

Rational Suite®

Tutorial

VERSION: 2003.06.00

PART NUMBER: 800-026081-000

WINDOWS

Legal Notices

©1998-2003, Rational Software Corporation. All rights reserved.

Part Number: 800-026081-000

Version Number: 2003.06.00

This manual (the "Work") is protected under the copyright laws of the United States and/or other jurisdictions, as well as various international treaties. Any reproduction or distribution of the Work is expressly prohibited without the prior written consent of Rational Software Corporation.

The Work is furnished under a license and may be used or copied only in accordance with the terms of that license. Unless specifically allowed under the license, the Work or copies of it may not be provided or otherwise made available to any other person. No title to or ownership of the manual is transferred. Read the license agreement for complete terms.

Rational Software Corporation, Rational, Rational Suite, Rational Suite ContentStudio, Rational Apex, Rational Process Workbench, Rational Rose, Rational Summit, Rational Unified process, Rational Visual Test, AnalystStudio, ClearCase, ClearCase Attache, ClearCase MultiSite, ClearDDTS, ClearGuide, ClearQuest, PerformanceStudio, PureCoverage, Purify, Quantify, Requisite, RequisitePro, RUP, SiteCheck, SiteLoad, SoDa, TestFactory, TestFoundation, TestMate and TestStudio are registered trademarks of Rational Software Corporation in the United States and are trademarks or registered trademarks in other countries. The Rational logo, Connexis, ObjecTime, Rational Developer Network, RDN, ScriptAssure, and XDE, among others, are trademarks of Rational Software Corporation in the United States and/or in other countries. All other names are used for identification purposes only and are trademarks or registered trademarks of their respective companies.

Portions covered by U.S. Patent Nos. 5,193,180 and 5,335,344 and 5,535,329 and 5,574,898 and 5,649,200 and 5,675,802 and 5,754,760 and 5,835,701 and 6,049,666 and 6,126,329 and 6,167,534 and 6,206,584. Additional U.S. Patents and International Patents pending.

U.S. Government Restricted Rights

Licensee agrees that this software and/or documentation is delivered as "commercial computer software," a "commercial item," or as "restricted computer software," as those terms are defined in DFARS 252.227, DFARS 252.211, FAR 2.101, OR FAR 52.227, (or any successor provisions thereto), whichever is applicable. The use, duplication, and disclosure of the software and/or documentation shall be subject to the terms and conditions set forth in the applicable Rational Software Corporation license agreement as provided in DFARS 227.7202, subsection (c) of FAR 52.227-19, or FAR 52.227-14, (or any successor provisions thereto), whichever is applicable.

Warranty Disclaimer

This document and its associated software may be used as stated in the underlying license agreement. Except as explicitly stated otherwise in such license agreement, and except to the extent prohibited or limited by law from jurisdiction to jurisdiction, Rational Software Corporation expressly disclaims all other warranties, express or implied, with respect to the media and software product and its documentation, including without limitation, the warranties of merchantability, non-infringement, title or fitness for a particular purpose or arising from a course of dealing, usage or trade practice, and any warranty against interference with Licensee's quiet enjoyment of the product.

Third Party Notices, Code, Licenses, and Acknowledgements

Portions Copyright ©1992-1999, Summit Software Company. All rights reserved.

Microsoft, the Microsoft logo, Active Accessibility, Active Client, Active Desktop, Active Directory, ActiveMovie, Active Platform, ActiveStore, ActiveSync, ActiveX, Ask Maxwell, Authenticode, AutoSum, BackOffice, the BackOffice logo, bCentral, BizTalk, Bookshelf, ClearType, CodeView, DataTips, Developer Studio, Direct3D, DirectAnimation, DirectDraw, DirectInput, DirectX, DirectXJ, DoubleSpace, DriveSpace, FrontPage, Funstone, Genuine Microsoft Products logo, IntelliEye, the IntelliEye logo, IntelliMirror, IntelliSense, J/Direct, JScript, LineShare, Liquid Motion, Mapbase, MapManager, MapPoint, MapVision, Microsoft Agent logo, the Microsoft eMbedded Visual Tools logo, the Microsoft Internet Explorer logo, the Microsoft Office Compatible logo, Microsoft Press, the Microsoft Press logo, Microsoft QuickBasic, MS-DOS, MSDN, NetMeeting, NetShow, the Office logo, Outlook, PhotoDraw, PivotChart, PivotTable, PowerPoint, QuickAssembler, QuickShelf, RelayOne, Rushmore, SharePoint, SourceSafe, TipWizard, V-Chat, VideoFlash, Visual Basic, the Visual Basic logo, Visual C++, Visual C#, Visual FoxPro, Visual InterDev, Visual J++, Visual SourceSafe, Visual Studio, the Visual Studio logo, Vizact, WebBot, WebPIP, Win32, Win32s, Win64, Windows, the Windows CE logo, the Windows logo, Windows NT, the Windows Start logo, and XENIX, are either trademarks or registered trademarks of Microsoft Corporation in the United States and/or in other countries.

Sun, Sun Microsystems, the Sun Logo, Ultra, AnswerBook 2, medialib, OpenBoot, Solaris, Java, Java 3D, ShowMe TV, SunForum, SunVTS, SunFDDI, StarOffice, and SunPCi, among others, are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Purify is licensed under Sun Microsystems, Inc., U.S. Patent No. 5,404,499.

Licensee shall not incorporate any GLOBEtrouter software (FLEXlm libraries and utilities) into any product or application the primary purpose of which is software license management.

BasicScript is a registered trademark of Summit Software, Inc.

Design Patterns: Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Copyright © 1995 by Addison-Wesley Publishing Company, Inc. All rights reserved.

Additional legal notices are described in the legal_information.html file that is included in your Rational software installation.

Contents

Preface	xi
Audience	xi
Other Resources	xi
Rational Suite Documentation Roadmap	xii
Contacting Rational Customer Support	xiii
1 Welcome to Rational Suite	15
Principles of Software Development	15
Rational Suite Can Help	18
What Is Rational Suite?	19
Tools That Unify Your Team	19
Rational Suite Team Unifying Platform	19
Tools for Analysts	21
Rational Suite AnalystStudio	21
Tools for Developers	21
Rational Suite DevelopmentStudio	22
Rational Suite DevelopmentStudio – RealTime Edition	23
Tools for Testers	23
Rational Suite TestStudio	23
Rational Suite Enterprise	24
Rational Suite: A Complete Lifecycle Solution	25
For More Information	26
What's Next	26
2 About This Tutorial	27
Prerequisites	27
Determining Which Rational Suite Tools Are Installed	27
ClassicsCD.com: The Tutorial Sample Application	28
Tutorial Background	28
Installing the Tutorial Sample Application and Related Files	28
Getting Started	30
Registering the Project	30
Associating the Change Request Database with the Project	31
Using the Application	33
Discovering What to Build	34

How to Use This Tutorial	35
Summary	36
What You Learned in This Chapter	36
What's Next	36
3 Learning About the Rational Unified Process.	37
Audience	37
Getting Your Bearings	37
What Is the Rational Unified Process (RUP)?	38
The Rational Unified Process and Rational Suite	39
Learning the Mechanics	39
The Process at a Glance	40
Key Concepts	41
Exploring a Workflow	43
Starting with Actors and Use Cases	44
Tool Mentors: Using Rational Tools to Support the Process	45
Learning the Basics	46
Summary	47
For More Information	47
Cleaning Up	47
What You Learned in This Chapter	47
What's Next	47
4 Managing Change to Project Artifacts.	49
Audience	49
What Is Unified Change Management?	49
UCM Tools	50
Using the Tools with UCM – ClearQuest and ClearCase LT	51
Unifying Code and Content for Web Development	53
Summary	53
For More Information	53
What You Learned in This Chapter	54
What's Next	54

