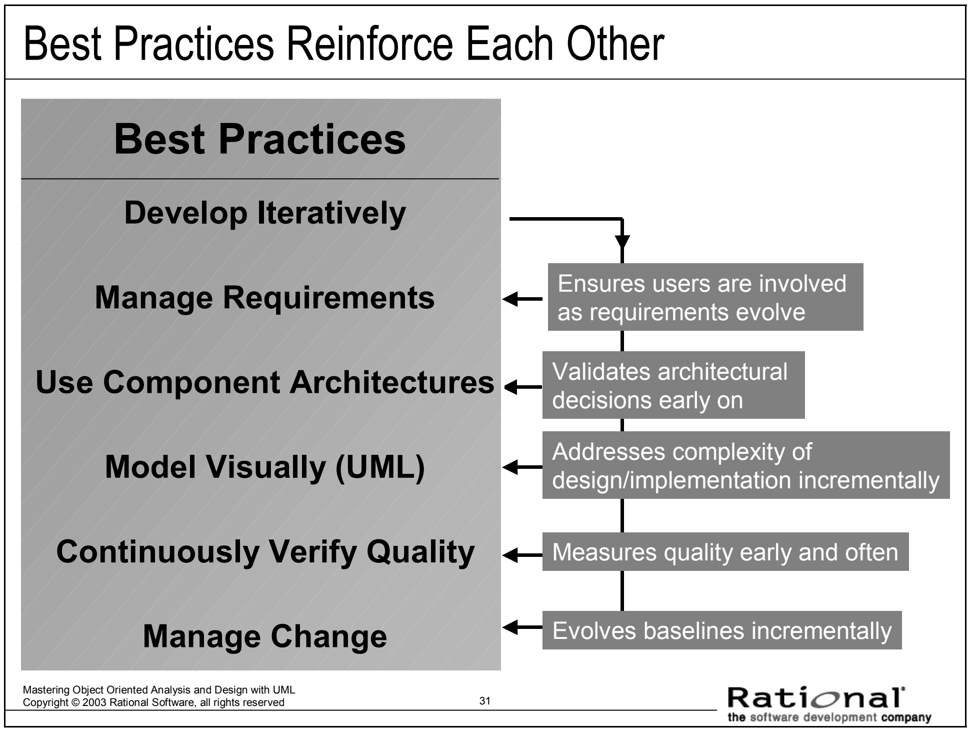
UCM involves:

- Management across the lifecycle
  - System
  - Project Management
- Activity-Based Management
  - Tasks
  - Defects
  - Enhancements
- Progress Tracking
  - Charts
  - Reports

Unified Change Management (UCM) is Rational Software's approach to managing change in software system development, from requirements to release. UCM spans the development lifecycle, defining how to manage change to requirements, design models, documentation, components, test cases, and source code.

One of the key aspects of the UCM model is that it unifies the activities used to plan and track project progress and the artifacts undergoing change.

Best Practices Reinforce Each Other



Best Practices Reinforce Each Other

Best Practices
Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change
Ensures users are involved as requirements evolve
Validates architectural decisions early on
Addresses complexity of design/implementation incrementally
Measures quality early and often
Evolves baselines incrementally

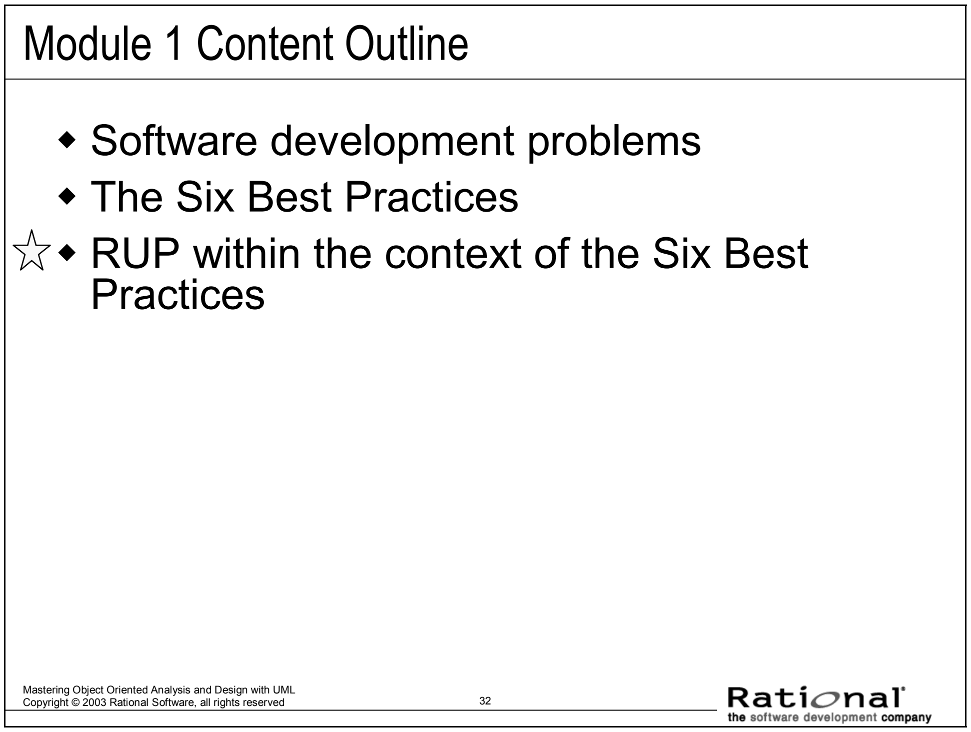
Mastering Object Oriented Analysis and Design with UML

