Rational the e-development company™

WINDOWS

# Getting Started with Rational Suite

VERSION: 2001.03.00

PART NUMBER: 800-024085-00

support@rational.com
http://www.rational.com

Rational
the e-development company™

# Contents

# Preface

*Getting Started with Rational Suite* shows how software teams use Rational Suite to plan, design, implement, and test software applications. This book also points you to other information about Rational Suite, so that you can learn more on your own.

Rational Suite delivers a comprehensive set of integrated tools that embody software engineering best practices and span the entire software development life cycle. Rational Suite's unparalleled level of integration improves communication both within teams and across team boundaries, reducing development time and improving software quality.

## Audience

Read this book if you:

- Are a member of a software development team – an analyst, developer, test engineer, or manager
- Have experience with some aspect of Windows application development

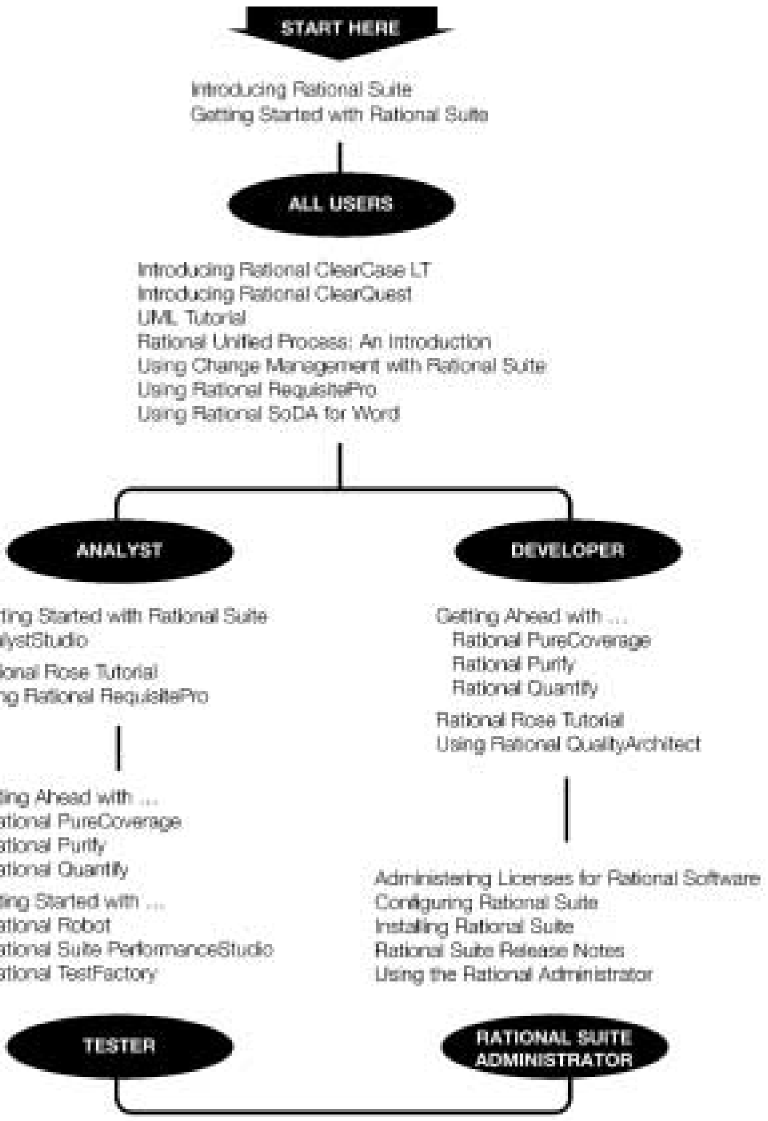
You do not need prior experience with any Rational tools before reading this book.

## Other Resources

- Online Help is available for Rational Suite.

  From a Suite tool, select an option from the **Help** menu.
- All manuals are available online, either in HTML or PDF format. The online manuals are on the Rational Solutions for Windows Online Documentation CD.
- To send feedback about documentation for Rational products, please send e-mail to `techpubs@rational.com`.
- For more information about Rational Software technical publications, see: http://www.rational.com/documentation.
- For more information on training opportunities, see the Rational University Web site: http://www.rational.com/university.

## Rational Suite Documentation Roadmap


START HERE
Introducing Rational Suite
Getting Started with Rational Suite
ALL USERS
Introducing Rational ClearCase LT
Introducing Rational ClearQuest
UML Tutorial
Rational Unified Process: An Introduction
Using Change Management with Rational Suite
Using Rational RequisitePro
Using Rational SoDA for Word
ANALYST
Getting Started with Rational Suite AnalystStudio
Rational Rose Tutorial
Using Rational RequisitePro
DEVELOPER
Getting Ahead with ...
Rational PureCoverage
Rational Purify
Rational Quantify
Rational Rose Tutorial
Using Rational QualityArchitect
Getting Ahead with ...
Rational PureCoverage
Rational Purify
Rational Quantify
Getting Started with ...
Rational Robot
Rational Suite PerformanceStudio
Rational TestFactory
Administering Licenses for Rational Software
Configuring Rational Suite
Installing Rational Suite
Rational Suite Release Notes
Using the Rational Administrator
TESTER
RATIONAL SUITE ADMINISTRATOR

## Contacting Rational Technical Support

If you have questions about installing, using, or maintaining this product, contact Rational Technical Support as follows:

| Your Location | Telephone | Facsimile | E-mail |
|---|---|---|---|
| North America | (800) 433-5444<br>(toll free)<br>(408) 863-4000<br>Cupertino, CA | (781) 676-2460<br>Lexington, MA | support@rational.com |
| Europe, Middle East, Africa | +31 (0) 20-4546-200<br>Netherlands | +31 (0) 20-4545-201<br>Netherlands | support@europe.rational.com |
| Asia Pacific | +61-2-9419-0111<br>Australia | +61-2-9419-0123<br>Australia | support@apac.rational.com |

**Note:** When you contact Rational Technical Support, please be prepared to supply the following information:

- Your name, telephone number, and company name
- Your computer's make and model
- Your operating system and version number
- Product release number and serial number
- Your case ID number (if you are following up on a previously-reported problem)

# Welcome to Rational Suite

# 1

Does your organization focus on developing and delivering software?

If so, think about your last project. Was it on time? Within its budget? Was communication among team members clear and timely? Did your team maintain consistency throughout the project as it defined requirements, developed designs, wrote code, and tested the application? Was your build process repeatable? Did your software meet requirements, satisfy users, and perform reliably?

Many project teams experience problems in these areas. In fact, many software projects finish late (or not at all), and the result often doesn't match the requirements. Many projects uncover serious design flaws late in the process. Defects are often found after the software ships, instead of during development.

How can you make your next project more successful?

## About this Book

*Getting Started with Rational Suite* helps you learn how the process and tools built into Rational Suite help software development team members accomplish their goals. Using the Rational Unified Process, you learn about developing requirements, creating visual models, code, and tests, and managing change. You learn how your team can use Rational's tools to automate the software engineering process.

This book guides you through an iteration of software development, describing typical ways to use and integrate Rational Suite so that your team can accomplish core work successfully.

Rational Suite will help you increase your productivity and effectiveness by:

- Managing change across the development lifecycle
- Improving collaboration using proven tools and process
- Building quality in software development from requirements to release

## Principles of Software Development

Rational Software Corporation, the e-development company, helps organizations develop and deploy systems and applications through a combination of software engineering best practices, integrated tools that automate these best practices, and professional services that accelerate the adoption and implementation of these best practices and tools. Rational's e-development solution helps organizations overcome many software development issues by accelerating time to market while improving quality.

Rational helps you increase your productivity and effectiveness by focusing on these software development best practices:

**Develop software iteratively.** Iterative development means analyzing, designing, and implementing incremental subsets of the system over the course of a project. The project team plans, develops, and tests one subset of system functionality per iteration. The team develops the next increment, integrates it with the first iteration, and so on. Each iteration results in either an internal or external release and moves you closer to the goal of delivering a product that meets its requirements.

Developing iteratively helps make your project more predictable, lets you collect feedback early, helps you identify and eliminate risks early in the project, and allows you to test continuously throughout the project lifecycle.

**Manage requirements.** A requirement is one criterion for your project's success. Your project's requirements answer questions such as "What do customers want?" and "What new features must we absolutely ship in the next version?" Most project teams work with requirements. On smaller, less formal projects, requirements might be kept in text files or e-mail messages. Other projects may use more formal ways of recording and maintaining requirements.

When you manage requirements, you can understand how changed requirements affect your project. You can effectively communicate requirements to all team members and to stakeholders. Effective requirements management helps your organization ensure that its products meet their stated goals.

**Use component-based architectures.** Software architecture is the fundamental framework on which you construct a software project. When you define an architecture, you design a system's structural elements and their behavior, and you decide how these elements fit into progressively larger subsystems.

A component is a non-trivial, independent, and replaceable part of a system that combines data and functions to fulfill a clear purpose. You can build components from scratch, reuse components you previously built, or even purchase components from other companies.

Designing a component-based architecture enables you to improve your project's predictability and helps enhance maintainability and extensibility.

**Model software visually.** Visual modeling helps you manage software complexity. At its simplest level, visual modeling means creating a graphical blueprint of your system's architecture. From this visual representation of your architecture, you can quickly detect problems such as inconsistencies and lack of modularity. With a visual model, you have a powerful and unambiguous communication mechanism that your whole team can use.

Visual models help you improve communications across your entire team. They help you detect inconsistencies among requirements, designs, and implementations. They also help you evaluate your system's architecture, ensuring sound design.

**Verify your software's quality.** Verifying software means testing what's been built against defined requirements. This includes testing that the system delivers required functionality and verifying reliability and performance.

An important benefit of iterative development is that you can begin testing early in the development process. Testing every iteration allows you to discover problems early in the development cycle and to expose inconsistencies among requirements, designs, and implementations.

**Manage change.** It is important to manage change in a trackable, repeatable, and predictable way. Change management includes facilitating parallel development; tracking and handling change requests; defining development processes so that they are repeatable; and reliably reproducing software builds.

When development teams are successful at managing change throughout a project's lifecycle, team members can consistently focus on completing their activities rather than, as examples, searching for the latest version of a requirements document or trying to figure out why a build failed during testing.

Managing changes to your project facilitates clear communication. It helps you control change propagation, define and repeat development processes, and control risk.

## Rational Suite Can Help

To put these principles to work, Rational Software offers Rational Suite, a family of market-leading software development tools supported by the Rational Unified Process. These tools facilitate work throughout a project's lifecycle. Rational packages the tools and the process into several Suite editions, each of which is tailored for specific roles in your development team – analysts, developers, testers, and project managers. Alone, these tools have helped organizations around the world successfully create software. Integrated, Rational tools and process:

- **Unify your team** by enhancing communication.
- **Optimize individual productivity** by providing market-leading development tools packaged in Suite editions that have been specifically designed for the major roles on your team.
- **Simplify adoption** by providing a comprehensive set of integrated tools that have simplified installation, licensing, and user support plans.

## What's in Rational Suite?

Rational Suite provides sets of tools tailored for every member of your team. Each Suite edition combines tools designed for a specific practitioner with the Rational Suite Team Unifying Platform. The following sections describe these tools and the Suite editions.

### Tools that Unify Your Team

#### Rational Suite Team Unifying Platform

Rational Suite Team Unifying Platform unifies all members of a software development team to maximize productivity and quality. This Suite edition provides best practices and integrated tools for managing change, building quality, and communicating results from requirements to release.

Rational Suite Team Unifying Platform is useful to project members who need access to common project artifacts, but do not need any of the optimized, role-specific tools found in the other Suite editions. For example, project and program managers, project administrators, and development managers use the tools in this Suite. Every Rational Suite edition includes the Team Unifying Platform.

The Team Unifying Platform Suite edition contains the following tools:

**Rational Unified Process.** An online collection of software best practices that guide your team through software development. Provides guidelines, templates, and Tool Mentors (instructions for applying the guidelines to specific Rational tools) for each phase of the development lifecycle.

**Rational RequisitePro.** Helps you organize, prioritize, track, and control changing project requirements.

**Rational ClearQuest.** Manages change activity associated with software development, including enhancement requests, defect reports, and documentation modifications.