5	Creating Requirements	55
	Audience	55
	Getting Your Bearings	55
	Why Worry About Requirements?	57
	Where Do Requirements Come From?	57
	Managing Requirements	58
	Using RequisitePro	58
	Starting with a Use Case	58
	Why Work with Use Cases?	58
	How Does RequisitePro Handle Requirements?	61
	Learning More About Use Cases	61
	Continuing Use Case Work Using Rose	61
	Working with a Use Case Diagram	62
	Associating the Rose Model with the RequisitePro Project	63
	Creating a New Requirement	64
	Looking at Requirements in the Database	65
	Linking to Another Requirement	66
	Traceability and Suspect Links	67
	Other Requirement Types	67
	When Have You Finished Gathering Requirements?	68
	Extended Help	68
	Summary	69
	For More Information	69
	Cleaning Up	69
	What You Learned in This Chapter	69
	What's Next	69
6	Test Planning	71
	Audience	71
	Getting Your Bearings	71
	What Is Test Planning?	72
	Managing Risk	72
	Making a Plan and Measuring Progress	72
	Developing a Test Plan	73
	Organizing Your Test Plan	73
	Determining What to Test	75

Working with Test Cases	76
Test Inputs from Rational Rose	77
Test Inputs from Rational RequisitePro	77
Traceability and Suspect Links	79
Elaborating on Test Cases	79
Understanding the Impact of Test Planning	79
Continuing with Test Planning	80
Risks and Resources	80
Types of Tests to Perform	81
Stages of Testing	81
Project Scheduling	81
More on Test Artifacts	82
Summary	83
For More Information	83
Cleaning Up	83
What You Learned in This Chapter	83
What's Next	83
7 Modeling the Enhancement	85
Audience	85
Getting Your Bearings	85
What Is Visual Modeling?	86
Using Rational Rose	86
Visual Modeling and the Tutorial	86
Working with a Sequence Diagram	86
Opening a Sequence Diagram	87
Adding Messages for the Enhancement	89
Publishing Part of the Model to the Web	90
Continuing Work with the Sequence Diagram	91
Refining the Objects	91
Implementing Code	92
Modeling Data	93
Benefits	93
Summary	93
For More Information	93
Cleaning Up	94
What You Learned in This Chapter	94
What's Next	94

8 Communicating Project Status	95
Audience	95
Getting Your Bearings	95
Managing Project Status	96
What Is SoDA?	96
Using SoDA Templates	96
Why Generate a Use Case Report?	97
Creating the Use Case Report	97
Working with SoDA Templates	99
What Is ProjectConsole?	99
Using the Project Web Site	100
Working with Project Metrics	100
Summary	103
For More Information	103
Cleaning Up	103
What You Learned in This Chapter	103
What's Next	103
9 Reliability Testing	105
Audience	105
Reliability Testing Tools	105
Run-Time Analysis Tools in Rational Suite	106
Rational Purify	106
Rational PureCoverage	107
Rational Quantify	108
Using Purify, PureCoverage, and Quantify with Other Rational Tools	108
Summary	109
For More Information	109
What You Learned in This Chapter	109
What's Next	109
10 Functional Testing	111
Audience	111
Getting Your Bearings	111
What Is Functional Testing?	112
Working with Test Scripts	112
Scripts and Modularity	112
Getting to a Starting Point	113
Working with Test Scripts	113

Recording the Script	115
Starting to Record the Script	115
Creating a Verification Point	116
Finishing the Recording Session	117
Adding a Test Script to a Suite	118
Incorporating a Test Script into Your Test Plan.	119
Playing Back the Script on a New Build	120
Analyzing the Results	121
Handling Failures	121
Handling an Intentional Change.	121
Handling a Real Error.	122
Reporting the Error.	123
Summary.	125
For More Information	125
Cleaning Up	125
What You Learned in This Chapter.	125
What's Next.	125
11 Planning the Next Iteration	127
Audience	127
Getting Your Bearings	127
Assessing the State of your Project	128
Showing the Workload	128
Working with Enhancement Requests	130
Other Planning Activities	132
What Will Happen in the Next Iteration?	132
Summary.	134
For More Information	134
Cleaning Up	134
What You Learned in This Chapter.	134
What You Learned in This Tutorial	134
What's Next.	135
Glossary	137
Index	143

Preface

Rational Suite delivers a comprehensive set of integrated tools that embody software engineering best practices and span the entire software development lifecycle. This tutorial teaches you the basics of using Rational Suite to plan, design, implement, and test applications. It also points you to additional information so that you can learn more on your own.

Audience

Read this tutorial if you:

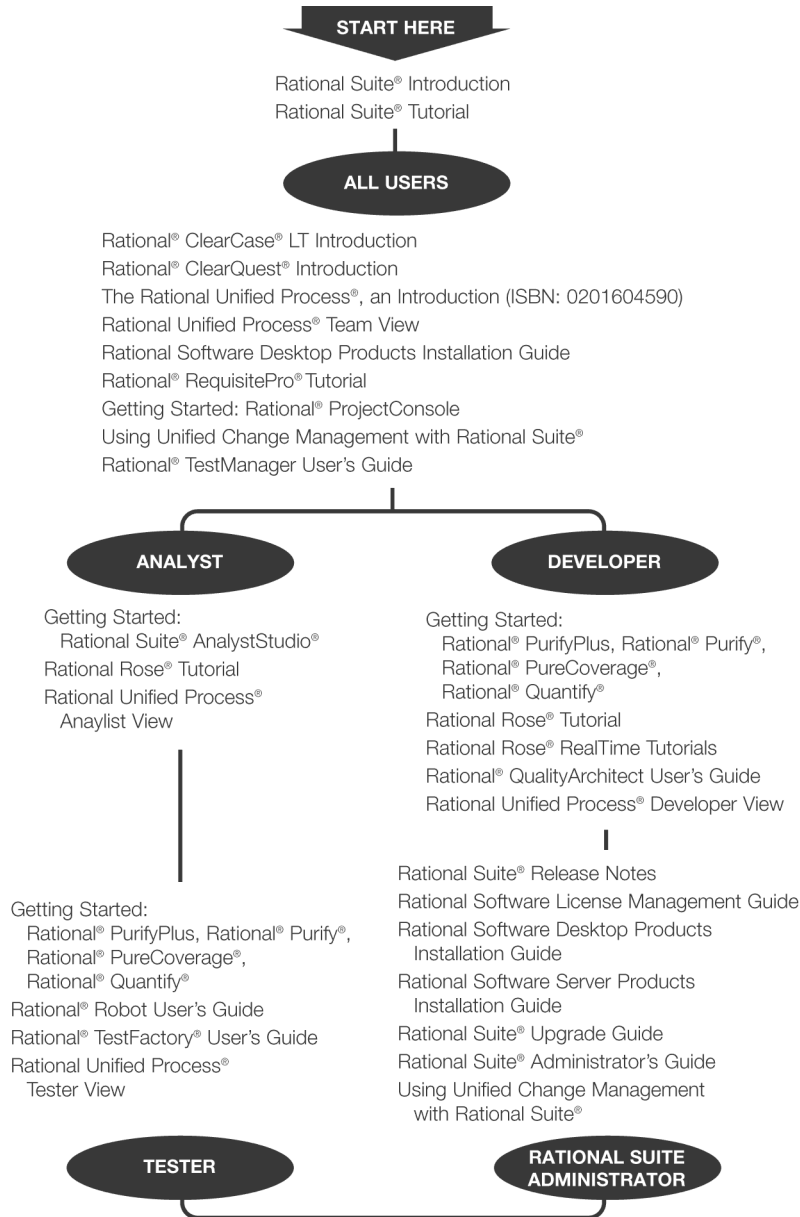
- Are a member of a development team – an analyst, developer, tester, project leader, or manager.
- Have experience with some aspect of Windows-based application development.

You do not need prior experience with any Rational tools to use this tutorial.

Other Resources

- 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>.
- For articles, discussion forums, and Web-based training courses on developing software with Rational Suite products, join the Rational Developer Network by selecting **Start > Programs > Rational Software > Logon to the Rational Developer Network**.

Rational Suite Documentation Roadmap



Contacting Rational Customer Support

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

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

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

- Your name, company name, telephone number, and e-mail address
- Your operating system, version number, and any service packs or patches you have applied
- Product name and release number
- Your case ID number (if you are following up on a previously reported problem).