31
Rational the software development company

In the case of our Six Best Practices, the whole is much greater than the sum of the parts. Each of the Best Practices reinforces and, in some cases, enables the others. This slide shows just one example: how iterative development leverages the other five Best Practices. However, each of the other five practices also enhances iterative development. For example, iterative development done without adequate requirements management can easily fail to converge on a solution. Requirements can change at will, users cannot agree, and the iterations go on forever.

When requirements are managed, this is less likely to happen. Changes to requirements are visible, and the impact on the development process is assessed before the changes are accepted. Convergence on a stable set of requirements is ensured. Similarly, each pair of Best Practices provides mutual support. Hence, although it is possible to use one Best Practice without the others, it is not recommended, since the resulting benefits will be significantly decreased.

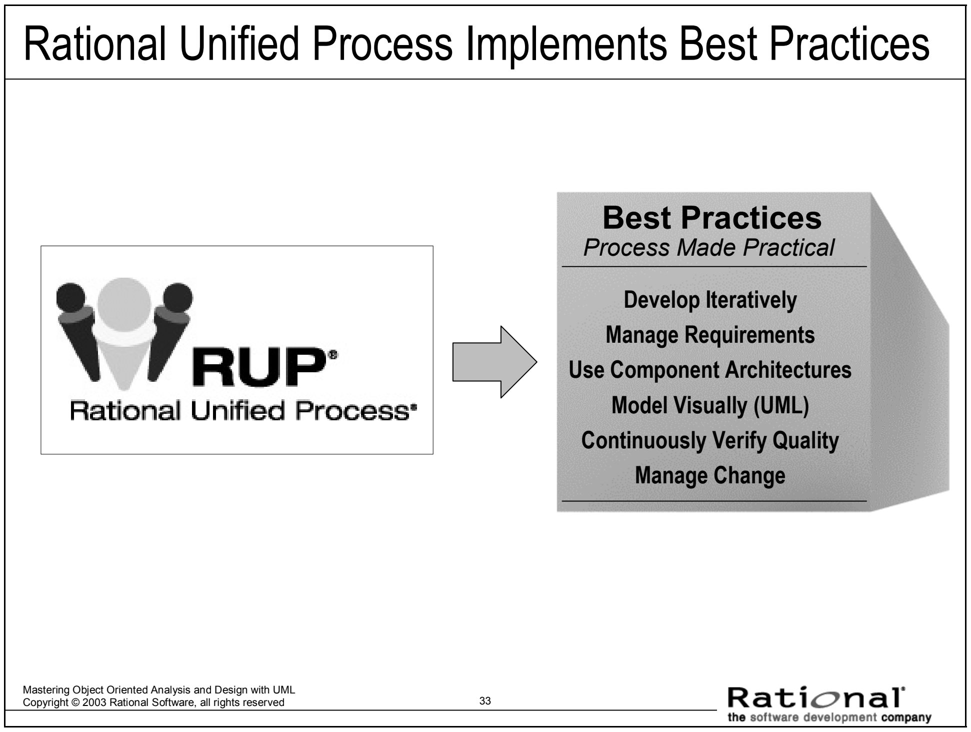
Module 1 Content Outline



Module 1 Content Outline

- Software development problems
- The Six Best Practices
- ☆ RUP within the context of the Six Best Practices

Rational Unified Process Implements Best Practices



Rational Unified Process Implements Best Practices

RUP®
Rational Unified Process®
Best Practices
Process Made Practical
Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change
Rational®
the software development company

Mastering Object Oriented Analysis and Design with UML

33

Why have a process?

- Provides guidelines for efficient development of quality software
- Reduces risk and increases predictability
- Promotes a common vision and culture
- Captures and institutionalizes Best Practices

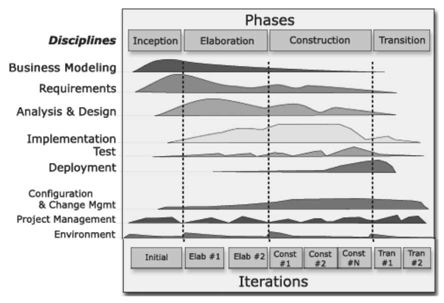
The Rational Unified Process (RUP) is a generic business process for object-oriented software engineering. It describes a family of related software-engineering processes sharing a common structure and a common process architecture. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its 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 RUP captures the Best Practices in modern software development in a form that can be adapted for a wide range of projects and organizations.