**Rational ClearCase LT.** Provides software configuration management and a built-in process to track changes to all software project artifacts, including requirements, visual models, tests, and code.

**Note:** Rational ClearCase LT is not included inside any UNIX Suite.

**Rational SoDA.** Automatically generates project documents by extracting information from files you produce during project development, including source code and files produced by Rational tools. Formats the information according to predefined templates. SoDA is integrated with Microsoft Word for ease of use and easy customizing.

**Rational TestManager.** Helps you create real-world functionality and multi-user performance tests to test the performance and reliability of e-business, multi-tier, and database applications. TestManager tracks how many tests have been planned, scripted, and run; which requirements have been covered; and the number of tests that have passed and failed. TestManager allows your team to obtain an objective assessment of project status and create reports to communicate these findings to project stakeholders.

## Tools for Analysts

An analyst’s role is to:

- Represent the user’s needs to the development organization
- Determine *what* the system does
- Specify and manage requirements

### Rational Suite AnalystStudio

Rational Suite AnalystStudio is designed to support the analyst on your team. It contains the Rational Suite Team Unifying Platform (**Rational Unified Process**, **RequisitePro**, **ClearQuest**, **ClearCase LT, SoDA,** and **TestManager**) and:

**Rational Rose (Professional Data Modeler Edition).** Enables visual modeling of architectures, components, and data using the industry-standard Unified Modeling Language (UML). The UML is a language for specifying, visualizing, constructing, and documenting software systems.

## Tools for Developers

A developer's role is to:

- Determine *how* the system works
- Define architecture
- Create, modify, manage, and test code

Rational Suite provides two editions to support the Software Development function on your team: Rational Suite DevelopmentStudio and Rational Suite.

### Rational Suite DevelopmentStudio

Rational Suite DevelopmentStudio is the Rational Suite edition designed for system developers, designers, and architects. Rational Suite DevelopmentStudio contains the Team Unifying Platform (**Rational Unified Process**, **RequisitePro**, **ClearQuest**, **ClearCase LT, SoDA,** and **TestManager**) and:

**Rational Rose (Enterprise Edition).** Enables visual modeling of architectures, components, and data using the industry-standard Unified Modeling Language (UML). Automatically implements the framework of your code in Java, C++, Microsoft Visual Basic, and other popular languages. Because it is tightly integrated with Microsoft Visual Studio and IBM VisualAge for Java, Rose's round-trip engineering feature helps you automate the process of maintaining consistency between a model and its implementation.

Rational Rose also automates the mechanical aspects of test code creation by generating test code from visual models. This allows developers to automatically generate component tests and build stubs and drivers before an application is complete. This feature helps to reduce project risk because your team can test early and often, determining how a potential system architecture meets functional and performance requirements before developing the design further. Enterprise JavaBeans, COM, COM+, and DCOM models are supported in this feature.

**Rational Purify.** Pinpoints run-time errors and memory leaks in Visual C++ application code and memory usage in Java application code.

**Rational PureCoverage.** Identifies which parts of your Java, Visual C++, or Visual Basic program have and have not been exercised. Exposes testing gaps so you can prevent untested application code from reaching your users.

**Rational Quantify.** Profiles your Java, Visual C++, or Visual Basic application to help you identify performance bottlenecks in your code.

### Rational Suite DevelopmentStudio – RealTime Edition

Rational Suite DevelopmentStudio – RealTime Edition is the Rational Suite edition designed for practitioners who focus on real-time and embedded development. This Suite edition contains all the tools in Rational Suite DevelopmentStudio but replaces Rational Rose with Rational Rose RealTime.

**Rational Rose RealTime.** Delivers a powerful combination of notation, processes, and tools to meet the challenges of real-time development. Using Rose RealTime, you can:

- Create executable models, allowing you to compile and observe simulations of your UML designs on the host or target platform. The result is that you can refine your design early and continually verify quality.
- Generate complete, deployable executables in C or C++ directly from UML design models targeted to real-time operating systems. Generating these applications eliminates the need for manual translation and avoids costly design interpretation errors.

To learn more about Rose RealTime, see the online tutorials available from the Rose RealTime **Help** menu.

## Tools for Test Engineers

A test engineer’s role is to:

- Ensure that software meets all requirements
- Create, manage, and execute tests
- Report results and verify fixes

### Rational Suite TestStudio

Rational Suite TestStudio, the Rational Suite edition designed for test engineers, contains the Team Unifying Platform (**Rational Unified Process**, **RequisitePro**, **ClearQuest**, **ClearCase LT, SoDA,** and **TestManager**) and:

**Rational Robot.** Facilitates functional and performance testing by automating record and playback of test scripts. Allows you to write, organize, and run tests, and to capture and analyze the results.

**Rational TestFactory.** Automates testing by combining automatic test generation with source-code coverage analysis. Tests an entire application, including all GUI features and all lines of source code.

**Rational Purify.** Pinpoints run-time errors and memory leaks in Visual C++ application code and memory usage in Java application code.

**Rational PureCoverage.** Identifies which parts of your Java, Visual C++, or Visual Basic program have and have not been exercised. Exposes testing gaps so you can prevent untested application code from reaching your users.

**Rational Quantify.** Profiles your Java, Visual C++, or Visual Basic application to help you identify performance bottlenecks in your code.

### Rational Suite Enterprise

On some projects, team members may perform many types of tasks. For example, on smaller projects, team members might perform more than one role. On larger projects, team members might move from task to task. It may therefore make sense to equip each team member of an organization with a full complement of tools.

Rational Suite Enterprise contains all the tools in AnalystStudio, DevelopmentStudio, and TestStudio, so it can accommodate the needs of all members of your team.

## For More Information

For more information about Rational Suite and the principles of software development, see *Introducing Rational Suite*.

For more information about the Unified Modeling Language, visit the UML Resource Center at: http://www.rational.com/uml

This Web site contains UML information, tips about getting started with UML, and a bibliography for further reading.

## What’s Next

The next chapter introduces you to the Rational Unified Process, a software engineering process. By providing a disciplined approach to software development activities and responsibilities, based upon industry best practices, the Rational Unified Process helps you develop high-quality software that meets the needs of its end users within a predictable schedule and budget.

# Learning About the Rational Unified Process 2

This chapter introduces you to the Rational Unified Process. You familiarize yourself with the Rational Unified Process and read guidelines for the work you will perform in your projects.

## Audience

This chapter applies to all members of a software development team.

## What Is the Rational Unified Process?

The Rational Unified Process is a software engineering process that helps you:

- Produce high-quality software
- Meet the needs of your end users
- Work within a predictable schedule and budget

The Rational Unified Process serves as a personal and team-centered guide through best practices for controlled, iterative software development. Many organizations worldwide have successfully used it for both small- and large-scale development efforts.

The Rational Unified Process is implemented as an online guide and knowledge base, which you view with a Web browser. It is a customizable process framework that can be adapted to the way you work. The Rational Unified Process provides prescriptive guidelines, templates, and examples, which are tightly integrated with other Rational tools.

### The Rational Unified Process and Rational Suite

The Rational Unified Process can help you and your team work more effectively. We recommend that you adopt all or part of it to support your development efforts. You can, of course, successfully use Rational Suite without adopting any of the process. (You can also use Rational Unified Process with projects that do not use Rational Suite or its component tools.)

Even if you do not follow the Rational Unified Process, you can use it as a source of information about software engineering. For example, it contains topics to help you better understand Unified Modeling Language (UML) concepts, the industry-standard language for specifying, visualizing, constructing, and documenting software systems.

This book follows Rational Unified Process guidelines.

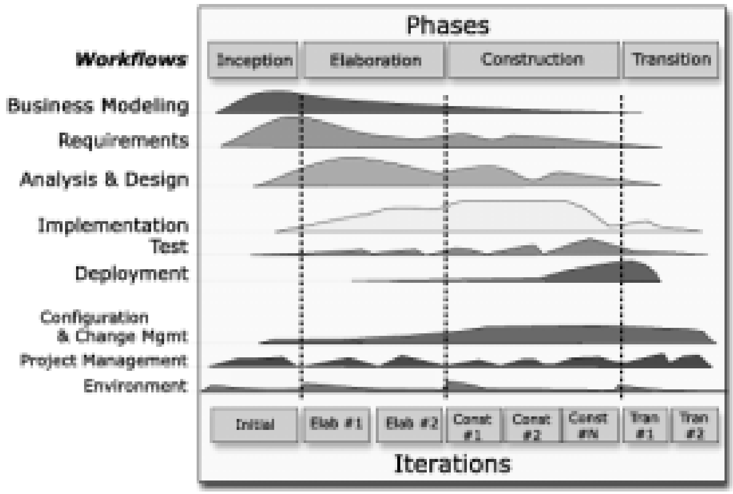
## Learning the Mechanics

The Rational Unified Process guides you through the full software development lifecycle: project management, business modeling, requirements management, analysis and design, implementation, testing, deployment, configuration management, and environment management.

## The Process at a Glance

Below is an overview of the Rational Unified Process. This diagram represents empirical data collected by Rational Software, showing that software development is best organized into *phases*, each of which is performed in a series of *iterations*. Throughout each phase, project team members from each of the software development disciplines (analysts, developers, testers) perform activities in one or more *workflows*. The diagram shows how emphasis on different workflows varies with each iteration.

**Figure 1 Rational Unified Process Overview**

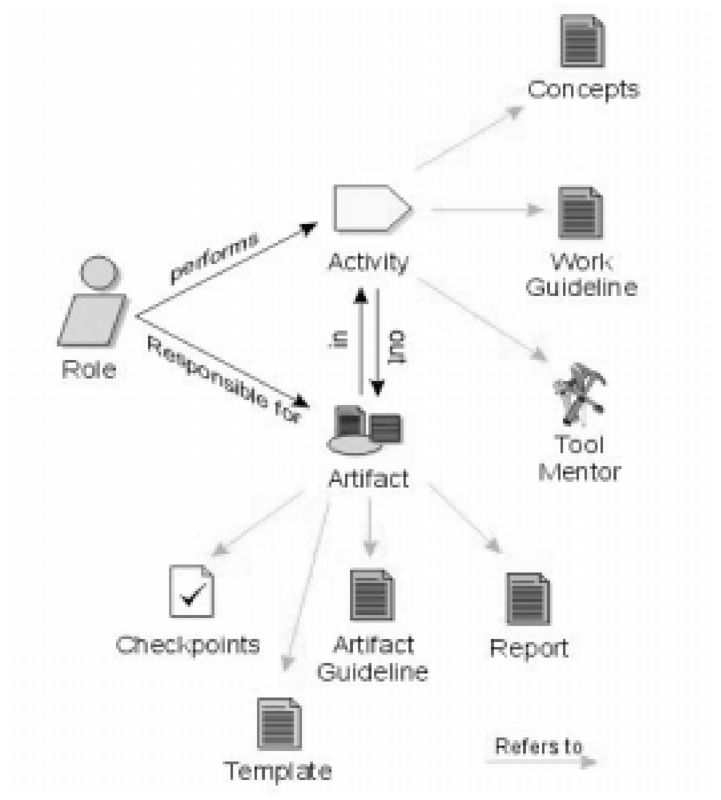


## Key Concepts

The Rational Unified Process defines the following important concepts:

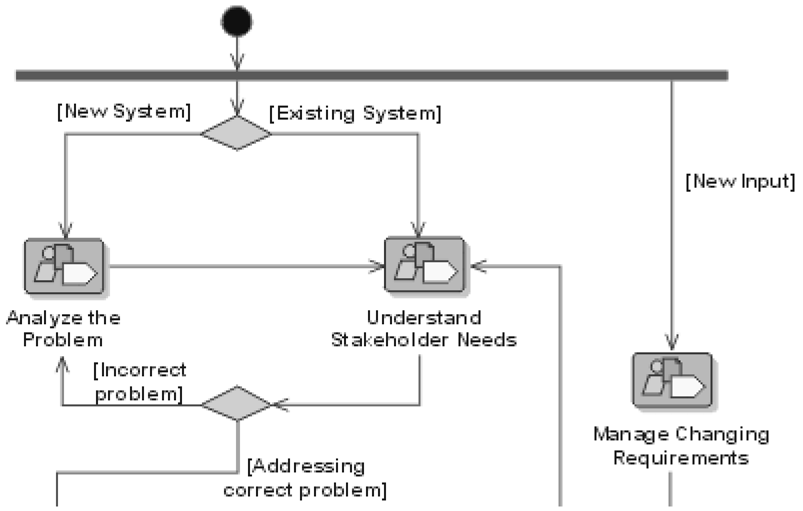
- A *role* is defined as the behavior and responsibilities of an individual or a group of individuals on a project team. One person may act in the capacity of several roles over the course of a project. Conversely, many people may act in the capacity of a single role in a project. Roles are responsible for creation of artifacts.
- An *artifact* is something a role produces as the result of performing an activity. In the Rational Unified Process, the artifacts produced in one activity are often used as input into other activities. An artifact can be small or large, simple or complex, formal or informal. Examples of artifacts are: a test plan, a vision document, a model of a system's architecture, a script that automates builds, or application code.
- An *activity* is a unit of work that is performed by a particular role. It is a set of ordered steps, like a recipe, for creating an artifact. See Figure 2 for examples.