Think about your last software project. Was it delivered on time? Was it released within its budget? Was communication between team members clear and timely? Did your team maintain consistency throughout the project while it defined requirements, developed designs, and wrote code? 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 results often don'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?

Principles of Software Development

Rational Software Corporation helps organizations overcome many software development issues while accelerating time to market and improving quality. The Rational solution helps organizations develop software through a combination of:

- Software engineering best practices.
- Integrated tools that automate these best practices.
- Professional services that accelerate the adoption and implementation of these best practices and tools (see Figure 1).

Figure 1 The Rational Best Practices, Tools, and Services



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 project lifecycle. The project team plans, develops, and tests an identified subset of system functionality for each 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 you:

- Make your project more predictable.
- Collect feedback early.
- Identify and eliminate risks early in the project.
- Test continuously throughout the project lifecycle.

Manage Requirements

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

Managing requirements means that you understand how changing requirements affect your project and 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 nontrivial, 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 helps you reduce the size and complexity of your application and enhance maintainability and extensibility so your systems are more robust and resilient.

Model Software Visually

Visual modeling helps you manage software design complexity. At its simplest level, visual modeling means creating a graphical blueprint of your system's architecture. Visual models can also help you detect inconsistencies between requirements, designs, and implementations. They help you evaluate your system's architecture, ensuring sound design.

Visual models also improve communication across your entire team because they concisely convey a lot of information. With Rational tools, you can create visual models in the *Unified Modeling Language (UML)*, the industry-standard language for visualizing and documenting software systems.

Continuously Verify Quality

Verifying product quality means that you perform activities such as testing to ensure quality of the code, documentation, and any product-related training. Testing includes verifying that the system delivers required functionality, reliability, and the ability to perform under load. It also means that there are effective user support materials.

An important benefit of iterative development is that you can begin testing early in the development process. Testing every iteration helps you discover problems early and expose inconsistencies between 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 enhancement and change requests, defining repeatable development processes, and reliably reproducing software builds.

As change propagates throughout a project, clearly defined and repeatable change process guidelines help facilitate clear communication about progress. Making team members aware of change helps you control risks associated with unmanaged change.

Rational Suite Can Help

To put these software development principles to work, Rational Software offers Rational Suite, a family of market-leading software development tools supported by the Rational Unified Process. These tools help you throughout the project lifecycle.

Rational Suite packages the tools and the process into several editions, each of which is customized for specific practitioners on your development team, including analysts, developers, and testers.

Alone, these tools have helped organizations around the world successfully create software. Integrated into Rational Suite, they:

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

What Is Rational Suite?

Rational Suite editions are sets of tools customized for every member of your team. Each Suite edition contains the tools from the Rational Suite Team Unifying Platform. The Team Unifying Platform is a common set of tools that focus on helping your team perform more effectively. Each Rational Suite edition also contains tools selected for a specific practitioner on your development team. The following sections describe each Suite edition and the tools they contain.

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. It provides best practices and integrated tools for managing change, building high-quality applications, and communicating results from requirements to release.

Rational Suite Team Unifying Platform is useful to project members who need access to common project information, 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 edition.

The Team Unifying Platform is included in every Rational Suite edition. It contains the following tools:

Rational Unified Process (RUP)

The RUP is an online collection of software best practices (see pages 16 – 18 for an overview) that guide your team through the software development process. The Rational Unified Process (also known as the RUP) provides guidelines, templates, and Tool Mentors (instructions for applying the guidelines to specific Rational tools) for each phase of the development lifecycle. Additionally, you can customize the RUP to:

- Use only the process components you need for each stage of a project.
- Develop your own process components.
- Exchange best practices with peers and industry leaders.

Rational RequisitePro

RequisitePro helps you organize, prioritize, track, and control changing project requirements. With the RequisiteWeb interface, users can access, create, and manage requirements from a Web browser.

Rational ClearQuest

ClearQuest manages change activity associated with software development, including enhancement requests, defect reports, and documentation modifications. The ClearQuest Web interface provides all major ClearQuest operations, such as submitting records, finding records, creating or editing queries and reports, and creating shortcuts, from a Web browser. ClearQuest MultiSite helps you share change request information across a geographically distributed team.

Rational SoDA

SoDA automatically generates project documents by extracting information from files you produce during project development, including: source code and files produced with Rational tools. SoDA uses predefined templates or your own customized templates to format the information. SoDA is integrated with Microsoft Word for ease of use and easy customizing.

Rational ClearCase LT

ClearCase LT provides software configuration management and a built-in process to track changes to all software project assets, including: requirements, visual models, and code. It also provides a Web interface, allowing users to perform all major ClearCase LT operations. ClearCase LT supports *Unified Change Management*, Rational's best practices process for managing change and controlling workflow.

Rational TestManager

TestManager helps you create real-world functional and multiuser tests to determine the reliability and performance of Web, multi-tier, and database applications. TestManager tracks how many tests have been planned, scripted, and carried out; which requirements have been covered; and the number of tests that have passed and failed. TestManager allows your team to objectively assess project status, and create and customize reports to communicate these findings to project stakeholders.

Rational ProjectConsole

ProjectConsole helps you track project metrics by automatically generating charts and gauges from data produced during software development. ProjectConsole is integrated with Microsoft Project so that you can create a centralized project plan. ProjectConsole helps you organize project artifacts on a central Web site so all team members can view them.

Rational Developer Network

The Rational Developer Network helps you expand and hone professional skills, and stay ahead of the technology curve. The Rational Developer Network is an online community for software professionals, providing useful information, an opportunity to exchange ideas with other software professionals, and best practices for software development teams. The network delivers white papers, documentation, articles, and training.

Tools for Analysts

An analyst's role is to:

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

Rational Suite AnalystStudio

Rational Suite AnalystStudio supports the analysts on your team. It contains the Team Unifying Platform and:

Rational Rose (Professional Data Modeler Edition)

Rose enables visual modeling of databases, architectures, and components using the industry-standard Unified Modeling Language (UML). The UML is a language for specifying, visualizing, constructing, and documenting software systems. This edition of Rose integrates the modeling environment with the database design environment, mapping the object and data models, tracking changes across business, application and data models.

Tools for Developers

A developer's role is to:

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

Rational Suite offers two editions to support the developers and architects on your team: Rational Suite DevelopmentStudio and Rational Suite DevelopmentStudio – RealTime Edition.

Rational Suite DevelopmentStudio

Rational Suite DevelopmentStudio supports system developers, designers, and architects. Rational Suite DevelopmentStudio contains the Team Unifying Platform and:

Rational Rose (Enterprise Edition)

Rose enables visual modeling of architectures, components, and data using UML. Rational Rose can regenerate the framework of your code in Java, C++, Microsoft Visual Basic, and other popular languages. Rose is also tightly integrated with Microsoft Visual Studio and IBM VisualAge for Java. The Rose round-trip engineering feature helps you automate the process of maintaining consistency between a model and its implementation.

Rational QualityArchitect

Quality Architect is a Rational Rose add-in that automates the mechanical aspects of test code creation by generating test code from visual models. This feature of Rose allows developers to automatically generate component tests by building stubs and drivers before an application is complete. This is important because early testing helps to reduce project risk. Your team can determine how a potential system architecture meets functional and performance requirements, before developing the design further. Rational QualityArchitect supports Enterprise JavaBeans, COM, COM+, and DCOM models.

Rational PureCoverage

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

Rational Purify

Purify pinpoints run-time errors and memory leaks in Visual C++ application code, and errors related to garbage-collection in Java and .NET managed application code.

Rational Quantify

Quantify profiles your Java, Visual C++, Visual Basic, Visual Basic.NET, or Visual C#.NET application to help you identify performance issues in your source 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 Enterprise Edition with Rational Rose RealTime Edition.

Rational Rose (RealTime Edition)

Rose Real-Time 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 to compile and then observe simulations of your UML designs on the host or target platform. As a result, 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.

This tutorial does not discuss Rose RealTime in detail. To learn more about Rose RealTime, see the online tutorials available from the Rose RealTime Help menu.