The UML provides a standard for the artifacts of 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 development also requires employing an equally robust development process, which is where the RUP comes in.

Achieving Best Practices

Achieving Best Practices

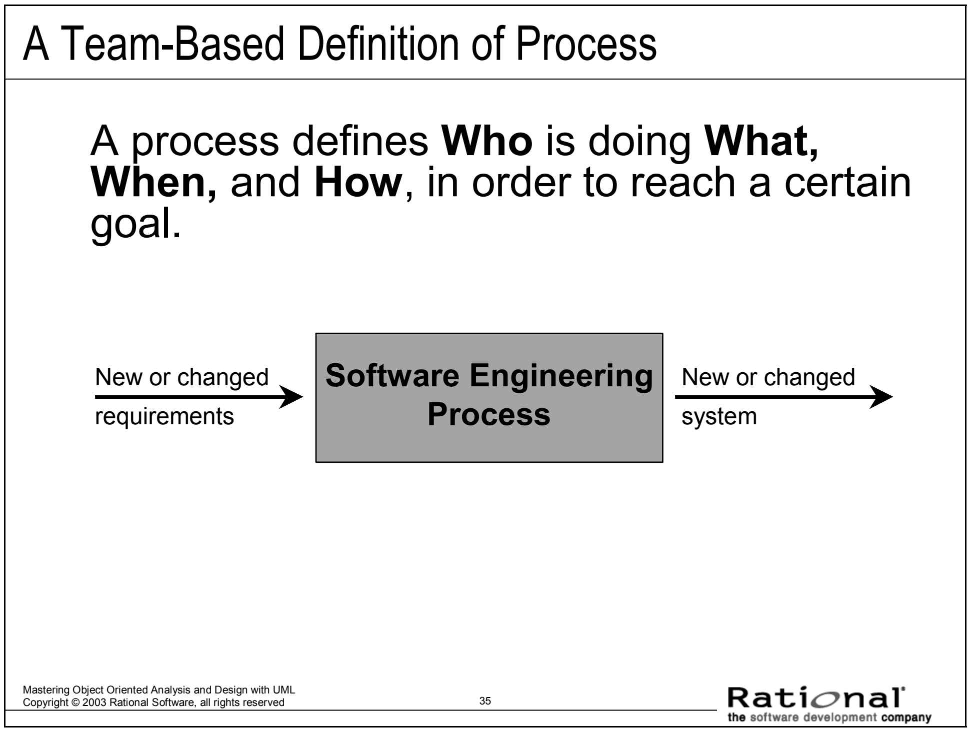
- Iterative approach
- Guidance for activities and artifacts
- Process focus on architecture
- Use cases that drive design and implementation
- Models that abstract the system



Examples:

- The dynamic structure (phases and iterations) of the Rational Unified Process creates the basis of iterative development.
- The Project Management discipline describes how to set up and execute a project using phases and iterations.
- The Use-Case Model of the Requirements discipline and the risk list determine what functionality you implement in an iteration.
- The Workflow Details of the Requirements discipline show the activities and artifacts that make requirements management possible.
- The iterative approach allows you to progressively identify components, and to decide which one to develop, which one to reuse, and which one to buy.
- The Unified Modeling Language (UML) used in the process represents the basis of visual modeling and has become the de facto modeling language standard.
- The focus on software architecture allows you to articulate the structure: the components, the ways in which they integrate, and the fundamental mechanisms and patterns by which they interact.