**Figure 2 Relationship between a Role, an Activity, and Artifacts**



- A *workflow* is the sequence of activities performed by the roles to attain an observable value. The activity diagram in Figure 3 shows a portion of a Requirements workflow.

**Figure 3 Sample Requirements Workflow**


[New System]
[Existing System]
[New Input]
Analyze the Problem
Understand Stakeholder Needs
Manage Changing Requirements
[Incorrect problem]
[Addressing correct problem]

A workflow diagram serves as a high-level map for a set of related activities. The arrows between activities represent the typical flow of work between activities. You do not have to adhere to the prescribed path. You can jump to activities that are not connected by arrows. You can also repeat an activity whenever you need to (for example, if more information becomes available).

When using the Rational Unified Process, you can learn more about a workflow simply by clicking on it. The workflow detail appears and shows the roles involved, the artifacts used as inputs, the resulting artifacts, and the activities that make up this part of the workflow. For more information about any of these elements, just click in that area of the diagram.

## Starting with Actors and Use Cases

When you design or enhance a system, the Rational Unified Process recommends that members of your team start by agreeing on the system’s high-level behavior. To do so, you identify actors and use cases.

- *Actors* are the entities that interact with your system. An actor is often a person (for example, a sales clerk or administrator). An actor can also be an external hardware or software system (for example, a cash register or credit card verification system provided by a financial institution).

- *Use cases* describe how an actor uses and interacts with the system. More formally, a use case describes what services a system provides to a certain actor. You define a use case by describing a complete sequence of actions that yields observable results of value to an actor.

  Use cases are a key concept in the Rational Unified Process. They enhance communication across development teams so that you can solve the right problem and define the right system for your users.

  For more information on use cases, see *Managing Requirements* on page 17.

## Tool Mentors: Implementing the Process Using Rational Tools

The Rational Unified Process provides guidelines for all phases of software development. It uses Tool Mentors to provide guidance on using Rational tools. Tool Mentors give detailed descriptions of how to perform specific process activities or steps, or produce a particular artifact or report, using one or more tools.

## Extended Help

Extended Help is a powerful feature of Rational Suite that provides links to the Rational Unified Process and to other information. You use Extended Help directly from the tools you use to accomplish your work.

Extended Help provides information about the higher-level tasks you may want to accomplish. In addition, you can add your own organizational guidelines or standards to Extended Help.

You can learn more about Extended Help from any Rational tool. From the tool's menu, select **Help > Extended Help**. The Extended Help window appears showing a list of Rational Unified Process topics related to the tool you are using. From the Extended Help menu, select **Help > Help**. Then choose your topic of interest.

## Learning about Developing e-Business Solutions

The Rational Unified Process provides guidance for developing *e-business* solutions. E-business is about building systems that automate business processes. Organizations building e-business solutions use best practices (like visual modeling) to develop rapidly and in a controlled manner. The Rational Unified Process allows you to explore e-business in greater detail. Some concepts include Distribution Patterns, e-Business Development, User-Centered Designs, and Web Architecture Patterns.

## Summary

### For More Information

To learn more about the Rational Unified Process, start the Rational Unified Process by clicking the **Start** button. Then choose **Programs > Rational Suite > Rational Unified Process**. Read the topics on the Getting Started page. (To return to the Getting Started page, click **Getting Started** at the top of any Rational Unified Process page.)

### What You Learned in this Chapter

In this chapter, you learned:

- The Rational Unified Process helps you implement best practices for software development. In Rational Suite, use of the Rational Unified Process is recommended, but optional.
- A workflow describes a set of related activities focused on meeting a goal. For each activity, an actor uses artifacts created in previous activities and produces other artifacts.
- Early in a development project, you define actors (users and external systems that interact with your system) and use cases (services that the system provides to actors).
- Tool Mentors provide detailed instructions for performing a Unified Process activity using the appropriate Rational tool.

### What's Next

In the next chapter, you learn about working on and managing project requirements.

# Managing Requirements 3

In this chapter, you learn about how to create and manage use case requirements using Rational Rose and Rational RequisitePro.

## Audience

This chapter applies to analysts, project managers, system architects, and other team members responsible for system definition.

## Why Worry About Requirements?

One definition of project success is that the product you deliver meets its requirements. The formal definition of a requirement is “a condition or capability to which the system must conform.” More informally, a requirement describes a feature or behavior that a system must have.

### Where Do Requirements Come From?

As an analyst, your job starts with gathering the needs of everyone who has an interest in your project – your stakeholders. To gather needs, you interview users and other stakeholders, analyze enhancement requests, and work with project team members. You then determine which needs should become project requirements.

## Starting Requirements Work with Use Cases

Once you capture the needs of your project stakeholders and determine which needs to include in the project, you develop use cases to communicate the proposed behaviors of the system.

## Why Work with Use Cases?

Use cases describe system behavior from a user's perspective, using a common language that everyone on the team can understand. Working with use cases is a key unifying mechanism in the Rational Unified Process.

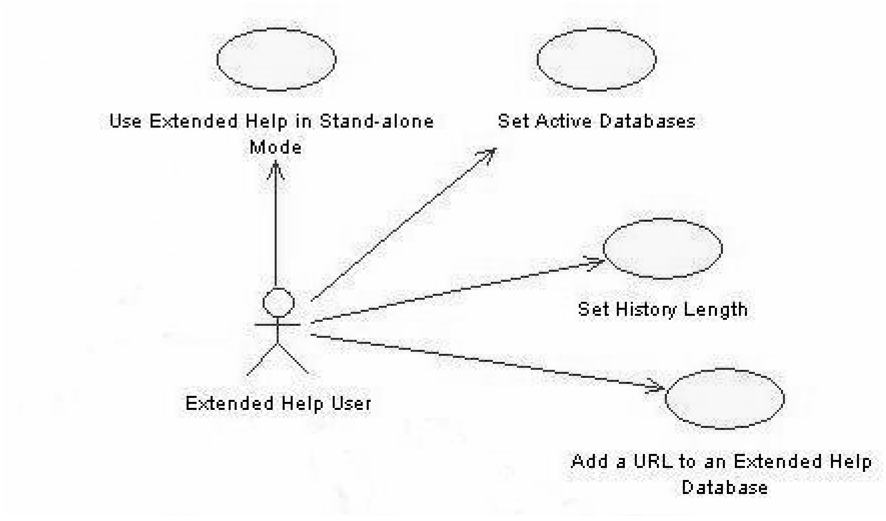
Use cases are important to everyone on the project:

- *Analysts* use them to express how the system should behave.
- System users use them to validate system behavior starting as early as the design phase.
- *Developers* and *designers* can start with human-language and graphical use cases. They elaborate them first into architectural specifications and then into classes.
- *Testers* develop test designs based on use cases.

Once you create a use case, you want to add it to a *use case diagram* (see Figure 4 for an example). A use case diagram shows:

- The behaviors of a system. The use cases describe what the system does.
- The boundaries of a system. The actors represent external entities that interact with the system.
- The relationships among use cases and actors.

**Figure 4 Sample Use Case Diagram**


Use Extended Help in Stand-alone Mode
Set Active Databases
Set History Length
Extended Help User
Add a URL to an Extended Help Database

### Creating Use Case Diagrams with Rational Rose

**Rational Rose** helps analysts visualize the behavior of a system through use case and other diagrams. These diagrams help you manage complexity because they allow you to see the "big picture." Rational Rose helps you create use case diagrams so all stakeholders share a common understanding of the project's goals and expected deliverables.

Using Rose is also an effective way to continuously communicate the impact of change throughout the development lifecycle. All team members can easily share and revise use case diagrams because they are written in the Unified Modeling Language (UML) - an easily understood, industry-standard language for designing software. For example, analysts use Rose to create use case diagrams to describe a system at a high level, and architects continue this work by using Rose to design the system in more detail. Therefore, your system diagram, architecture, and data are managed by one tool, Rational Rose, and with one language, UML.

### Learning More About Use Cases

Developing use cases is a typical way of starting requirements work. The Rational Unified Process describes how to write a use case. It includes details about the artifacts that you will need to get started and the artifacts that result from the activity. It then provides a step-by-step description of the activity, and offers a template for creating use cases.

This template provides guidelines about how to structure a use case. It is the framework for you to define your *use case requirements*, software requirements that describe the system's behavior. We recommend that you use this template, or another template designed by your group, to ensure consistency and completeness in use case development. This makes it easy for all stakeholders to locate and understand important project information.

## Managing Requirements

*Managing requirements* is a systematic approach to:

- Finding, documenting, organizing, and tracking requirements.
- Establishing and maintaining agreement between the customer and the project team on the system's changing requirements.

Requirements management is challenging, in part because requirements change throughout a project. For example, users may change their minds about essential features, or they may not have articulated their wishes clearly in the first place. Competitors may release new versions of their software and you must respond by changing project plans midstream. Even changing laws may affect your software.

## Using RequisitePro to Manage Requirements

RequisitePro makes it easy to develop and manage requirements. Each RequisitePro project includes a requirements database and its related documents. RequisitePro is integrated with Microsoft Word and is packaged with Word templates so that you can get started quickly.

RequisitePro is designed to work for your entire team:

- *Analysts* use RequisitePro to document and maintain requirements.
- *Developers* use requirements to design an architecture and write more detailed specifications.
- *Test engineers* use requirements to design tests and check test coverage.
- *Project leaders* and *managers* use RequisitePro to define project work based on available resources (for example, time, budget, and personnel).

## How Does RequisitePro Work?

RequisitePro is both document-centric and database-centric and integrates the strengths of both:

- The *document* features provide a familiar environment (Microsoft Word) for creating descriptions and communicating your work to project stakeholders. You can start a requirements document either by importing existing Word files into RequisitePro or by using the RequisitePro Word Workplace.
- The *database* features help you organize your requirements, prioritize your work, track requirements changes, and share information with other Rational tools. To work with database features, you use the RequisitePro Views Workplace.

RequisitePro helps you to identify and elaborate on your project's requirements. RequisitePro allows you to document your use cases as requirements, dynamically registering them in a database for powerful sort and query capabilities. These features help you track the requirement as it changes (and requirements always change).

RequisitePro also allows you to *link* requirements to each other, establishing relationships between requirements. Through these links, RequisitePro helps you track how changes to the system affect your requirements and how changes to requirements affect your system.

## Developing Requirements with Use Cases

You are ready to begin defining requirements and developing use cases for your system once you gather the needs of everyone who has an interest in your project. We recommend that you develop use cases in Rose and then detail and manage them using RequisitePro.