Tools for Testers

A tester's role is to:

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

Rational Suite TestStudio

Rational Suite TestStudio is the Rational Suite edition designed for testers. It contains the Team Unifying Platform and:

Rational Robot

Robot facilitates functional and performance testing by automating the recording and playback of test scripts for both functional and performance testing. Allows you to write, troubleshoot, and run tests, and to capture results for analysis.

Rational TestFactory

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 PureCoverage

PureCoverage identifies which parts of your Java, Visual C++, or Visual Basic program have and have not been exercised. Uses test scripts to drive the application and expose testing gaps so you can prevent untested application code from reaching users.

Rational Purify

Purify pinpoints run-time errors and memory leaks in Visual C++ application code, and errors related to garbage-collection in Java application code. Purify does this by using test scripts to drive the application.

Rational Quantify

Quantify profiles your Java, Visual C++, or Visual Basic application to help you identify performance bottlenecks in your code. Quantify does this by using test scripts to drive the application.

Rational Suite Enterprise

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

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

Rational Suite: A Complete Lifecycle Solution

Rational Suite provides an integrated solution for development teams and offers tailored editions to best suit your team's needs. This table shows which tools are included with each edition of Rational Suite.

		Project Leaders/ Managers	Analysts	Developers				Testers	All Roles
		Team Unifying Platform	Analyst Studio	DevelopmentStudio		DevelopmentStudio RealTime Edition		TestStudio	Enterprise
		Windows	Windows	Windows	UNIX	Windows	UNIX	Windows	Windows
Team Unifying Platform	Rational Unified Process	•	•	•	•	•	•	•	•
	Rational RequisitePro	•	•	•	• ^{WEB}	•	• ^{WEB}	•	•
	Rational ClearQuest	•	•	•	•	•	•	•	•
	Rational SoDA	• ^{SW}	• ^{SW}	• ^{SW}	• ^{SF}	• ^{SW}	• ^{SF}	• ^{SW}	• ^{SW}
	Rational ClearCase LT	•	•	•	•	•	•	•	•
	Rational TestManager	•	•	•	• ^{TA}	•	• ^{TA}	•	•
	Rational ProjectConsole	•	•	•		•		•	•
Rational Rose		• ^M	• ^E	• ^U	• ^{RT}	• ^{RT/U}		• ^E	
Rational PureCoverage			•	•	•	•	•	•	
Rational Purify			•	•	•	•	•	•	
Rational Quantify			•	•	•	•	•	•	
Rational Robot							•	•	
Rational TestFactory							•	•	
E = Enterprise Edition M = Professional Data Modeler Edition RT = RealTime Edition SF = Rational SoDA for FrameMaker SW = Rational SoDA for Word TA = Used in conjunction with test agents U = UNIX Edition WEB = Can be accessed through Web interface only									

For More Information

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

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

In the next chapter, you will learn more about this tutorial, and you install and set up the files you will use.

This tutorial teaches you the basics of using Rational Suite to plan, design, implement, and test applications. It also points you to additional information about Rational Suite so that you can learn more on your own.

Prerequisites

Before continuing, make sure you have the following software installed on your computer, and a valid license to use each:

- An English edition of Rational Suite, version 2003.06.00.
- Microsoft Internet Explorer 5.5, with Service Pack 1 or 2, or later.
- Microsoft Word 2000 or later.

If any of these prerequisites are *not* met, you can still benefit from reading this tutorial, but you may not be able to perform the exercises.

Determining Which Rational Suite Tools Are Installed

Table 1 shows which tools are included in Rational Suite.

Table 1 Rational Suite Tools

<input type="checkbox"/> Rational ClearCase LT	<input type="checkbox"/> Rational RequisitePro
<input type="checkbox"/> Rational ClearQuest	<input type="checkbox"/> Rational Robot
<input type="checkbox"/> Rational ProjectConsole	<input type="checkbox"/> Rational Rose
<input type="checkbox"/> Rational PureCoverage	<input type="checkbox"/> Rational SoDA for Word
<input type="checkbox"/> Rational Purify	<input type="checkbox"/> Rational TestManager
<input type="checkbox"/> Rational Quantify	<input type="checkbox"/> Rational Unified Process

Exercise: Find out which tools are on your computer.

To determine whether a tool is installed:

- 1 Click **Start**.
- 2 Check to see if the tool's name is in the **Programs > Rational Software** menu.
(From now on, we'll abbreviate these two steps as follows: "**Start > Programs > Rational Software.**")

You may discover that some tools listed in Table 1 are not installed on your computer. For example, a tool may be excluded because of the Suite edition you, or your company, purchased. Or someone in your organization may have chosen not to install certain tools.

- 3 Using Table 1 on page 27, place check marks next to the tools that are installed on your computer.
- 4 Place a bookmark on page 27 so you can refer back to Table 1 later in this tutorial.

ClassicsCD.com: The Tutorial Sample Application

In this tutorial, you implement a small part of a large development project. Using the Rational tools and process, you develop requirements, work with a visual model, and test an application.

Tutorial Background

In this tutorial, you work for *ClassicsCD, Inc.*, a growing online store that sells classical music CDs. Your team is working on Version 2 of the ClassicsCD.com Web site, and uses Rational Suite to plan, design, implement, and test this version of the Visual Basic application. In this tutorial, you add one new enhancement to ClassicsCD.com.

Installing the Tutorial Sample Application and Related Files

Before you perform the tutorial exercises, you must install and set up the files you will use. The *Rational Suite Tutorial* installation requires approximately:

- 10 MB of disk space to download the setup application.
- 65 MB of additional disk space to download the tutorial files.

Note: If you are using Windows NT, make sure you have Administrator privileges so that you can complete the setup successfully.

Exercise: Install the tutorial application and files.

- 1** Start the tutorial installation now: click **Start > Programs > Rational Software > Download Rational Suite Tutorial**.

This path automatically connects you to the Documentation area of the Rational Software Web site, <http://www.rational.com/documentation>.

- 2** Under View Documentation Online (Login required), click **Rational Suite Tutorial** from the drop-down menu.

- 3** Log on, or create your Rational Member Profile.

If, after logging on, you are returned to the Rational Software home page, go to <http://www.rational.com/documentation> to access the Tutorial as you did in Step 2.

- 4** From the Rational Suite Tutorials page, scroll to the section, Version 2003 Tutorial and then click **Rational Suite Tutorial ReadMe File**.

This page provides information and instructions about the tutorial installation and set up process. Use this information to install the tutorial program files and documentation.

- 5** After you read the information, click **Back** to return to the Rational Suite Tutorials page, and if necessary, scroll to the section, Version 2003 Tutorial.

- 6** Click **Rational Suite Tutorial Manual**.

After the PDF file opens, we recommend that you print the book so you can refer to it more easily. You may also want to save the book to your hard disk.

- 7** Return to the Rational Suite Tutorials page, and if necessary, scroll to the section Version 2003 Tutorial.

- 8** Click **Rational Suite Tutorial v2003.06.00 Kit**.

Follow the instructions to install the tutorial files on your computer.

- 9** After you install the files, close your Web browser.

Getting Started

Now you are ready to start your work for *ClassicsCD, Inc.*! To understand how Rational Suite fits into your development environment, we recommend that you work through the entire tutorial. Follow the instructions for each exercise exactly as they are printed, and make sure you complete all exercises in each chapter before attempting to perform the exercises in subsequent chapters. Each series of exercises builds upon the information and skills you learned in previous chapters.

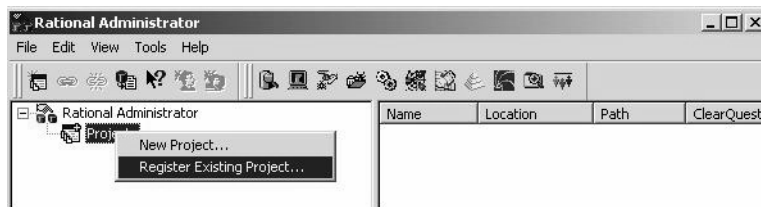
Registering the Project

In a typical development environment, a project administrator sets up the Rational software environment and creates a *Rational project* in the Rational Administrator. Your project administrator uses the Rational Administrator to group a set of artifacts associated with Rational Suite (for example, a RequisitePro database and a Rose model). As a user, you *register* the project to your computer so that you can connect to, and work with, the project on the network. In this tutorial, the Rational project has been created for you.