A Team-Based Definition of Process



A Team-Based Definition of Process

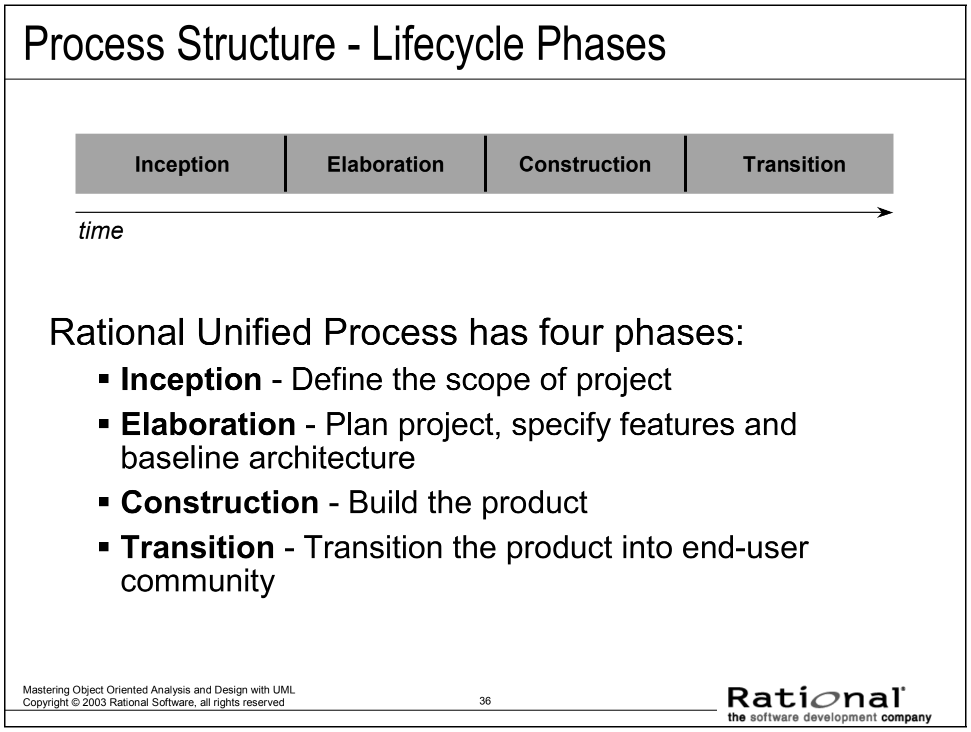
A process defines **Who** is doing **What, When,** and **How**, in order to reach a certain goal.

New or changed requirements → **Software Engineering Process** → New or changed system

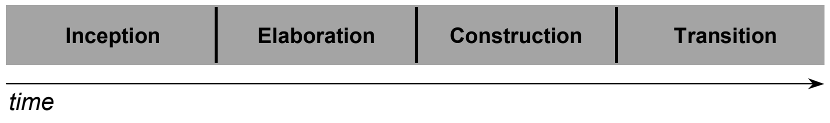
Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 35

Rational
the software development company

Process Structure - Lifecycle Phases



Process Structure - Lifecycle Phases



Rational Unified Process has four phases:

- **Inception** - Define the scope of project
- **Elaboration** - Plan project, specify features and baseline architecture
- **Construction** - Build the product
- **Transition** - Transition the product into end-user community

During Inception, we define the scope of the project: what is included and what is not. We do this by identifying all the actors and use cases, and by drafting the most essential use cases (typically 20% of the complete model). A business plan is developed to determine whether resources should be committed to the project.

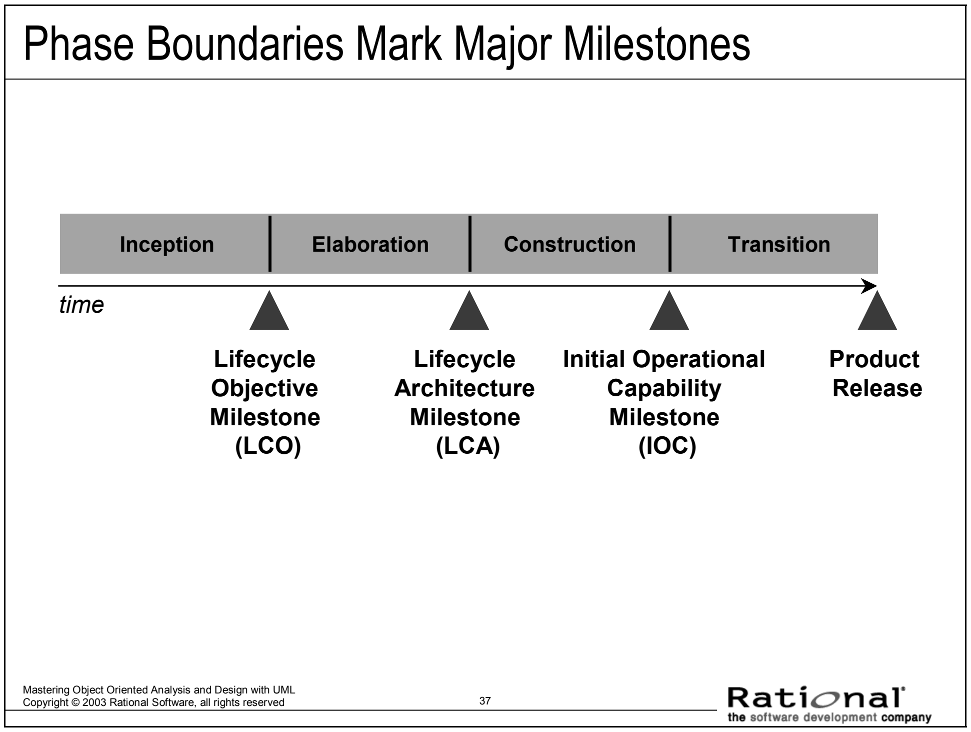
During Elaboration, we focus on two things: getting a good grasp of the requirements (80% complete) and establishing an architectural baseline. If we have a good grasp of the requirements and the architecture, we can eliminate a lot of the risks, and we will have a good idea of how much work remains to be done. We can make detailed cost/resource estimations at the end of Elaboration.