Here's a typical way to use Rational Rose and RequisitePro to develop requirements by capitalizing on the power of uses cases:

1. [one time] An *administrator* defines and sets up the RequisitePro and Rose environments. The administrator creates RequisitePro projects, determines their structure, and establishes security permissions. Each project has its own database, where requirements can be added, modified, or deleted. Information extracted from requirements documents is stored in the database.
2. Gather stakeholders needs and determine which proposed features will be implemented in your system. Some of these features will help you identify your use cases.
3. Create the new use cases using Rational Rose. If you are developing an original system, you will need to create a use case diagram, then add use cases to it. If you are enhancing an existing system, you simply add the new use cases to the existing use case diagram.
4. Use the integration between Rose and RequisitePro to associate the Rose model with a RequisitePro project. The use case is now represented both in words and by a visual model.
5. In RequisitePro, create a new use case requirement. Use RequisitePro's Word Workplace to define project requirements, and the Views Workplace to analyze them.
6. Verify that the use case requirement was added. Using a RequisitePro Views Workplace, view requirements related to the use case you are working on.
7. [optional] Revise attributes of your requirements. Using RequisitePro, you can review each requirement's attributes and their properties. This allows you to modify information related to your requirement, including attribute values, traceability relationships to other requirements, and descriptions.
8. [optional] Link the use case requirement to another type of requirement (for example, a *feature* requirement; a system's feature requirements are written at a very high level and form a foundation for the entire system). Links describe dependencies and relationships between requirements.

9 [optional] Display a traceability matrix in the RequisitePro Views Workplace to show which links which have changed. If a change occurs in one requirement, existing links with that requirement become *suspect*. An analyst needs to examine the changes, and decide whether to edit the dependent requirements before clearing the suspect link.

## Other Requirement Types

So far, we've discussed use case requirements and high-level feature requirements. Some requirements do not lend themselves to use cases, RequisitePro supports other types of requirements also. Different requirements types help separate the various levels of abstraction and purposes of your project's requirements. You can also define new requirement types. For example, you can define requirements for performance targets and platform support. RequisitePro can manage any type of requirement that you need on your project.

## When Have You Finished Gathering Requirements?

Requirements emerge from a series of communications between analysts and project stakeholders (application users, members of the marketing team, funders, and so on). As you capture requirements, you check your work with the appropriate stakeholders. When the stakeholders and your team come to agreement, your initial job of gathering requirements is done.

Of course, as the project progresses, you will continue to manage the requirements, adding some, possibly removing others, and responding to changes.

### Planning and Scheduling

RequisitePro is integrated with Microsoft Project so that you can link requirements with tasks on your project schedule. For more information:

1 On the RequisitePro Tool Palette, choose **Help > Contents and Index**.

2 In the RequisitePro Help Browser, on the Contents tab, navigate (by double-clicking) to **Wizards, Integrations and Components > RequisitePro Wizards > MS Project Integration Wizard**.

A Help topic appears, describing how to work with RequisitePro and Microsoft Project.

## Summary

### For More Information

For more information on getting started with RequisitePro, start Rational RequisitePro and explore Let's Go RequisitePro: from the RequisitePro Tool Palette menu, choose **Help > Let's Go RequisitePro**.

For more information about Rose, see *Modeling an Enhancement* on page 39.

### What You Learned in this Chapter

In this chapter, you learned:

- A *requirement* is a condition or capability to which the system must conform. RequisitePro manages your requirements and supports multiple requirement types.
- Managing requirements is a systematic approach to finding, documenting, organizing, and tracking system features and attributes.
- RequisitePro is both document-centric and database-centric, allowing your team to benefit from the strengths of both.
- To create a use case, you work in Rose to incorporate the use case in your visual model, then work in RequisitePro to add textual descriptions, attributes, and links.
- You have finished writing the first set of requirements when project stakeholders and your team agree that you're done.

### What's Next

In the next chapter, you learn how to start with use cases to do test planning, so that you can begin testing early in the development cycle.

# Test Planning 4

So far, you have learned how to define requirements for a new feature and develop a use case diagram. Now you are ready to learn about test planning.

## Audience

This chapter applies to test engineers, quality assurance managers, and other team members responsible for system testing.

## What Is Test Planning?

Test planning allows you and your group to effectively measure and manage test efforts over the course of the project. During test planning, you identify the types of tests you will perform, the strategies for implementing and running those tests, and the resources you will need during testing.

Test planning starts early in the development cycle, as soon as you understand the initial set of requirements. Artifacts such as use case requirements, project schedules, and visual models can be used as *test inputs* to help you determine what needs to be tested. You create *test cases* with information contained within these inputs, and use these throughout the test planning process as a “checklist” against which you determine the acceptance criteria of your tests. You also use test inputs to define *test configurations*, the sequence of attributes for potential organizational structures of the system that you will apply to your test cases.

As with development, test planning is an iterative process. You continue to plan testing throughout a project’s lifecycle, as analysts change or elaborate on requirements, and as developers design and implement code.

## Managing Risk

The recommended strategy for test planning is to focus on the riskiest areas of the project first. For example, you can identify risks by considering:

- The consequences of not testing a specific part of an application
- The consequences if a particular part of the application does not work properly
- The likelihood that an error will be discovered after the product ships
- The ramifications if a user, rather than a project member, discovers an error in the application

## Making a Plan and Measuring Progress

**Rational TestManager** allows you to use all types of project artifacts to plan, design, and run tests. You can use requirements, visual models, and source code to create a test plan so that all aspects of your system can be tested, including product features, system architecture, and code. With TestManager, artifacts used in your test plan are used to create a specific test asset. This ensures that any change made to the original artifact automatically prompts you to re-evaluate the related test asset. If appropriate, you can then update the test asset. The integration between Rational tools that enables this verifiable relationship between project assets helps you start testing for quality early in the development lifecycle.

Rational TestManager also:

- Provides access to all test-related information and artifacts so your team can easily assess project status
- Enables team members to share testing progress
- Helps you track how many tests have been planned, scripted, and run
- Shows which requirements have been covered by tests, and the number of tests that have passed and failed

Because TestManager is part of each Suite edition, team members can use it to evaluate how well they are meeting project requirements, to monitor the project's overall status, and to more effectively share and discuss information about testing activities with other project stakeholders.

## Developing a Test Plan

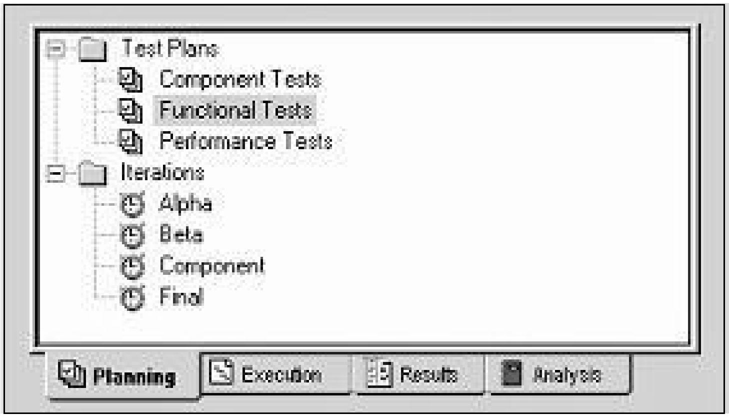
In TestManager, a *test plan* contains information about the purpose and goals of testing within a Rational project, and the strategies to implement and execute testing. You can have one or more test plans in a project. TestManager provides you with an empty test plan that you can use to start your planning. This test plan includes properties such as the name of the test plan, the configurations associated with the test plan, and the iterations associated with the test plan (when a test plan must pass).

You can run reports based on a test plan's properties. For example, you can run reports to determine which iterations are associated with a test plan. These reports can give you valuable information about the state of your testing project.

### Organizing Your Test Plan

As we mentioned earlier, you can have one or more test plans in a project. Each test plan, contains test cases. The test cases are organized hierarchically in *test case folders*. Both the test plan and test case folders can be placed in any order that makes sense for your organization. For example, you can create a test plan for each testing type (see Figure 5). And you can have a test case folder for each tester in your department, or for each phase of testing.

**Figure 5 Sample Test Plan Hierarchy**



Test case folders have properties such as the name of the test case folder, and the configurations and iterations associated with the test case folder. Any iterations and configurations associated with a test case folder are automatically associated with any test cases that are placed within it. TestManager allows you to change the test case's associations after you first establish them.

## Determining What to Test

You continue test planning by identifying test cases for your application. Each test case describes a specific area of the application to test. Each area can encompass a broad class of situations that you need to test. For example, in testing a cash sales transaction, you would probably test:

- Valid input (the customer pays the exact price; the customer pays more and needs change)
- Invalid input (the customer pays less than the sales price; the sales clerk enters an invalid part number)

So how do you determine what to test? This part of test planning – test analysis and design – often requires you to rely on your own human intelligence and experience, using existing project assets as a reference.

When you design tests, the first step is to understand how the system is supposed to behave. During analysis, you identify the conditions you need to test to verify that:

- The application does what *is* intended.
- The application does *not* do what is *not* intended.

## Working with Test Cases

A test case describes the extent to which you will test an area of the application. It lists the preconditions for performing a test, the input to provide during testing, the variables you will examine, and the expected results of each test. Existing project artifacts, such as requirements, provide information about the application and can be used as your test inputs. TestManager provides built-in test input types, but almost any artifact can be used as a test input.

For example, here's what the following artifacts offer as test inputs:

- Requirements describe a condition or capability to which a system must conform
- Visual models provide a graphic representation of a system's structure and interrelationships

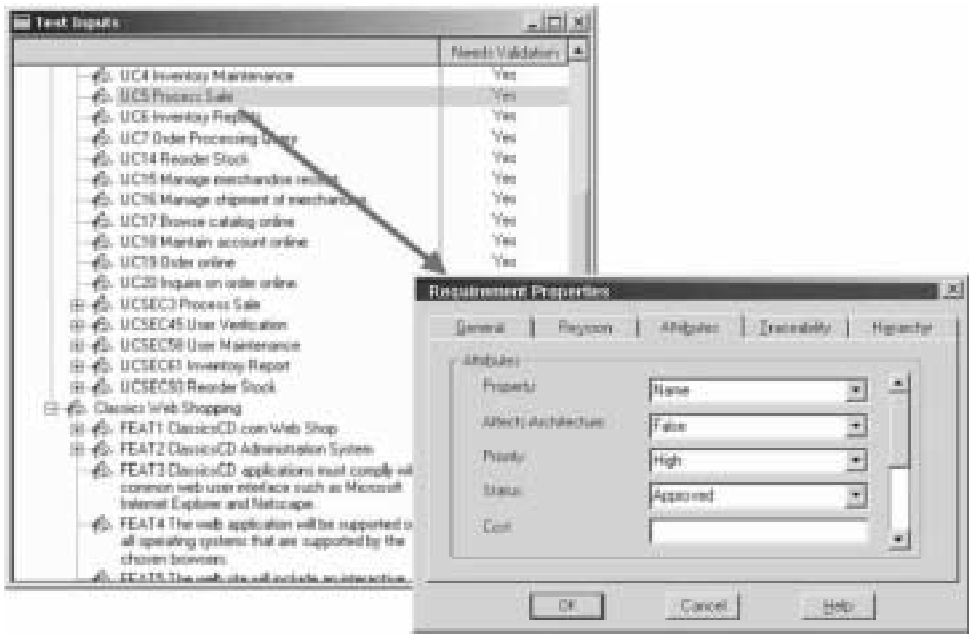
You can also define custom test input types, such as source code, software builds, and functional specifications.

Once you identify your test inputs, you can create test cases and associate them with test inputs. These associations allow you to react when changes to test inputs occur. These changes might require you to change the test cases or their implementations.

## Test Inputs from Rational RequisitePro

You can use RequisitePro requirements as test inputs. You or an administrator can use the Rational Administrator to associate a RequisitePro project with a Rational project. This association causes requirements to appear in the Test Input window after you log on to that project. You can then create a link between a requirement and a test case. The requirements, themselves, are created and managed in RequisitePro, but you can modify the properties of the requirements in TestManager (see Figure 6 for an example).