Exercise: Register the Rational project you will use in this tutorial.

- 1 To start the Rational Administrator, click **Start > Programs > Rational Software > Rational Administrator**.
- 2 In the left pane of the Rational Administrator, right-click **Projects**. From the shortcut menu that appears, click **Register Existing Project** (see Figure 2).

Figure 2 Registering a Project Using Rational Administrator



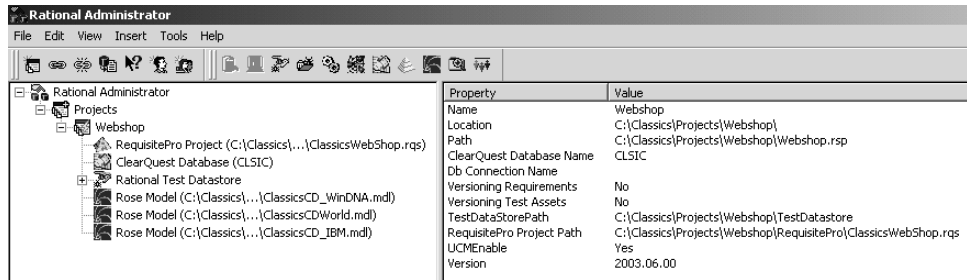
- 3 In the Select Rational Project dialog box, browse to C:\Classics\Projects\Webshop.
- 4 Click Webshop.rsp and then click **Open**.

The Rational Administrator adds **Webshop** under the **Projects** entry in the tree browser. This is the project you will use throughout this tutorial.

- 5 In the left pane of the Rational Administrator, right-click **Webshop** and then click **Connect**.

All assets and development information associated with the Webshop project appear under the **Webshop** entry (see Figure 3), indicating that you have successfully connected to the project.

Figure 3 Connecting to a Project Using Rational Administrator



Keep the Rational Administrator open for the next task.

Associating the Change Request Database with the Project

A Rational project associates software development information collected from Rational tools that are installed on your computer. A Rational project points to development information stored in a database or in a *datastore*, which consists of one or more databases. A project administrator creating a Rational project can associate it with various Rational product datastores, databases, and projects.

After the project administrator associates development assets with a Rational project, you can link data from one database or datastore to another by using individual product features.

In this tutorial, you work with:

- A ClearQuest change request database that contains the project's change requests (defects and enhancement requests).
- A RequisitePro database that contains the project's business and system requirements.
- A Rational Test datastore that contains the project's testing information (test assets, logs, and reports).
- Rational Rose models that contain the project's visual models.

Exercise: Attach the change request database to your project.

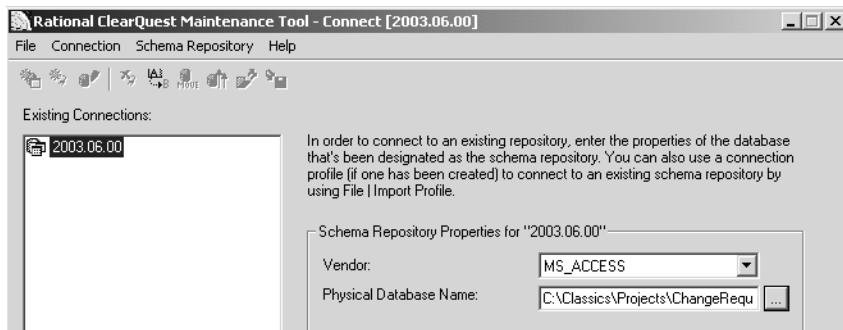
- 1 Check whether Rational ClearQuest is installed on your computer by referring to Table 1 on page 27. If it *is* installed and then you can proceed with this section's instructions. If it is *not* installed, you cannot use ClearQuest during this tutorial.
- 2 From the Rational Administrator menu bar, click **Tools > Rational ClearQuest Maintenance Tool**.
- 3 From the ClearQuest Maintenance Tool menu bar, click **Connection > New**.

Note: The next few steps show you how to connect to a schema repository supplied with this tutorial. If you have been using ClearQuest and a connection to another schema repository, you must reconnect to it after you finish with the tutorial. If you have any questions, contact your project administrator.

- 4 Under Schema Repository Properties, look for "2003.06.00:"
 - a Make sure the value in the **Vendor** box is **MS_ACCESS**.
 - b In the **Physical Database Name** box, click ... (the Browse button) and go to C:\Classics\Projects\ChangeRequests.
 - c Click CQMaster.mdb and then click **Open**.

You have now entered the properties of the change request database to be used in this tutorial (see Figure 4).

Figure 4 Attaching a Change Request Database



- d Click **Finish**.
- 5 Review Status messages to confirm that you have connected the change request database to the project and then click **Done**.
 - 6 Close the ClearQuest Maintenance Tool.

7 Quit the Rational Administrator.



Note: Typically, these tasks are completed by your project administrator. To work with this tutorial, though, it is necessary that you complete these tasks.

Using the Application

In a real-world online store, Web pages would load dynamically, based on information stored in databases, and in reaction to user input. The application you work with during the tutorial, ClassicsCD.com, contains static Web pages. These pages do not change in response to user input. In a typical project, you would create a prototype using static pages and later change to using dynamic pages.

Also, follow the instructions throughout this tutorial exactly. If you diverge from the prescribed path, the application may appear to malfunction. Because the Web pages are static, they do not actually respond to your input.

Exercise: Start ClassicsCD.com and order two CDs.

- 1 To start the application, use Windows Explorer and go to C:\Classics\ClassicsCD_com_sites\v1 and then open index.htm.
Your Web browser displays the first page of ClassicsCD.com.
- 2 On the ClassicsCD.com home page, click **Explore our storefront**.
- 3 On the storefront page, click **Catalog**.
- 4 In the Beethoven section (composers are listed alphabetically), click **Beethoven: Symphonie Nr. 5** to view details of that album.
- 5 On the album's page, click  (the Shopping Cart button) to add the album to your order.
The CD Catalog page reappears.
- 6 At the top of the page, click  next to **Bach: Brandenburg Concertos 1 + 3**.

Exercise: Complete the purchase.

Now complete the purchase and provide feedback to *ClassicsCD, Inc.*

- 1 In the left column of the page, click **Shopping Cart**.
- 2 On the Shopping Cart page in the left column, click **Cashier**.
Before you can complete the order, you must log on.
- 3 In both the **CustomerID** and **Password** boxes, type **jmoore** and then click **Submit**.

If Windows prompts you to remember this password, click **No**.

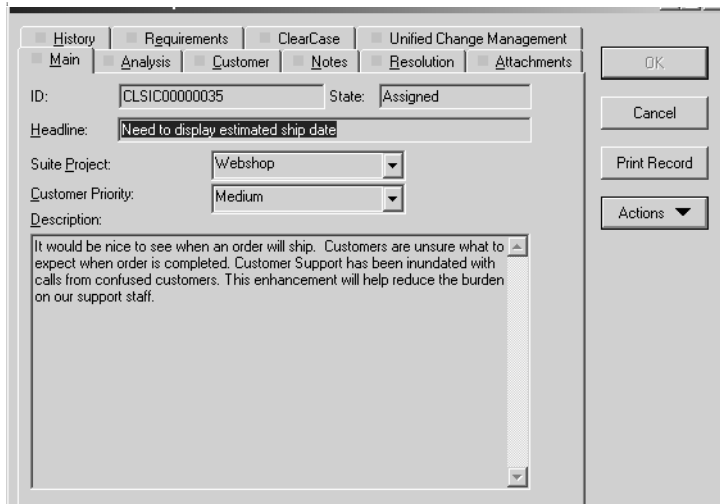
Notice that the Checkout page summarizes your order but does not tell you when the order will ship.