During Construction, we build the product in several iterations up to a beta release.

During Transition, we move the product to the end user and focus on end-user training, installation, and support.

The amount of time spent in each phase varies. For a complex project with many technical unknowns and unclear requirements, Elaboration may include three-to-five iterations. For a simple project, where requirements are known and the architecture is simple, Elaboration may include only a single iteration.

Phase Boundaries Mark Major Milestones



Phase Boundaries Mark Major Milestones

Inception
Elaboration
Construction
Transition
time
Lifecycle Objective Milestone (LCO)
Lifecycle Architecture Milestone (LCA)
Initial Operational Capability Milestone (IOC)
Product Release

At each of the major milestones, we review the project and decide whether to proceed with it as planned, to cancel the it, or to revise it. The criteria used to make these decisions vary by phase.

LCO: *scope is agreed upon and risks are understood and reasonable*

LCA: *high risks are addressed and architecture is stable*

IOC: *product is complete and quality is acceptable*

The evaluation criteria for the Inception phase (LCO) include: stakeholder concurrence on scope definition and cost/schedule estimates; requirements understanding as evidenced by the fidelity of the primary use cases; credibility of cost/schedule estimates, priorities, risks, and development process; depth and breadth of any architectural prototype; actual expenditures versus planned expenditures.

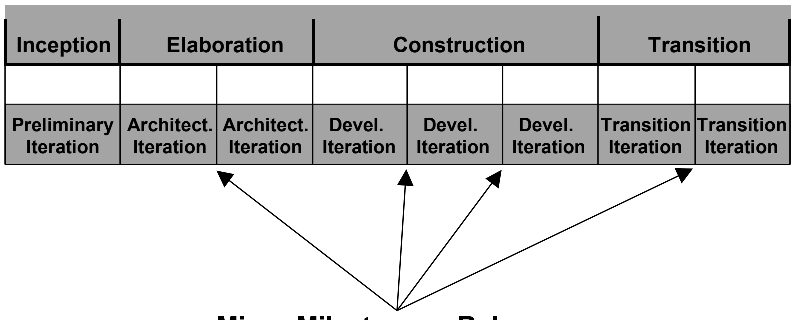
The evaluation criteria for the Elaboration phase (LCA) include: stability of the product vision and architecture; resolution of major risk elements; adequate planning and reasonable estimates for project completion; stakeholder acceptance of the product vision and project plan; and acceptable expenditure level.

The evaluation criteria for the Construction phase (IOC) include: stability and maturity of the product release (that is, is it ready to be deployed?); readiness of the stakeholders for the transition; and acceptable expenditure level.

At the end of the Transition phase, we decide whether to release the product. We base this primarily on the level of user satisfaction achieved during the Transition phase. Often this milestone coincides with the initiation of another development cycle to improve or enhance the product. In many cases, this new development cycle may already be underway.

Iterations and Phases

Iterations and Phases



Minor Milestones: Releases

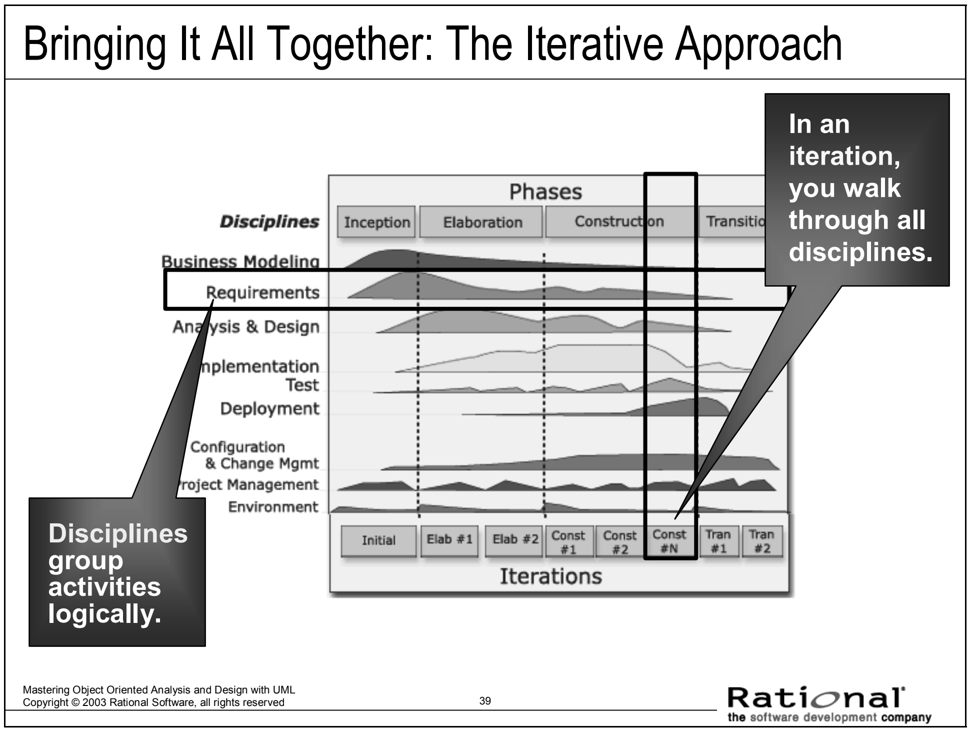
An **iteration** is a distinct sequence of activities based on an established plan and evaluation criteria, resulting in an executable release (internal or external).