**Figure 6 Sample Test Inputs from RequisitePro and Related Properties**



| | Needs Validation |
|---|---|
| UC4 Inventory Maintenance | Yes |
| UC5 Process Sale | Yes |
| UC6 Inventory Report | Yes |
| UC7 Order Processing Query | Yes |
| UC14 Reorder Stock | Yes |
| UC15 Manage merchandise receipt | Yes |
| UC16 Manage shipment of merchandise | Yes |
| UC17 Browse catalog online | Yes |
| UC18 Maintain account online | Yes |
| UC19 Order online | Yes |
| UC20 Inquire on order online | |
| UCSEC3 Process Sale | |
| UCSEC45 User Verification | |
| UCSEC58 User Maintenance | |
| UCSEC61 Inventory Report | |
| UCSEC93 Reorder Stock | |
| Classics Web Shopping | |
| FEAT1 ClassicsCD.com Web Shop | |
| FEAT2 ClassicsCD Administration System | |
| FEAT3 ClassicsCD applications must comply wi[illegible] common web user interface such as Microsoft Internet Explorer and Netscape | |
| FEAT4 The web application will be supported o[illegible] all operating systems that are supported by the chosen browsers | |
| FEAT5 The web site will include an interactive | |

| | |
|---|---|
| Property | Name |
| Affects Architecture | False |
| Priority | High |
| Status | Approved |
| Cost | |

## Test Inputs from Rational Rose

If you use Rational Rose, then you can use Rose model elements as test inputs. You use the Rational Administrator to associate a Rose model with a project. You can then look at each individual model element in the Test Input window, and create an association between a model element and a test case.

### Elaborating on Test Cases

As part of developing your test plan, you need to design your tests. Test designs are elaborations of test cases. They provide the detail needed for understanding how the test case will be implemented. Design work is typically done when you plan your test cases.

Using TestManager, you design tests using the Design Editor. During this step, you capture specific steps or flows in a test case and add validation criteria or verification points. Later on, you learn how these designs help you implement your tests.

## Continuing with Test Planning

Writing a test plan is an iterative process that starts early in the project. It continues as analysts change requirements and elaborate on use cases, as developers design and write code, and as testers revisit requirements and use cases, discovering more areas or conditions to test. Test planning occurs in parallel with other development efforts, including testing.

As you work on your own test plan, we suggest you consider at least the following topics, described in the remainder of this section:

- Risks and resources
- Types of tests to perform
- Stages of Testing
- Scheduling

### Risks and Resources

Identifying risk is an important part of test planning. Once you identify the available testing resources, you need to balance inevitable resource constraints with the project and testing risks. As a result, you can refine the testing strategy.

We recommend that you prioritize tests as follows:

- **Must test (high)** – You must run this test to avoid severe risk or to identify weak project areas early in the development cycle. You cannot complete project testing without completing this test.

- **Should test (medium)** – You should schedule this test, but in a resource crunch, could consider not running it.

- **Could test (low)** – This test might be useful to run, but is not essential to the project. Run this test if you cannot make further progress on other, more important, tests.

- **Won’t test (low)** – This test is not part of the testing project. A test with this priority defines the boundaries of the test plan and helps focus attention on what will be tested.

## Types of Tests

There are many types of tests to consider as you create a test plan, including, but not limited to:

- **Functional tests –** Does the application meet its functional requirements? Use Rational Robot and Rational Rose for functional testing.
- **Reliability tests –** Can the application run without errors? Use Rational TestFactory, Purify, Quantify, and PureCoverage for reliability testing.
- **Performance tests** – Is the application’s performance acceptable under varying loads? Use Rational Suite TestStudio with Rational Rose for performance testing.

## Stages of Testing

There are several stages of testing to consider as you create a test plan. These stages progress from testing small components to testing completed systems and usually apply to different stages of the system’s development cycle:

- **Unit testing –** Verifies individual components, the smallest testable elements of the software.
- **Integration testing –** Ensures that the components in the implementation model operate properly when combined to execute a use case.
- **System testing** – Ensures that the software is functioning as a whole.
- **Acceptance testing** – Verifies that the software is ready and that it meets its requirements.

Unit testing is typically performed by software developers. As a test engineer, your focus is primarily on integration, system, and acceptance testing.

## Scheduling

Part of creating a test plan involves writing a schedule. You work with team leaders from other areas of the project to understand when their contributions will be ready for testing. You then need to balance your original schedule against the risks and resources you identified in order to arrive at the most effective schedule for testing. Each testing iteration presents an opportunity to validate one or more of your test cases. Developing a testing schedule based on iterations helps you filter your test cases so that you can more effectively design, implement, and run your tests for each stage of software development.

If you prioritized your tests as described in *Risks and Resources* on page 30, make sure you schedule at least the “must” (high priority) and “should” (medium priority) tests. If resources become more constrained over the course of the project, you can sacrifice tests of lower priority without compromising the absolute quality objectives expressed by the “must” tests.

## Summary

### For More Information

For more information about test planning:

- Read about test plans in the Rational Unified Process.
- Read about test plans in *Getting Started with Rational Robot*.
- For a more in-depth treatment of test planning, read *Testing Computer Software* (Vnr Computer Library) by Cem Kaner and others (ISBN: 1850328471).

### What You Learned in this Chapter

In this chapter, you learned:

- You can start test planning early in the project, after initial requirements are identified.
- Test planning is an iterative process, encompassing project and testing risks, evolving product requirements, available resources, and project schedule.
- Part of test planning involves creating test cases and relating them to test inputs.
- Analysis and design are important components of writing effective tests.
- Prioritizing tests helps you focus your testing effort on the riskiest and most important areas of the application to test.

### What’s Next

In the next chapter, you learn how Rational ClearCase LT helps you effectively manage change throughout the software development lifecycle.

# Managing Changing Artifacts 5

In this chapter, you learn about Rational's approach to managing change in software development using ClearCase LT and Unified Change Management (UCM). Together, UCM and ClearCase LT help you successfully manage changing project artifacts from requirements to releases, iteration through iteration.

## Audience

This chapter applies to all team members.

## What is Unified Change Management?

Unified Change Management (UCM) is Rational Software's best practices process for managing change in software system development from requirements to release. UCM focuses on these guiding concepts:

- An *activity* represents a unit of work you want to complete. Activities can be derived from a variety of sources, including a defect or an enhancement request.
- An *artifact* is an item that is produced, modified, or used in the software development lifecycle. Conceptually, artifacts can be requirements, tests, visual models, source code, or project plans. Artifacts are items that are critical to the success of your project and should be placed under *configuration management*, or version control. Usually, an artifact is represented by a file or a set of files.

UCM's key strength is that it unifies the activities used to plan and track software development progress with the artifacts used to create, design, and build software applications.

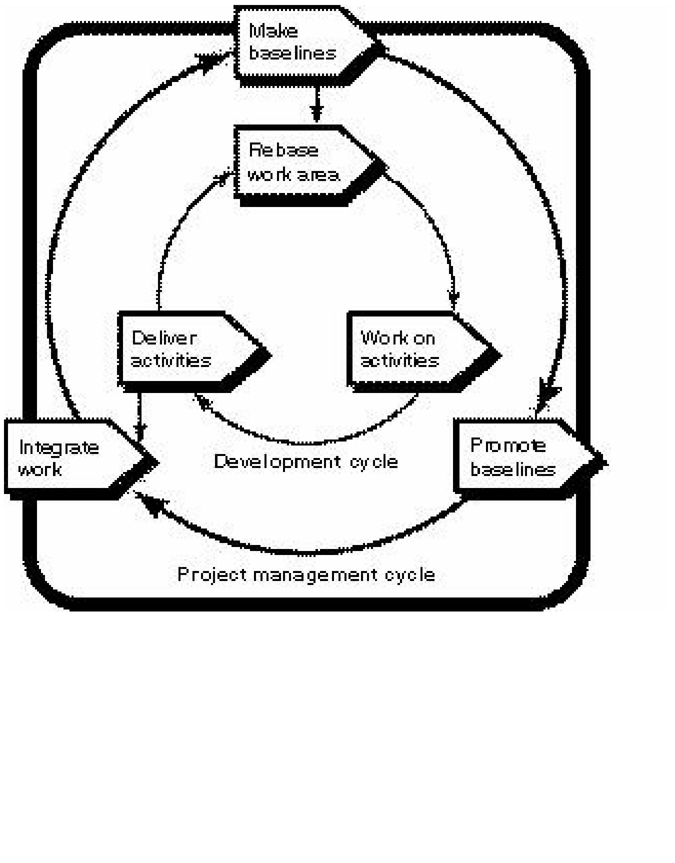
## UCM Tools

**Rational ClearQuest.** Manages change activity associated with software development, including enhancement requests, defect reports, and documentation modifications. ClearQuest also offers charting and reporting features to track and communicate project progress to all stakeholders.

**Rational ClearCase LT.** Uses UCM's built-in development process to track and manage changes to all software project files, including requirements, visual models, tests, and source code, among others.

Rational ClearQuest and Rational ClearCase LT provide tool support for Unified Change Management. Used together, these Rational tools help your software team better manage changing requirements and development complexity throughout the software development lifecycle. Figure 7 shows a typical way to manage change using Unified Change Management.

**Figure 7 Typical UCM Workflow**



## ClearCase LT and UCM - Using the Tool with the Process

The Rational Unified Process describes the process of working with UCM. You typically use UCM with ClearCase LT as follows:

1. [one time] A project manager or administrator installs Rational software. This individual sets up the ClearCase LT environment and creates a Rational project. (A *Rational project* associates the data created and maintained by Rational tools and enables integrations among them.)
2. [one time] You identify yourself to the project by *joining the project*. As a result, a private workspace (consisting of a *development stream* and a *development view*) is created for you. You also gain access to a workspace available to your entire team. This public workspace consists of an *integration stream*; you can create a companion *integration view* for your own use.
3. Your project manager uses ClearQuest to assign activities to you.
4. You run a ClearQuest query to find the activities assigned to you. This is your to-do list. From this list, you decide which activity to work on.
5. You work with artifacts as usual. Using ClearCase LT, you work in your private development workspace to:
   - Check out artifacts. When you check out an artifact, ClearCase LT asks which activity you want to work on. In the background, ClearCase LT keeps track of the *change set* (the list of changed artifacts) associated with the activity.
   - Edit and verify the changes.
   - Check the artifacts back in. When you check in an artifact, it is still part of your private workspace. Your change does not become publicly available until you deliver it, as described in Step 6.
6. When you have finished work on the activity, you *deliver* changes for an entire activity. Because ClearCase LT keeps track of the change set, you don’t have to specify the list of artifacts to deliver. Delivering the changes makes the changes publicly available through the integration stream. It may also close the activity, depending on the policies you set up.
7. After developers have delivered a set of activities, the project manager creates a *baseline*, a new common starting place for all developers that includes the new activities or modified artifacts. On your project, a new baseline may be created on a regular basis, perhaps even daily.

8 If the changes in the baseline are approved (through testing or through another review process), your project manager *promotes* it, making it the recommended baseline.

9 You *rebase* your development stream so that when you work on your next activity, you're starting from the most recent recommended baseline. Restart with Step 4 to select the next activity to work on.

Depending on the structure of your organization, you may identify these roles using different names, assign the responsibility of different roles to be performed by one individual rather than several, or share the responsibility of a single role among several team members, as defined by the Unified Process for UCM. No matter how your team is structured, you can use UCM successfully because the model follows a basic process for configuration and change management, and ClearCase LT helps to automate much of the artifact and activity auditing.

## Summary

### For More Information

For more information about Rational ClearCase LT, read *Introducing Rational ClearCase LT*.

For more information about UCM, read Using *Unified Change Management with Rational Suite*.

### What You Learned in this Chapter

In this chapter, you learned:

- Unified Change Management (UCM) helps software teams manage change in software development, from requirements to release.
- ClearCase LT and ClearQuest are the foundations for UCM. ClearCase LT manages the artifacts associated with a software development project. ClearQuest manages the project's activities and offers charting and reporting features to track and communicate project progress to all stakeholders.
- Team members use ClearCase LT to access artifacts under source control, they work on activities in their personal development workspaces, and deliver modified artifacts to the integration stream once they complete an activity.

## What’s Next

Now you understand how ClearCase LT and UCM help you manage changing artifacts throughout the development lifecycle. In the next chapter, we discuss how to use these tools and Rational Rose to update an architectural model for a new feature.

# Modeling an Enhancement 6

So far, you have learned about managing requirements, test planning, and change management for software artifacts. In this chapter, you learn how to incorporate designs for new features into existing visual models using Rational Rose. You also learn about delivering modified artifacts to a project's integration stream.

## Audience

This chapter applies to software designers, developers, and other team members responsible for carrying out a software development function.