- 4 Scroll to the bottom of the page and then click **Place Order**.
- 5 Provide feedback to the company by clicking **Your Feedback** at the bottom of the page.
- 6 On the feedback form, in the **Dear ClassicsCD.com** box:
 - a Type: **When I place an order, I want to know when my order will ship.**
 - b In the **My e-mail box**, type: **jmoore@clicker.com**.
 - c Click **Send!**.
- 7 Quit ClassicsCD.com, and if necessary, quit Windows Explorer.

Discovering What to Build

At the *ClassicsCD, Inc.* headquarters, someone in marketing received your feedback and entered it into ClearQuest, the tool that manages change requests (see Figure 5).

Figure 5 Viewing Your Enhancement Request Using ClearQuest



The screenshot shows the ClearQuest web interface for viewing an enhancement request. The interface includes a navigation menu at the top with tabs for History, Requirements, ClearCase, and Unified Change Management. Below this, there are sub-tabs for Main, Analysis, Customer, Notes, Resolution, and Attachments. The main content area displays the following information:

- ID: CLSIC00000035
- State: Assigned
- Headline: Need to display estimated ship date
- Suite Project: Webshop
- Customer Priority: Medium
- Description: It would be nice to see when an order will ship. Customers are unsure what to expect when order is completed. Customer Support has been inundated with calls from confused customers. This enhancement will help reduce the burden on our support staff.

On the right side of the interface, there are several buttons: OK, Cancel, Print Record, and an Actions dropdown menu.

In this tutorial, you learn how to implement this enhancement request using Rational Suite. You develop a new requirement, work with a visual model, and perform application testing.

In Chapter 11, *Planning the Next Iteration*, you work with ClearQuest.

How to Use This Tutorial

To understand how Rational Suite fits into your development environment, you can work through the entire tutorial, or read only those chapters most appropriate to your role. If you choose to complete the exercises in the tutorial, it is important that you complete all exercises in each chapter before attempting to perform the exercises in subsequent chapters, because each series of exercises builds upon the information and skills you learned in previous chapters. If you choose only to read this tutorial, we recommend that you start with Chapter 1, *Welcome to Rational Suite*, and this chapter.

Table 2 presents our recommendations if you choose only to read this tutorial.

Table 2 How to Use This Tutorial

Role	Your Main Tasks	Recommended Tutorial Chapters
Analyst	Determine <i>what</i> the system does Represent the user Specify and manage requirements	3 <i>Learning About the Rational Unified Process</i> 4 <i>Managing Change to Project Artifacts</i> 5 <i>Creating Requirements</i> 8 <i>Communicating Project Status</i> 11 <i>Planning the Next Iteration</i>
Developer	Determine <i>how</i> the system works Define architecture Create, modify, manage, and test code	3 <i>Learning About the Rational Unified Process</i> 4 <i>Managing Change to Project Artifacts</i> 5 <i>Creating Requirements</i> 7 <i>Modeling the Enhancement</i> 8 <i>Communicating Project Status</i> 9 <i>Reliability Testing</i> 11 <i>Planning the Next Iteration</i>
Tester	Ensure requirements are met Create, manage, and run tests Report results and verify fixes	3 <i>Learning About the Rational Unified Process</i> 4 <i>Managing Change to Project Artifacts</i> 6 <i>Test Planning</i> 8 <i>Communicating Project Status</i> 9 <i>Reliability Testing</i> 10 <i>Functional Testing</i> 11 <i>Planning the Next Iteration</i>
Manager or Project Leader	Identify and manage project risks Monitor team progress Plan each iteration	3 <i>Learning About the Rational Unified Process</i> 4 <i>Managing Change to Project Artifacts</i> 5 <i>Creating Requirements</i> 6 <i>Test Planning</i> 7 <i>Modeling the Enhancement</i> 8 <i>Communicating Project Status</i> 9 <i>Reliability Testing</i> 10 <i>Functional Testing</i> 11 <i>Planning the Next Iteration</i>

Summary

What You Learned in This Chapter

In this chapter, you learned how:

- To determine which Rational Suite tools are installed on your computer.
- To install and set up the files you will use in the tutorial.
- A Rational project associates software development information collected from Rational Suite tools that are installed on your computer.

What's Next

In this tutorial, you will use the Classics CD Webshop project to implement an enhancement request. The next chapter introduces you to the Rational Unified Process. You use this process to learn about work you will do in subsequent chapters.

Let's get started!

Learning About the Rational Unified Process

3

This chapter introduces you to the Rational Unified Process (RUP). In this chapter, you familiarize yourself with the RUP and read guidelines for the work you will perform in the next chapter.

Audience

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

Getting Your Bearings

In this chapter, you use the Rational Unified Process. To determine whether the RUP is installed on your computer, refer to the tool chart you filled out in Table 1, Rational Suite Tools, on page 27.

If the RUP *is not* installed, you can still benefit from reading this chapter, but you will not be able to perform the exercises.

If the RUP *is* installed, start it now by clicking **Start > Programs > Rational Software > Rational Unified Process**.

You may be prompted to install a version of the Java Plug-in, even if you already have a version installed on your computer. This occurs because the version on your computer differs from the version required by the RUP. If you do not install this particular version of the plug-in, you can still benefit from reading this section, but you will not be able to perform the exercises.

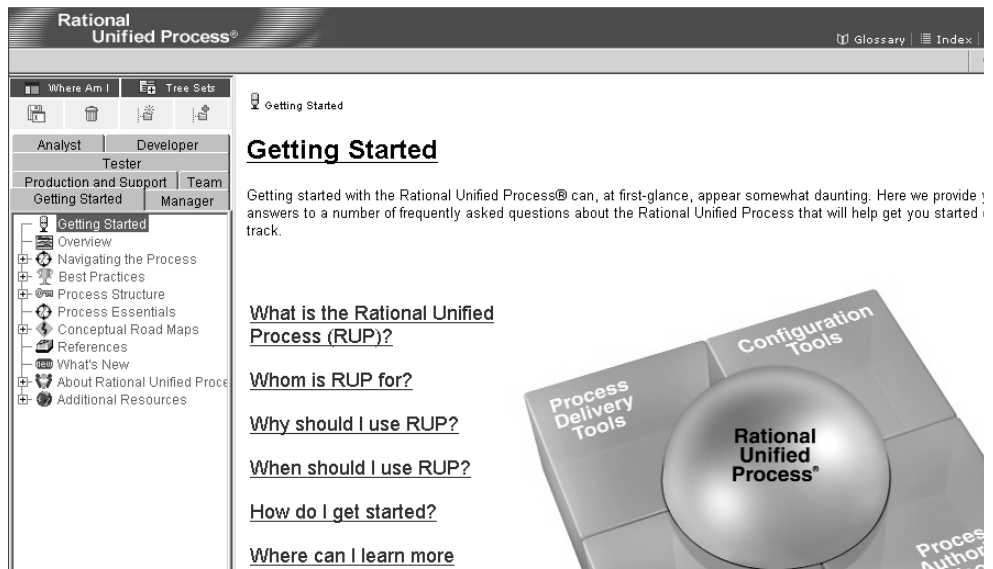
After the correct Java plug-in for the RUP is installed, you may be prompted to trust a signed applet from Rational Software. We suggest that you select **Always** so that you do not see this warning message again.

Then, your Web browser displays the Getting Started page.

In the left pane is the RUP tree browser. The RUP elements are organized in tree sets, named after topics from the software development process. When selected, each tree set is displayed in the tree browser. In the right pane, the RUP displays the view of a selected element. (See Figure 6.)

You can adjust these windows to better view your selection.

Figure 6 The Rational Unified Process Environment



What Is the Rational Unified Process (RUP)?

The RUP is a process framework for developing software that helps you:

- Coordinate the developmental responsibilities of the entire development team.
- Produce high-quality software.
- Meet the needs of your users.
- Work within a set schedule and budget.
- Leverage new technologies.

Many organizations worldwide have successfully used the RUP for both small- and large-scale development efforts because it leads teams through best practices for controlled, iterative software development.