Within each phase, there is a series of iterations. The number of iterations per phase will vary. Each iteration results in an executable release encompassing larger and larger subsets of the final application.

An internal release is kept within the development environment and (optionally) demonstrated to the stakeholder community. We provide stakeholders (usually users) with an external release for installation in their environment. External releases are much more expensive because they require user documentation and technical support — because of this, they normally occur only during the Transition phase.

The end of an iteration marks a minor milestone. At this point, we assess technical results and revise future plans as necessary.

Bringing It All Together: The Iterative Approach



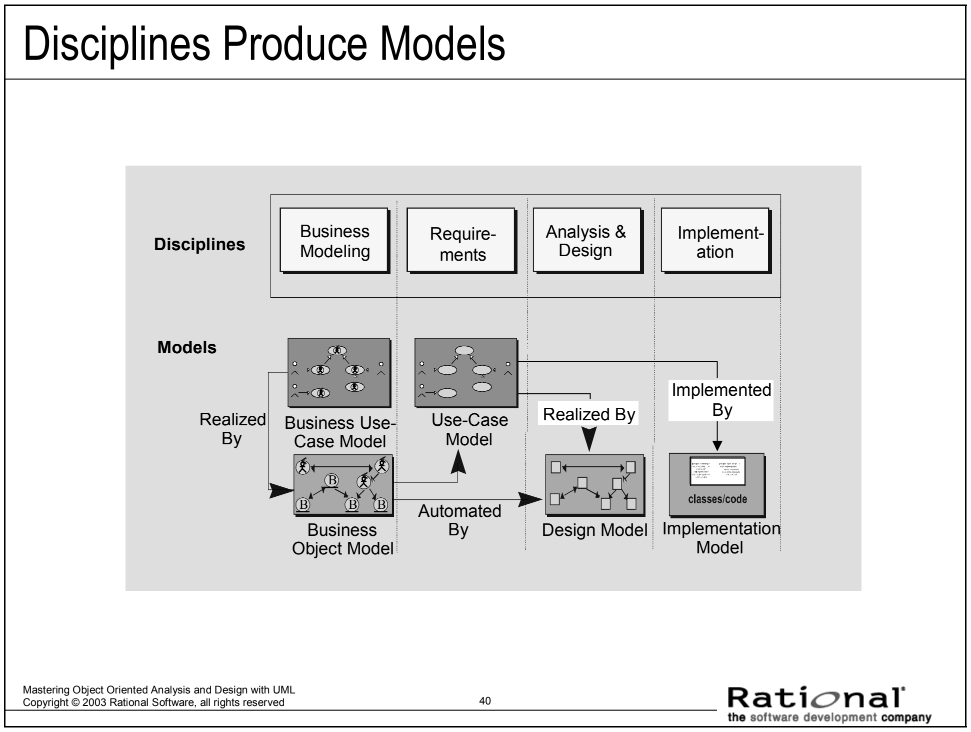
Bringing It All Together: The Iterative Approach

This slide illustrates how phases and iterations (the time dimension) relate to the development activities (the discipline dimension). The relative size of each color area in the graph indicates how much of the activity is performed in each phase or iteratino.

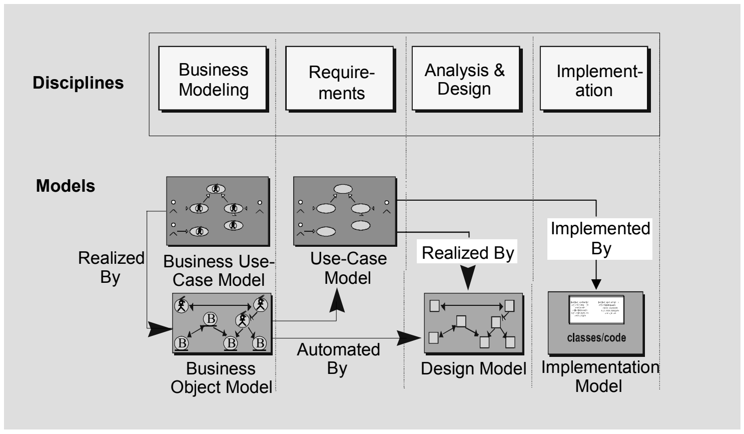
Each iteration involves activities from all disciplines. The relative amount of work related to the disciplines changes between iterations. For instance, during late Construction, the main work is related to Implementation and Test and very little work on Requirements is done.