## What Is Visual Modeling?

Visual modeling is the creation of graphical representations of your system's structure and interrelationships. The result is a blueprint of your system's architecture. A visual model:

- Is graphical, rather than text-based, making it easy to understand complex systems at a glance.
- Allows you to see relationships among design components, so that you can create cleaner designs and therefore write more maintainable code.
- Helps you meet customer needs because you base the visual model on project requirements.
- Improves communication across your team because you use a standard graphical language for conveying the system's architecture.

## Working with a Sequence Diagram

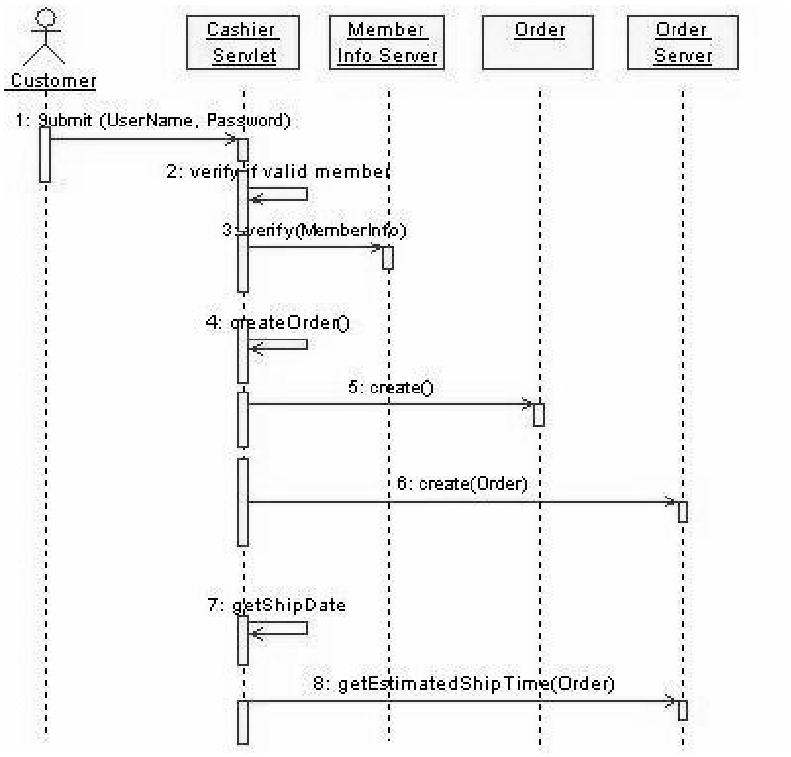
A *sequence diagram* is a visual representation of the steps through one path in a use case (see Figure 8 on page 40). Project members and other stakeholders can use a sequence diagram (graphical representation), use case requirements (text description), or both to evaluate the project direction and as a basis for their work.

A use case often contains more than one path. It always contains a basic flow which describes the most common path through the use case. It may contain alternative flows which describe other paths, including error conditions.

A sequence diagram shows how actors interact with the system, and in what order. When you first work on a sequence diagram, you tend to use human-language labels. As you refine the system design, you change the diagram so that it identifies:

- **Classes –** Sets of objects that share a common structure and common behaviors
- **Messages –** Interactions between classes

**Figure 8 Sample Rose Sequence Diagram**



The sample Rose sequence diagram in Figure 8 shows how the *actors* and other *objects* in the application communicate with each other. A *message icon*, a horizontal, solid arrow between two vertical, dashed *lifelines*, represents each communication. Items in a sequence diagram are arranged in chronological order.

### Updating a Sequence Diagram

Here's a typical way to update a sequence diagram for an existing system.

1. Use ClearCase LT to check-out the appropriate model and sub-model files.
2. Open the selected sequence diagram using Rational Rose.
3. Add messages to the sequence diagram for the new feature or enhancement you are working on.
4. Save the model and subunits (if prompted to do so).

## Continuing Work with a Sequence Diagram

There are a few additional tasks to perform once you make changes to a visual model. An overview of these steps is below.

### Publishing a Model to the Web

When you finish working on a visual model, we recommend that you create a Web version of the modifications so that people on your team who have not installed Rose can review the model and give you feedback.

**Rose Web Publisher** allows you to create a web-based version of a Rose model that others can view using a standard internet browser. A Rose Web Publisher recreates Rose model elements, including diagrams, classes, relationships, attributes, and operations. Once published, hypertext links enable you to traverse the model much as you would in Rose.

You can control what Rose Web Publisher includes by setting a variety of options. For example, you can select which views of a model are published, the amount of detail to include, the notation to use, and the graphics format for Rose diagrams. The Preview feature lets you launch your default browser and view the published model directly from Rose Web Publisher.

There are two ways to publish a model:

- By using Rose's dialog-based add-in available from **Tools > Web Publisher.**
- By using a command line batch interface that enables you to publish models using a batch processor.

## Refining Objects

Using a sequence diagram, you identify the objects involved with a use case. You next identify the classes to which the objects belong. Use Rose class diagrams to group related classes and to elaborate on them.

Once you identify classes, you revise the sequence diagram to use class and operation names instead of the human-language names you originally assigned.

## Implementing Code

You are now ready to implement code. From the diagrams you've created, Rose Enterprise Edition can create a code skeleton that is consistent with the models you've developed. This is called *forward engineering*. Starting from the generated code, you as a developer fill in the details of the algorithm.

To generate new code or to update existing code, choose an item from the Rose **Tools** menu. For example, to implement Java code for the enhancement you've been working on, you would choose **Tools > Java > Generate Java**.

When you start changing code, your model may become out of date. It is tedious to manually update the model whenever the code changes. It is also undesirable to create a model that immediately becomes obsolete. Rose helps you keep the code and the model consistent. For example, from the Rose menu, you would choose **Tools > Java > Reverse Engineer Java**. This is called *reverse engineering*.

As you can see from the **Tools** menu, Rose supports several languages in addition to Java. These languages include ANSI C++, Visual C++ and Visual Basic.

**Note:** Rose Enterprise Edition can generate code, update code, and update models. Rose Professional Data Modeler Edition, included in Rational Suite AnalystStudio, does not have these features. It can round-trip engineer database schemas only.

## Modeling Data

You can use Rose to model relational databases. Rational Rose Professional Data Modeler Edition is a database modeling and design tool that uses UML (Unified Modeling Language). It allows you to:

- Support most specific database functions, such as creating tables, columns, indexes, relationships, keys (primary and foreign), and stored procedures.
- Create column constraints, and both DRI (Declarative Referential Integrity) and RI triggers.
- Create custom triggers and their generated trigger code.

### Advantages

Advantages of using Rose Professional Data Modeler are:

- All your business, application, and data models are written in the same industry-standard language, UML, and tool, promoting consistency, traceability, and ease of communication.
- Both forward and reverse engineering of relational databases are supported, facilitating the process of keeping database implementations and models consistent.

## Completing Your Work

Once you have finished your work with a sequence diagram and published your changes to the Web, you are ready to complete your assigned activity for the feature requested and deliver the changes to the project integration stream. To do this, you will use ClearCase LT. To learn more about this process, read *Introducing Rational ClearCase LT*.

## Summary

### For More Information

For more information about Rational Rose, see the UML Tutorial, available on the Rational Solutions for Windows – Online Documentation CD.

For information about Rational Rose RealTime, see the online tutorials available from Rose RealTime online Help. These tutorials address the needs of Rose RealTime users at all levels.

For more information about object-oriented analysis and design, use Extended Help. From the Rose menu, choose **Help > Extended Help**. In the left pane of the Extended Help browser, navigate to a topic under **Guidelines**, for example, **Guidelines > Design Class**.

For more information about using Rational ClearCase LT and UCM with Rational Suite, read *Using Unified Change Management with Rational Suite*.

For general information about Rational ClearCase LT with or without UCM, read *Introducing Rational ClearCase LT*.

## What You Learned in this Chapter

In this chapter, you learned:

- Visual modeling means creating graphical representations of your system's structure and interrelationships.
- In Rose, you use sequence diagrams to elaborate on paths through use cases.
- Rational Rose helps you: create visual models for code and data; generate code; and keep the model synchronized with changed code. ClearCase LT allows you to manage versions of these artifacts as they are modified.
- You can publish read-only copies of your models and diagrams to the Web using Rose. This feature unifies the team by helping you create high-quality architecture models that can be shared, ensuring that all team members have the same understanding of the project.
- Rose supports many languages, including ANSI C++, Visual Basic, Visual C++, and Java.

## What's Next

You just learned about completing a visual model for a feature, and that ClearCase LT is used to complete and deliver activities to the integration stream. Next, you learn about creating reports to communicate the status of software development projects.

# Creating a Use Case Report

# 7

Once you have extended a use case to a visual model, you might want to generate a report consolidating all the information about the use case. Such a report could contain the use case diagram from the visual model in Rose and the corresponding basic flow from the use case in RequisitePro. In this chapter, you learn how to use Rational SoDA to produce that report.

## Audience

This chapter applies to all members of a software development team.

## What Is SoDA?

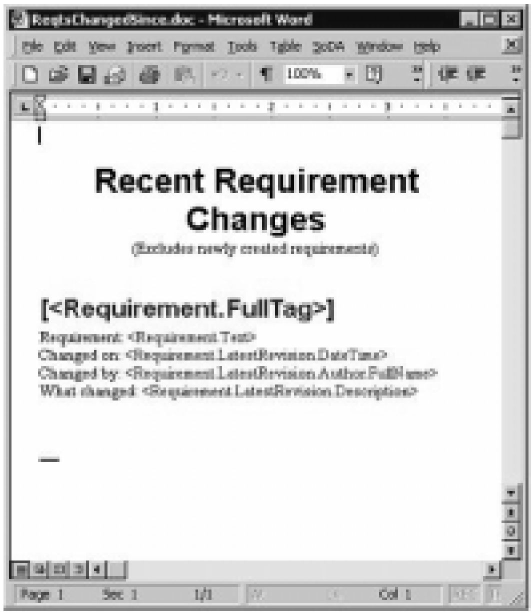
SoDA facilitates the creation of software documentation. It is tightly integrated with many of Rational's tools so that you can easily extract information to report on requirements, designs, tests, and defect status. You use SoDA:

- with ClearCase LT to document versioning information
- with ClearQuest to document reported defects
- with RequisitePro to document your requirements
- with Rose to document your designs
- with TestManager to document your test scripts

### Using SoDA Templates

To generate reports and other documents, SoDA relies on a predefined template for Word (on Windows) or FrameMaker (on UNIX). The template gathers information and formats it into a report, mapping between the document structure and the information source of the data to preserve the consistency of extracting data from multiple sources (see Figure 9 on page 46).

**Figure 9 Sample SoDA Template in Word**


ReqtsChangedSince.doc - Microsoft Word
File Edit View Insert Format Tools Table SoDA Window Help
100%
Page 1 Sec 1 1/1 Col 1

Recent Requirement Changes

(Excludes newly created requirements)

[<Requirement.FullTag>]

Requirement: <Requirement.Text>
Changed on: <Requirement.LatestRevision.DateTime>
Changed by: <Requirement.LatestRevision.Author.FullName>
What changed: <Requirement.LatestRevision.Description>

Figure 9 shows a sample SoDA template that extracts information about requirements from Rational RequisitePro. To add information manually to SoDA reports and documents, you simply use the Microsoft Word or FrameMaker interface and add your text before you generate the report or document. Inserting text into a SoDA template before generating a report or document preserves the information through subsequent cycles of generating the report or document.

You can choose from the many templates provided with SoDA, or you can create your own templates with SoDA's easy-to-use template creation tool.

## Why Generate a Use Case Report?

A use case report gathers into one document both text descriptions of expected system behavior (as described in use case requirements) and a diagram that shows how the system interacts with actors. This report is helpful to nearly every team member:

- An *analyst* can show the report to customers and other stakeholders. Together, they can verify that the project is on the right track. These discussions can be held early in the project, so that the analyst can address problems or gaps before, rather than after, the project ships.

- A *software engineer* can use the report's description of expected system behavior to start writing engineering specifications and designing the system architecture.
- A *test engineer* can use the report to design tests for the use case. From the report, a test engineer can identify the steps to test and determine which conditions to test.
- A *technical writer* can start planning documentation, based on the report's descriptions of how users interact with the system.
- A *usability engineer* can use the report to design usability tests, possibly starting with paper prototypes.