The RUP is implemented as an online guide and knowledge base that you view with a Web browser. It provides guidelines, templates, and Tool Mentors (instructions for applying the guidelines to specific Rational tools) for each phase of the development lifecycle. Additionally, you can customize the RUP to:

- Use only the process components you need for each stage of a project.
- Develop your own process components.
- Exchange best practices with peers and industry leaders.

The Rational Unified Process and Rational Suite

The Rational Unified Process can help you and your team work more effectively. The RUP is a customizable framework providing development teams with a common vision of software development best practices. You can use Rational Process Workbench to adapt the RUP to the specific needs of your projects and your team.

If your company has decided to use Rational Suite without adopting any of the RUP, your projects can still be successful. (You can also use the RUP with projects that do not use Rational Suite or its component tools.)

Even if you do not follow the RUP, you can use it as a source of information about software engineering. For example, it contains topics to help you better understand UML concepts.

This tutorial follows the Rational Unified Process.

Learning the Mechanics

The Rational Unified Process guides you through the full software development lifecycle for each of the following *disciplines*:

- Business Modeling
- Requirements
- Analysis and Design
- Implementation
- Testing
- Deployment
- Configuration and Change Management
- Project Management
- Environment Management.

The RUP provides information and guidance for each of these disciplines applied in a software development project.

Exercise: Learn to navigate within the RUP.

- 1 In the tree browser, select the **Getting Started** tree set.
- 2 In the **Getting Started** tree set, click **Navigating the Process**.

Your Web browser displays the Navigating the Process page, which contains elements of the RUP browser environment.

- 3 Click any area for a brief explanation of how to navigate.
- 4 When you have finished learning about how to navigate the RUP, return to the Navigating the Process page.

The Process at a Glance

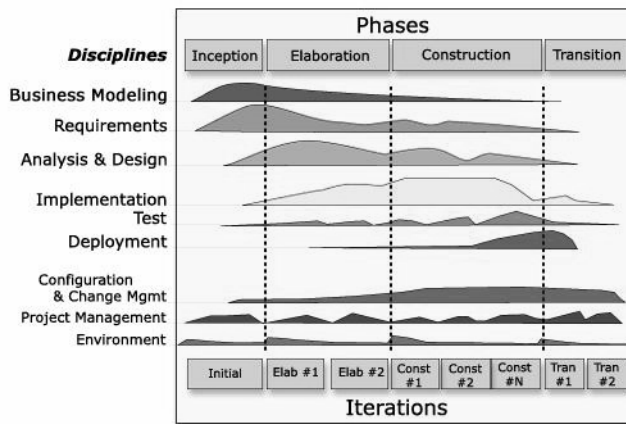
The overview of the Rational Unified Process presents a rich visual representation that can help you better understand the RUP.

Exercise: Get an overview of the Rational Unified Process.

- 1 In the **Getting Started** tree set, click **Overview**.

An overview diagram of the Rational Unified Process appears (see Figure 7).

Figure 7 Rational Unified Process Overview



The diagram in Figure 7 illustrates 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 primary software development *roles* (analysts, developers, testers, managers) perform activities in one or more disciplines. The diagram shows how emphasis on different disciplines varies with each iteration.

Notice, for example, that most of the work associated with requirements happens early in the development cycle, but continues throughout a project. Testing, however, can start early in the project, but typically intensifies at the end of construction.

The RUP Overview outlines the following important concepts:

- The software lifecycle of the Rational Unified Process is decomposed over time into four sequential phases, each concluded by a major milestone. Each phase is, essentially, a span of time between two major milestones.
 - A development iteration is one complete pass through all the disciplines, leading to a product release.
- 2 Optionally, click the terms **Phases** and **Iterations** in the Overview diagram to learn more.

When you finish reading about these concepts, click **Back** in your Web browser to return to the Rational Unified Process: Overview page.

- 3 Click other elements in the **Getting Started** tree set and read about other aspects of the RUP.
- 4 Leave the RUP open.

Key Concepts

The Rational Unified Process provides a quick summary of its most important components.

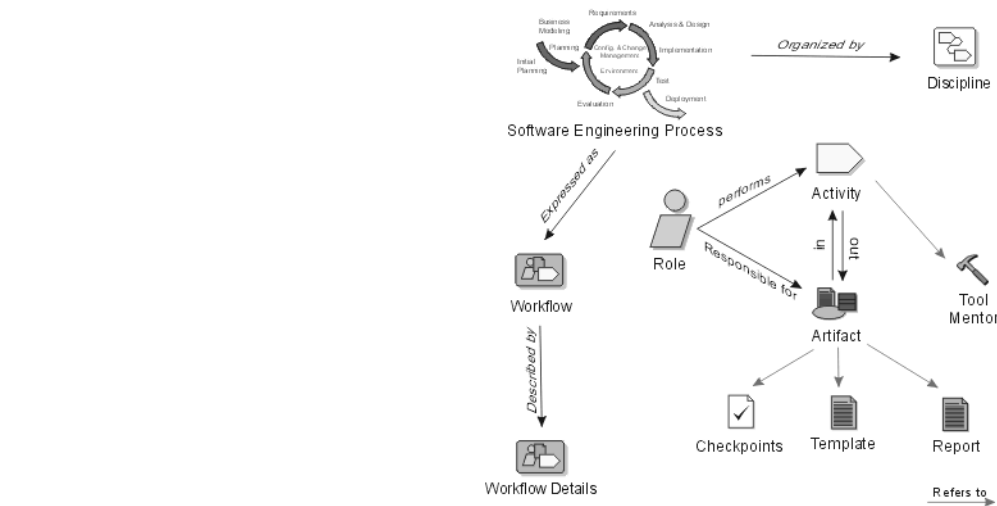
Exercise: Learn about key concepts.

- 1 In the **Getting Started** tree set, click **Process Structure**.

A new Web page appears. The diagram at the top of the page shows the relationships among key concepts of the RUP (see Figure 8).

Figure 8 Key Concepts in the Rational Unified Process

The Basic Elements of RUP ☺



2 Click a symbol in the graphic to learn more about that concept (excerpts follow).

- A *discipline* shows all activities that produce a particular set of software assets. The RUP describes development disciplines at an overview level—including a summary of all roles, workflows, activities, and artifacts that are involved.
- A *role* is defined as the behavior and responsibilities of an individual or a group of individuals on a project team. One person can act in the capacity of several roles over the course of a project. Conversely, many people can act in the capacity of a single role in a project. Roles are responsible for creating artifacts.
- A *workflow* is the sequence of activities that workers perform toward a common goal. A *workflow diagram* serves as a high-level map for a set of related activities. The arrows between activities represent the typical, though not required, flow of work between activities.
- 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.
- An *artifact* is something a role produces as the result of performing an activity. In the RUP, 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.

- 3 Below the diagram, scroll to the first section, *Software Engineering Process*, and read it for a quick summary of the Rational Unified Process.

Exploring a Workflow

Exercise: Explore the Requirements discipline.

The RUP provides guidance on how to enhance existing systems. During this tutorial, you use the RUP guidelines to work on refining the online store application, ClassicsCD.com.

Learn about the workflow details for Requirements:

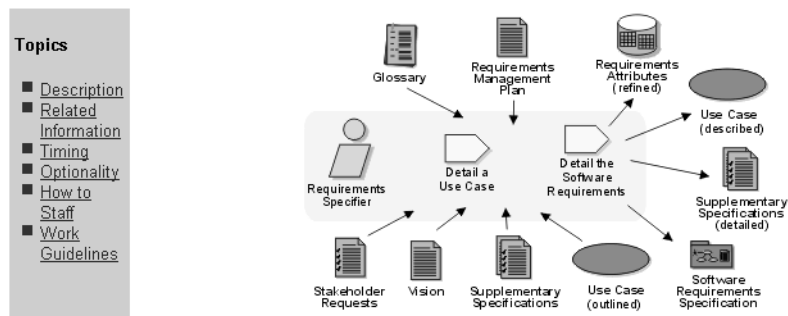
- 1 In the tree browser, select the **Analyst** tree set.
- 2 In the **Analyst** tree set, click **Requirements > Workflow** to display the Requirements: Workflow page.
- 3 On the Requirements: Workflow page, click **Refine the System Definition** to display workflow details (see Figure 9).