Note that requirements are not necessarily complete by the end of Elaboration. It is acceptable to delay the analysis and design of well-understood portions of the system until Construction because they are low in risk.

Disciplines Produce Models



Disciplines Produce Models



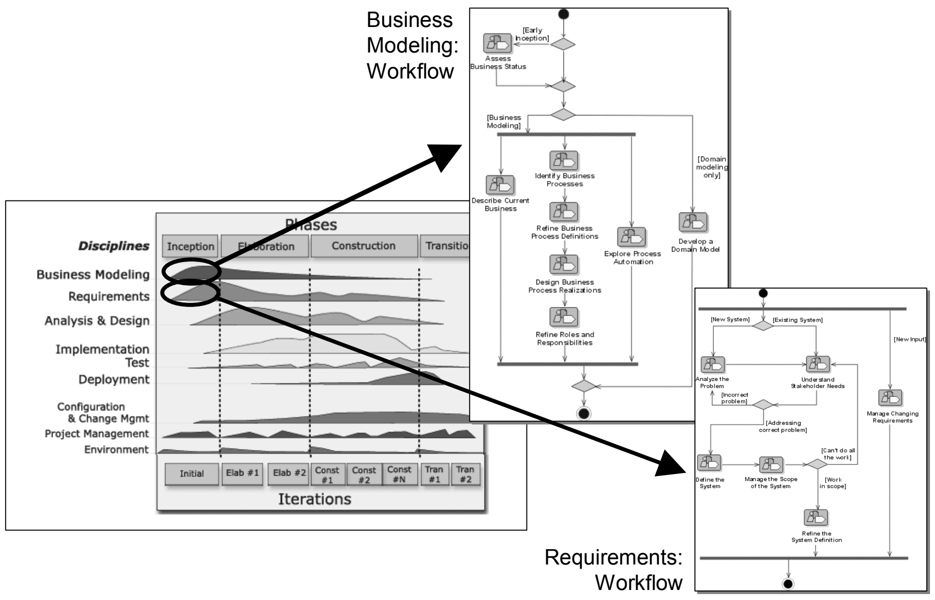
The RUP takes a model-driven approach. Several models are needed to fully describe the evolving system. Each major discipline produces one of those models. The models are developed incrementally across iterations.

- The **Business Model** is a model of what the business processes are and of the business environment. It can be used to generate requirements of supporting information systems.
- The **Use-Case Model** is a model of what the system is supposed to do and of the system environment.
- The **Design Model** is an object model describing the realization of use cases. It serves as an abstraction of the implementation model and its source code.
- The **Implementation Model** is a collection of components and the implementation subsystems that contain them.

Test Suites are derived from many of these models.

Disciplines Guide Iterative Development

Disciplines Guide Iterative Development



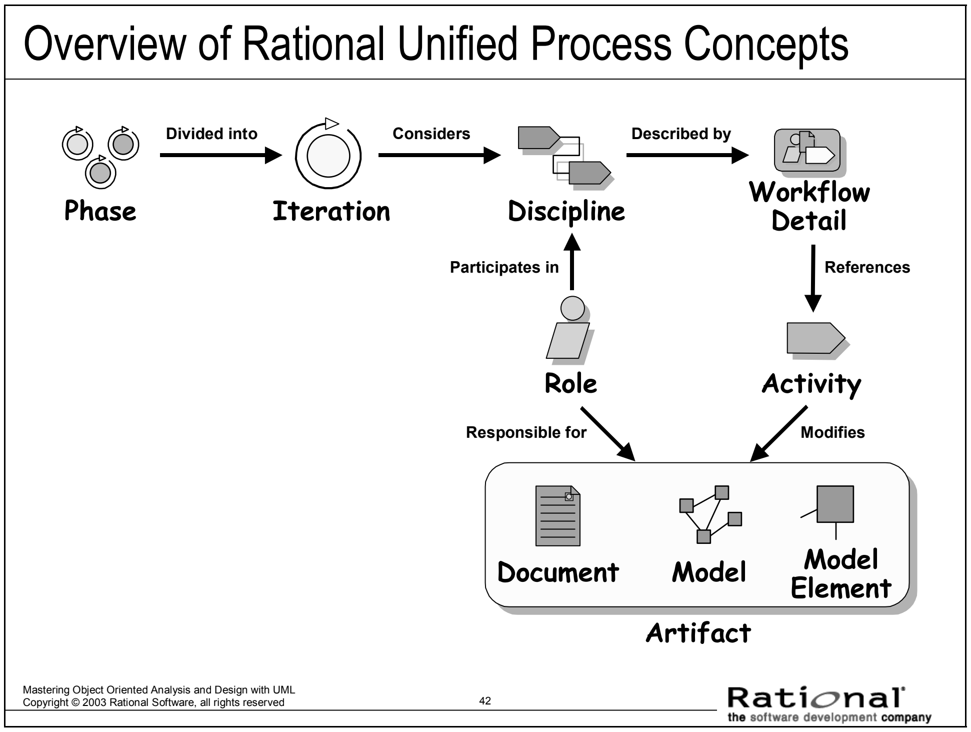
Mastering Object Oriented Analysis and Design with UML