### Generating a Use Case Report

A typical way to generate a use case report using SoDA involves:

1. Selecting a template
2. Identifying a data source
3. Actually generating the report
4. [optional] Publishing your report to the Web, making it easily accessible to all team members

## Summary

### For More Information

To learn more about SoDA, start SoDA and run the SoDA tutorial from SoDA Help.

### What You Learned in this Chapter

In this chapter, you learned:

- SoDA automates the creation of software documentation by creating reports based on templates.
- SoDA contains an easy-to-use tool that assists you with template creation.
- A use case report is useful to all members of your project.

### What's Next

In the next chapter, you learn about using Rational TestFactory to perform reliability tests.

# Reliability Testing 8

Once a new feature or enhancement is implemented and test planning has occurred, you are ready to test. In this chapter, we discuss testing for reliability. In the next chapter, you learn more about functional testing.

## Audience

This unit applies to testers, developers, and other team members responsible for reliability testing.

## Reliability Testing Tools

This chapter describes the following automated testing tools:

- **Rational TestFactory.** Automates testing by combining automatic test generation with source-code coverage analysis. Tests an entire application, including all GUI features and all lines of source code.
- **Rational Purify.** Pinpoints run-time errors and memory leaks in Visual C++ application code and memory usage in Java application code.
- **Rational PureCoverage.** Identifies which parts of your Java, Visual C++, or Visual Basic program have and have not been exercised. Exposes testing gaps so you can prevent untested application code from reaching your users.
- **Rational Quantify.** Profiles your Java, Visual C++, or Visual Basic application to help you identify performance bottlenecks in your code.

## What Is Rational TestFactory?

TestFactory optimizes the productivity of developers and testers by reducing the manual effort required to test software. TestFactory is an Automated Testing tool that generates scripts to thoroughly test Visual Basic, C++, and Java applications. These scripts discover defects and provide extensive code coverage.

You can start using TestFactory early in the development cycle, as soon as a user interface is available to test. You use TestFactory throughout development to verify the reliability of each new build.

### Overview of Process

You use TestFactory as follows:

1. *Instrument* the application to gather information about code coverage.
2. *Map* the application to create a hierarchical list of UI controls.
3. *Run a Pilot* to automatically generate scripts that test the application.

The rest of this section discusses each of these steps and provides pointers for using TestFactory as a complement to Rational Robot.

### Instrumenting the Application

You instrument an application so that TestFactory can optimize the *best scripts* it generates – scripts that provide code coverage of your application. During instrumentation, TestFactory creates a new version of the application’s executable file but does not permanently alter source code.

Anyone can instrument an application, but in many groups, the release engineer creates an instrumented executable file for others to test.

### Mapping the Application

The next step is to create an application map. To create a map, TestFactory thoroughly explores the application’s user interface and gathers detailed information about the user interface and its navigational pathways. TestFactory builds a comprehensive hierarchical application map that it uses as the foundation for automatic test generation.

You can build an application map of the entire application, or you can incrementally include application changes into an existing map.

Once the application map is complete, TestFactory displays a Mapping Summary. Using this report, you can identify changes to the application’s UI.

## Running a Pilot

A Pilot uses the instrumented application and the application map to generate a test script (the best script) that exercises as much of the application as possible. As it builds the best script, the Pilot automatically uncovers severe program defects and generates *defect scripts*. Playing back a defect script allows you to reproduce an error in the application.

To create these scripts, TestFactory thoroughly explores the application’s user interface and source code. After creating these scripts, TestFactory shows exactly which source code and which user interface objects its scripts test.

When it has finished, TestFactory displays a summary of the Pilot run:

- You can examine the best script both to read a human-language outline of the script and also to see code coverage results.
- From a defect result, you can quickly determine the line of code on which the defect occurred and you can easily submit a defect report into ClearQuest.

## Test Suites: Putting It All Together

Use a *test suite* to organize scripts and to run them automatically as a group. In a test suite, you can include:

- Scripts generated by TestFactory – best scripts and defect scripts
- Scripts you record or write in Robot

You can run a test suite locally on your own computer or you can use TestAccelerators to distribute scripts in a test suite to computers on a network, for example, in a test lab.

## Using TestFactory with Rational Robot

TestFactory complements and builds on Rational Robot features. By using both tools, you can develop and run regression tests that validate specific, mission-critical paths through an application. TestFactory automatically generates tests that cover an entire application. It takes advantage of the advanced object recognition and playback features of Robot. TestFactory also provides detailed coverage data on scripts created in Robot.

As described in the next chapter, you use Robot to discover defects based on product requirements. You can use Robot to add verification points to enhance an optimized TestFactory script.

## Additional Testing Tools in Rational Suite

### Rational Purify

Run-time memory-reference errors and memory leaks are some of the most difficult errors to locate and the most important to correct. They often remain undetected until triggered by a random event, so that a program can appear to work correctly when actually it's working only by accident.

Rational Purify is a comprehensive run-time error detection tool that works with Visual C++ and Java programs. Purify can find memory errors in every component of your program, even when you don't have the source code. If Purify detects an error in an area of the application for which the source code is available, it identifies and displays the command that caused the invalid memory reference. For Java programs, Purify analyzes and reports memory usage.

Purify can also collect coverage data as you check your code for errors, pinpointing the part of your program that you have not tested. Using Purify's coverage data, you can make sure that all your code is free of errors.

### Rational PureCoverage

To effectively test an application, you need to know which parts of the application were exercised during a test run and which ones were missed. Without this information, you can waste valuable time editing, compiling, and debugging your software without actually testing the critical problem areas.

With Rational PureCoverage, you can quickly and easily identify the gaps in your testing of Visual C++, Visual Basic, and Java programs.

PureCoverage is especially useful as a companion to Rational Purify and Rational Robot: it can tell you whether you are exercising your code sufficiently for Purify to find all of your memory errors and for Robot to test all of your application's functionality. It is essential to an automated testing environment.

### Rational Quantify

Rational Quantify quickly pinpoints performance bottlenecks in Visual C++, Visual Basic, and Java programs. It takes the difficulty and guesswork out of performance tuning by delivering accurate, repeatable timing data for all the components of your program, even when you don't have the source code.

Quantify gives you the insight you need to write more efficient code and make any program run faster. It can turn everyone on your team into a performance engineer.

## Summary

### For More Information

For more information about:

- **TestFactory.** Read the tutorial, *Getting Started with Rational TestFactory*, and the TestFactory online Help.
- **Purify.** Read *Getting Ahead with Purify* and the Purify online Help.
- **PureCoverage.** Read *Getting Ahead with PureCoverage* and the PureCoverage online Help.
- **Quantify.** Read *Getting Ahead with Quantify* and the Quantify online Help.

These books are all available on the *Rational Solutions for Windows Online Documentation CD*.

### What You Learned in this Chapter

In this chapter, you learned:

- TestFactory automatically maps an application and generates best scripts (which cover the most code in the least number of steps) and defect scripts (which reproduce any errors that TestFactory finds).
- Use TestFactory starting early in the development cycle to test for reliability and to discover severe defects in your application.
- Use test suites to organize test scripts and to run them automatically as a group.
- Additional testing tools include Purify (finds run-time memory errors), PureCoverage (determines testing coverage), and Quantify (identifies performance bottlenecks).

### What's Next

In the next chapter, you learn more about testing an enhancement. You will learn how to use Robot to create a script, include the script in an existing test suite, run the test suite, and handle errors discovered by the tests.

# Functional Testing 9

At this point in the development process, developers have implemented the enhancement. Test engineers have run initial reliability tests. In this chapter, you learn how to perform functional tests on the enhancement.

## Audience

This chapter applies to test engineers and other team members responsible for functional testing.

## What Is Functional Testing?

Functional testing helps you determine whether a system behaves as intended. The most natural way to test a system's behavior is to automate your testing and use the application's GUI to validate that the system responds appropriately to user input. Testing can focus on both the operation and the appearance of GUI objects.

### Working with Test Scripts

During test planning, you write test cases, as described in Chapter 4, *Test Planning*. A test case describes the extent to which you will test an area of the application. It lists the preconditions for performing a test, the input to provide during testing, the variables you will examine, and the expected results of each test.

To implement a test, you start with a test case and create *test scripts*. You then associate the test case with its test scripts. A test script has the following components:

- A set of properties, such as the type and purpose of the script. Typically, you define the script's properties during planning.
- A file containing scripting language commands. You generate a script file when you record activities with Rational Robot.

### Creating a Test Script

TestManager helps you to implement your test cases by generating test scripts. Test cases can be implemented as manual or automated scripts, and therefore need to be associated with a test case. For the purpose of this chapter, we will focus on automated testing.

To create an automated test script using Test Manager select **File > New Test Script**. When the **New Test Script** menu appears, pick the test script type of your choice (GUI or VU). Rational Robot will start and you will then record your test script.

## Recording the Script

You record a script by exercising parts of your application's GUI. When you record a script, Robot translates the activities you perform into scripting language commands. (Robot uses SQABasic for its scripting language. SQABasic resembles Microsoft Visual Basic and contains additional commands tailored for automated testing.) After you record a script, you can reuse it, for example, in regression tests or in test suites.

### Scripts and Modularity

Although you can record a script that starts an application and proceeds through several steps to achieve a certain end result, you may not be able to reuse that script without modifying it.

Instead, you can create a set of scripts that all start with the same steps and conclude by testing different parts of the application. Robot lets you create short modular scripts. You then combine a series of scripts into a test suite. With this technique, you can reuse the same script in different tests. Or, you can reuse a script that has already been recorded to get an application to an appropriate starting place for recording a subsequent script.

For example, you might create the following test scripts:

- A script that logs onto the system
- A scripts that selects an item to purchase
- A script that completes a purchase
- A script that logs you off the system

You then run each test script individually, or run them all at once, in succession, by combining them into a test suite.

### Creating a Verification Point

For each test, you need to create at least one *verification point*, which establishes a baseline value for object properties or data in a specific part of the application. When you play back a script, Robot compares the value it finds to the baseline value you establish. You can include any number of verification points in a script.

## Playing Back the Script On a New Build

While you build test scripts and test suites, developers are typically delivering a new build of the application. You can run your test suites on the new build.

Robot starts the application, interacts with it, captures properties and data at verification points, and quits the application. When it has finished running the script, it displays the results of the test in TestManager.

## Analyzing the Results

TestManager shows which verification points passed and which failed. If a verification point fails, then its script also fails. When scripts fail, you inspect the log display in TestManager and decide how to handle the failure.

### Handling Failures

The outcome of a particular test may change during subsequent iterations as old defects are fixed and new defects and other changes are introduced. There are two types of script failures:

- An intentional change is one in which the script fails due to planned changes in the application. In this case, you want to change the baseline for the verification point. Note that you only want to change the baseline when a test fails because of an intentional change in the application.
- A real error is one in which the script fails with a correct baseline. To report these kinds of errors, submit a defect record using ClearQuest, which is integrated with Rational TestManager.

## Summary

### For More Information

For more information about testing strategy, click **Help > Extended Help** from a Rational Test tool menu. In the Extended Help browser, read the articles under **Concepts**.

To get started with Rational Test tools, read *Getting Started with Robot*, available on the *Rational Solutions for Windows Online Documentation CD*.

### What You Learned in this Chapter

In this chapter, you learned:

- Functional testing helps you determine whether a system behaves as intended.
- Rational TestManager helps you plan, develop, run, and analyze functional tests.
- You develop test scripts by interacting with the application using Robot and including verification points in your scripts.
- You can develop modular scripts, then use test suites to call those scripts. You reuse scripts each time developers deliver a new software build.
- Robot makes it easy to address problems and updates that are discovered during testing.
- ClearQuest's integration with TestManager makes it easy to report errors, finish testing an iteration of development, and initiate the next iteration.

### What's Next

You have learned how to test an application for an iteration of development. In the next chapter, you work on planning the next iteration.

# Planning the Next Iteration 10

Now that you have learned how to implement an enhancement into an existing system, it is time to learn more about planning for the next iteration in your project's development.

## Audience

This chapter applies to all members of a software development team.

## Assessing the State of your Project

In *Functional Testing*, we described how you use ClearQuest to report a defect in the software. In this chapter, you learn how using ClearQuest helps you to assess the state of your project by creating charts and queries.