41

Rational
the software development company

Within a discipline, workflows group activities that are done together. Discipline workflows will be present in varying degrees, depending on the phase.

Overview of Rational Unified Process Concepts



Overview of Rational Unified Process Concepts

Basic Concepts in the RUP

A software lifecycle in the RUP is decomposed over time into four sequential **phases**, each concluded by a major milestone. Each phase is essentially a span of time between two major milestones.

An **iteration** is a pass through a sequence of process disciplines. Each iteration concludes with the release of an executable product.

A **discipline** shows all the activities that you may go through to produce a particular set of artifacts. We describe these disciplines at an overview level — a summary of all roles, activities, and artifacts that are involved.

A **workflow detail** is a grouping of activities that are done "together," presented with input and resulting artifacts.

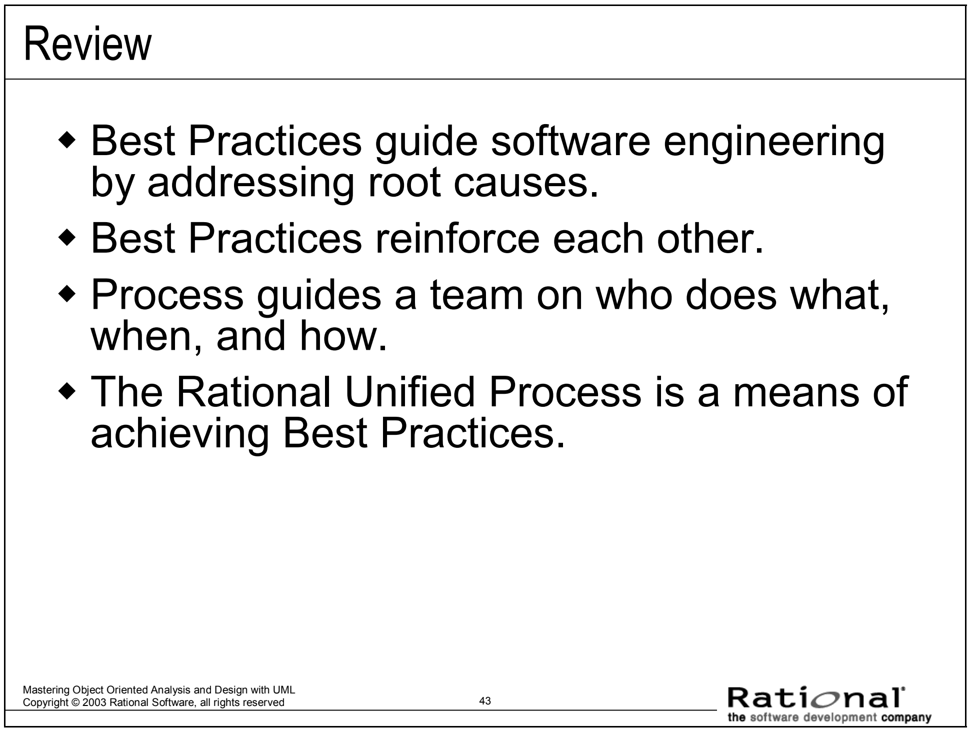
A **role** defines the behavior and responsibilities of an individual, or a set of individuals working together as a team.

An **activity** is the smallest piece of work that is relevant.

Artifacts: These are the modeling constructs and documents that activities evolve, maintain, or use as input.

- **Documents:** These record system requirements, including usability, reliability, performance, and supportability requirements.
- **Model:** This is a simplified view of a system. It shows the essentials of the system from a particular perspective and hides the non-essential details.
- **Model elements:** These help the team visualize, construct, and document the structure and behavior of the system, without getting lost in complexity.

Review



Review

- Best Practices guide software engineering by addressing root causes.
- Best Practices reinforce each other.
- Process guides a team on who does what, when, and how.
- The Rational Unified Process is a means of achieving Best Practices.

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved 43