ClearQuest is a change request management tool that helps you track and manage all the activities (such as defects and enhancement requests) associated with a project.

ClearQuest stores its information in a database, and comes with a ready-to-use *database schema.* A database schema describes the fields in the database). ClearQuest is easy to change; an administrator can customize and define queries, records, fields, activities, and states specific to your development process.

### Showing the Workload

At the end of an iteration, you want to review each project member's workload so that you can most effectively allocate work for the next iteration. Using ClearQuest, you can display a workload chart. Within a workload chart, you use ClearQuest to learn more about a specific team member's workload. This feature may be helpful if you are interested in learning about the defects and related details assigned to an individual.

## Other Planning Activities

During an iteration, you usually work both to correct defects and to implement enhancements. As part of planning, you might also use ClearQuest or RequisitePro to identify the work to do in the next iteration.

During iteration planning, you can produce a Rational SoDA report showing the defects and enhancements planned for the next iteration.

## What Will Happen in the Next Iteration?

The next iteration will proceed much as this one has. Once it's planned, the following activities will transpire:

- An analyst discusses planned enhancements with stakeholders. Using RequisitePro and Rose, the analyst creates one or more use cases and supplies step-by-step details, including basic flow and alternative flows.
- A test engineer uses Rational TestManager to plan the tests for this iteration. The engineer creates a test plan, develops test requirements, and designs tests.
- A developer uses visual modeling techniques available in Rose to describe how planned enhancements fit within the architecture of the system.
- Anyone on the team can use SoDA to create a use case report. This report is useful during discussions with stakeholders and design sessions.
- Developers use Rose to initiate implementation of the enhancement.
- Test engineers use TestFactory, Purify, Quantify, and PureCoverage to verify the iteration's reliability.
- Test engineers use Rational TestManager and Robot to verify that the enhancements meet requirements, that defects are fixed correctly, and that no regression failures have occurred.
- Throughout the project, team members use Unified Change Management (UCM) to manage change in their system's development from requirements to release.
- All members of the team use ClearCase LT to make changes to project artifacts. Project members each work in private development workspaces. When they finish their work, they deliver artifacts to the team's public integration workspace.
- Meanwhile, project and group managers use ClearQuest, SoDA, and RequisitePro to assess that state of the project. Later, they use these tools to plan subsequent iterations.

## Summary

### For More Information

To learn more about topics described in this book, consider taking a Rational University course. In these courses, you can get hands-on experience with a specific Rational tool, or you can learn more about software engineering principles (Object Oriented Analysis and Design, Automating Software Test, and so on).

To learn more about these courses, please see http://www.rational.com/university.

### What You Learned in this Chapter

In this chapter, you learned:

- ClearQuest is a powerful tool that helps you manage and monitor change requests on your project.

### What You Learned in this Book

- Rational Suite unifies your team by enhancing team communication.
- Rational Suite optimizes individual team member productivity by providing market-leading development tools for each member of a software development team.
- Rational Suite simplifies adoption by providing a comprehensive set of integrated tools that have simple installation, licensing, and user support.
- Rational Suite supports the entire development life cycle and the primary roles on a development team – analysts, developers, testers, and managers.

### What's Next

Congratulations! You have learned how Rational Suite helps you plan, design, implement, and test software applications. Your next job is to learn more about the tools you will use on your next project and to get to work! We hope that by using Rational Suite to plan, design, implement, and test applications, your team will successfully meet the challenges of rapidly developing high-quality software.

## Rational Suite Summary

This table shows which tools are included with each edition of Rational Suite.

| Rational Tool | Analyst Studio | Development Studio (Windows or UNIX) | Development Studio - RealTime Edition | Test Studio | Team Unifying Platform | Enterprise |
|---|---|---|---|---|---|---|
| **Rational Unified Process** | X | X | X | X | X | X |
| **Rational RequisitePro** | X | X | X | X | X | X |
| **Rational ClearQuest** | X | X | X | X | X | X |
| **Rational SoDA for Word (Windows) or for FrameMaker (UNIX)** | X | X | X | X | X | X |
| **Rational ClearCase LT** | X | X (Windows) | X | X | X | X |
| **Rational TestManager** | X | X | X | X | X | X |
| **Rational Rose** | Data Modeler Edition | Enterprise Edition | RealTime Edition | | | Enterprise Edition |
| **Rational Robot** | | | | X | | X |
| **Rational TestFactory** | | | | X | | X |
| **Rational PureCoverage** | | X | X | X | | X |
| **Rational Purify** | | X | X | X | | X |
| **Rational Quantify** | | X | X | X | | X |
| **Rational QualityArchitect** | | X (Windows) | | | | X |

# Glossary

**activity.** A unit of work that a team member performs.

**actor.** Someone or something, outside the system or business, that interacts with the system or business.

**analyst.** A person who determines what the system does, specifies and manages requirements, and represents the user's needs to the development organization.

**artifact.** A piece of information that is produced, modified, or used by a process; defines an area of responsibility; and is subject to version control. There are many types of artifacts, including requirements, models, model elements, documents, and source code.

**automated testing.** A testing technique wherein you use software tools to replace repetitive and error-prone manual work. Automated testing saves time and enables a reliable, predictable, repeatable, and accurate process.

**class.** In object-oriented programming, a set of objects that share the same responsibilities, relationships, operations, attributes, and semantics.

**component.** A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture.

**component-based architecture.** A design technique in which a software system is decomposed into individual components.

**configuration management.** Helps teams control their day-to-day management of software development activities as software is created, modified, built, and delivered. Comprehensive software configuration management includes version control, workspace management, build management, and process control to provide better project control and predictability.

**developer.** A person who determines how the system works; defines the architecture; and creates, modifies, tests, and manages the code.

**Extended Help.** A powerful feature of Rational Suite that provides links from Rational Suite products to the Rational Unified Process and any customized information you want to add.

**forward engineering.** The process of generating code from a Rational Rose visual model. See *visual model*.

**iterative development.** The process of delivering a distinct sequence of executable files according to a plan and evaluation criteria over the course of a project. Each executable file is more robust or contains more features than the previous executable file; each new iteration moves you closer to the goal of delivering a successful project.

**method.** In object-oriented programming, the implementation of an operation or procedure.

**metrics.** The measurements of project activity.

**object.** In object-oriented programming, a software component that contains a collection of data and methods (procedures) for operating on that data.

**phase.** The time between two major project milestones, during which a well-defined set of objectives is met, artifacts are completed, and decisions are made to move or not move into the next phase.

**Rational Administrator.** Tool that manages Rational projects and associates repositories to define a Rational project. For more information, see *Using the Rational Administrator*.

**Rational ClearCase LT.** Provides comprehensive configuration management, including version control, workspace management, and process control.

**Rational ClearQuest.** A highly customizable Windows and Web-based change request management tool that lets users track any type of change activity –defects and fixes, enhancement requests, documentation changes, and so on –throughout the software development lifecycle.

**Rational PureCoverage.** Automatically pinpoints areas of code that have not been tested.

**Rational Purify.** Automatically pinpoints hard-to-find runtime memory errors in Windows NT applications.

**Rational Quantify.** Automatically pinpoints performance bottlenecks in Visual Basic, Visual C++, and Java applications.

**Rational RequisitePro.** Helps teams easily and comprehensively organize, prioritize, track, and control changing requirements of a system or application. Rational RequisitePro does this through a deep integration with Microsoft Word and a secure, multi-user database.

**Rational Robot.** Helps with functional testing by automating record and playback of test scripts. Lets you organize, write, and run test suites, and capture and analyze the results.

**Rational Rose.** The world's leading visual component modeling and development tool; lets you model software applications that meet current business needs.

**Rational SoDA for Word.** Software Documentation Automation – Overcomes the obstacles of consolidating data from different development tools. Lets you automate the creation of comprehensive software, systems, and project documents from multiple sources.

**Rational Suite.** An easy-to-adopt-and-support solution that unifies software teams and optimizes the productivity of analysts, developers, testers, and project managers.

**Rational Suite AnalystStudio.** Edition of Rational Suite optimized for system definition. Contains the Team Unifying Platform –Rational Unified Process, RequisitePro, ClearCase LT, ClearQuest, SoDA, and TestManager – and Rational Rose (Professional Data Modeler Edition).

**Rational Suite DevelopmentStudio.** Edition of Rational Suite optimized for software development. Contains the Team Unifying Platform –Rational Unified Process, RequisitePro, ClearCase LT, ClearQuest, SoDA, and TestManager – plus Rational Rose (Enterprise Edition), Rational Purify, Rational Quantify, and Rational PureCoverage.

**Rational Suite DevelopmentStudio - RealTime Edition.** Edition of Rational Suite optimized for system developers and designers of real-time or embedded systems. Contains the Team Unifying Platform – Rational Unified Process, RequisitePro, ClearCase LT, ClearQuest, SoDA, and TestManager – plus Rational Rose RealTime, Rational Purify, Rational Quantify, and Rational PureCoverage.

**Rational Suite Enterprise.** Edition of Rational Suite containing all Rational Suite tools.

**Rational Suite Team Unifying Platform.** Edition of Rational Suite optimized for all members of software development teams to maximize productivity and quality. This Suite editions includes these Team Unifying Platform –Rational Unified Process, RequisitePro, ClearCase LT, ClearQuest, SoDA, and TestManager. All other Suite editions include the Team Unifying Platform.

**Rational Suite TestStudio.** Edition of Rational Suite optimized for test engineers. Contains the Team Unifying Platform –Rational Unified Process, RequisitePro, ClearCase LT, ClearQuest, SoDA, and TestManager – and Rational PureCoverage, Rational Purify, Rational Quantify, Rational Robot and Rational TestFactory.

**Rational Synchronizer.** Uses rules, either predefined or user-supplied, to give you a quick start on new work. Creates or updates project items based on the existence of other items in your project, ensuring that details do not fall through the cracks.

**Rational TestFactory.** Automates reliability testing by combining automatic test generation with source code coverage analysis.

**Rational Unified Process.** A Web-enabled, searchable knowledge base that enhances team productivity and delivers software best practices via guidelines, templates, and Tool Mentors for all critical software development activities.

**real-time application.** An application or system with stringent requirements for latency, throughput, reliability, and availability.

**requirement.** A condition or capability of a system, either derived directly from user needs or stated in a contract, standard, specification, or other formally imposed document.

**requirements management.** A systematic approach to eliciting, organizing, and documenting a system's changing requirements, and establishing and maintaining agreement between the customer and the project team.

**reverse engineering.** The process of updating a Rose visual model from code, so that the visual model and code match. See *visual model*.

**risk.** The probability of adverse project impact (for example, schedule, budget, or technical).

**risk management.** Consciously identifying, anticipating, and addressing project risks and devising plans for risk mitigation, as a way of ensuring the project's success.

**role.** The behavior and responsibilities of an individual, or a set of individuals working together as a team, within the context of a software engineering organization.

**round-trip engineering.** The ability to do both forward and reverse engineering as often as needed.

**test case.** A set of test inputs, execution conditions, and expected results developed for a particular objective.

**test configuration.** The sequence of attributes for potential organizational structures of the system that you will apply to your test cases.

**test engineer.** A person who creates, manages, and executes tests; ensures that the software meets all its requirements; and reports the results and verifies fixes.

**test input.** Any artifact used to develop a system, and can be used to influence testing.

**test plan.** Contains information about the purpose and goals of testing within a project, and the strategies to be used to implement and execute testing.

**Tool Mentor.** Step-by-step instructions on how to use a specific Rational tool to perform an activity described in the Rational Unified Process.

**traceability.** The ability to trace one project element to other, related project elements.

**Unified Modeling Language (UML).** The industry-standard language for specifying, visualizing, constructing, and documenting software systems. It simplifies software design, and communication about the design.

**use case.** A sequence of actions a system performs that yields observable results of value to a particular actor. A use case specification contains all the main, alternate, and exception flows of events related to producing the "observable result of value."

**version control.** The process of tracking the revision history of files and directories.

**vision document.** A document that contains a high-level view of the user's or customer's understanding of the system to be developed.

**visual model.** A graphic representation of a system's structure and interrelationships.

**workflow.** The sequence of activities performed by roles to attain an observable value.

# Index