Workflow details (like those in Figure 9) represent the roles involved, the artifacts used as input, the resulting artifacts, and the activities that make up this part of the overall workflow. For more information about any of these elements, click that element within the details page.

Figure 9 Requirements Workflow Details in the RUP

Workflow Detail: Refine the System Definition

The purpose of this workflow detail is to further refine the requirements in order to capture the consensus understanding of the system definition.



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 or features 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 RUP and in the UML. They enhance communication across development teams so that you can solve the right problem and define the right system for your users.

Exercise: Learn about use cases.

- 1 Within the Requirements Workflow detail, click **Detail a Use Case**.

The Rational Unified Process displays a page describing 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.

- 2 To learn more about use cases, including how to write them, in the Resulting Artifacts section, click **Use Case**.

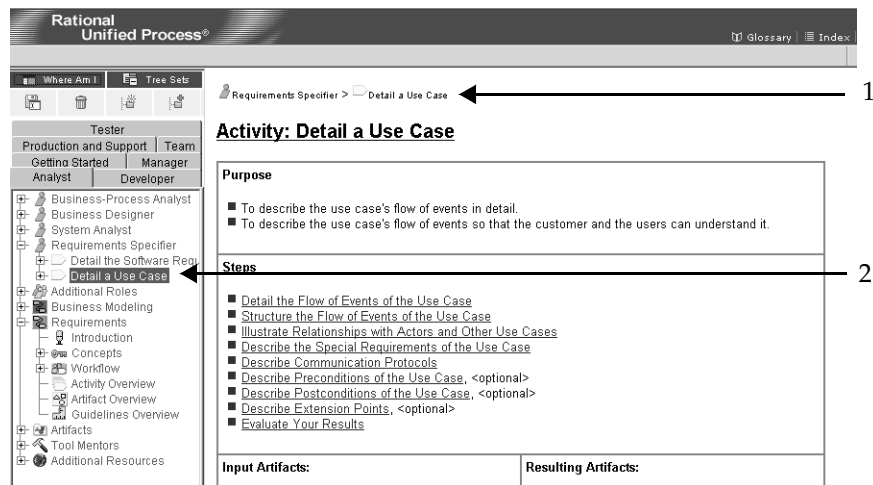
This new page provides a good overview of use cases as artifacts, including a description of how they're used, an outline of a typical use case, and responsible parties.

On this page, there's a link to a use case template. When you create use cases, we recommend that you use this template, or another template that your group has designed, to ensure consistency and completeness in use case development. This makes it easy for all stakeholders to locate and understand important project information.

Review the templates; you will work with a use case document later on in this tutorial.

- 3 Click **Back** in your Web browser to return to the Activity: Detail a Use Case page.
- 4 At this point, you might want to understand where you are in the RUP hierarchy. Do one of the following:
 - Look at the top of the displayed page to see the RUP visual that shows where you are (see Figure 10, “1”). Use this feature anytime you want to know the location of the RUP page that is displayed.
 - Or, click **Where am I** in the tree browser. The tree browser updates itself to show your location. Currently, you are within the **Analyst** tree set – **Requirements Specifier > Detail a Use Case** (see Figure 10, “2”).

Figure 10 Knowing Where You Are in the RUP



For more information on use cases, see Chapter 5, *Creating Requirements*.

Tool Mentors: Using Rational Tools to Support the Process

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 steps in the process, or produce a particular artifact or report, using one or more tools.

In the next exercise, you use a Tool Mentor to get instructions for the work you will do first: defining use cases.

Exercise: Work with a Tool Mentor.

Read a Tool Mentor to see how the RUP integrates with Rational tools.

- 1 On the Activity: Detail a Use Case page, scroll down and find the Tool Mentors section in the table.

- 2 Click **Detailing a Use Case Using Rational RequisitePro**.

The RUP displays the Tool Mentor, showing a statement of purpose, an overview, and a series of steps to use the tool.

- 3 Under Tool Steps, scroll to and then click Step 3, **Create requirements in the detailed Use-Case Specification**.

Review the instructions; you will perform a subset of these steps in this tutorial.

Learning the Basics

The Rational Unified Process provides guidance for performing certain kinds of work, including the work you will do in this tutorial with ClassicsCD.com.

Exercise: Learn about Process Essentials.

- 1 In the **Getting Started** tree set, click **Process Essentials**.
- 2 Read the topics to learn about essential elements of the RUP.

Exercise: Learn about developing applications for the Web.

- 1 In the **Getting Started** tree set, go to **Conceptual Road Maps > Developing e-business Solutions**.
- 2 Read the topics to learn more about developing Web applications.

Summary

For More Information

To learn more about the Rational Unified Process, use the tree browser to select and read the topics of interest.

Cleaning Up

Either quit the RUP, or minimize it and use the RUP as a supplement to learn more about topics covered later on in this tutorial.

What You Learned in This Chapter

In this chapter, you learned that:

- The Rational Unified Process provides customizable guidelines for best practices for software development. In Rational Suite, the RUP is recommended, but optional.
- A workflow describes a set of related activities focused on meeting a goal. For each activity, a role uses artifacts created in previous activities and produces other artifacts.
- Early in the requirements phase, 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 explicit instructions for performing a RUP activity using the appropriate Rational tool.

What's Next

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

Managing Change to Project Artifacts

4

In this chapter, you learn about the Rational approach to managing change during software development using Rational ClearCase LT and Unified Change Management (UCM). When used together, ClearCase LT and UCM help you successfully manage changing project artifacts from requirements to release, through multiple iterations.

Audience

This chapter applies to all team members.

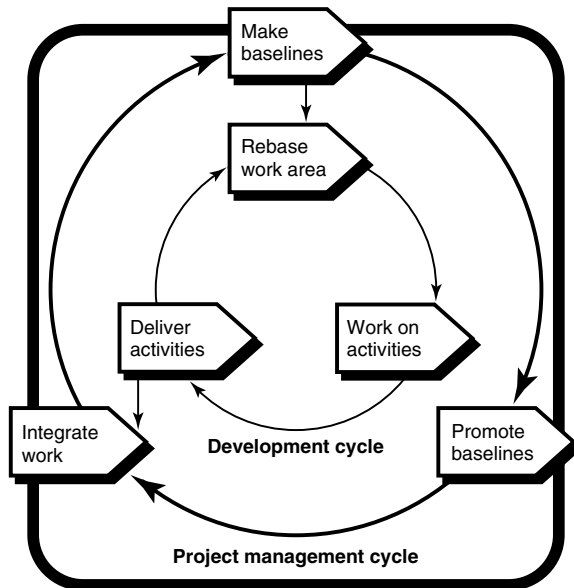
What Is Unified Change Management?

Rational Software offers *Unified Change Management (UCM)* as the best approach for managing change during software system development from requirements to release. UCM focuses on these guiding concepts:

- A UCM *activity* represents the work required to complete a development task. UCM 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 as the result of performing an activity. In the Rational Unified Process (RUP), the artifacts produced in one activity are often used as input into other activities. Conceptually, artifacts can be requirements, visual models, test cases, source code, documentation, 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.

The key strength of UCM 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. Figure 11 shows a typical way to manage change using UCM.

Figure 11 Typical UCM Workflow



UCM Tools

A key aspect of the UCM model is that it unifies the activities used to plan and track project progress and the artifacts undergoing change. The UCM model is realized by both process and tools. Rational ClearQuest and Rational ClearCase LT provide tool support for UCM. For example, ClearQuest manages the project's tasks, defects, and requests for enhancements (referred to generically as *activities*), and ClearCase LT manages the artifacts produced by a software project. When used together, these tools help your software team better manage changing requirements and development complexity throughout the software development lifecycle.

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 the built-in UCM development process to track and manage changes to all software project files, including requirements, visual models, and source code.

Using the Tools with UCM – ClearQuest and ClearCase LT

As described in the RUP, you typically use UCM with ClearQuest and ClearCase LT as follows:

- 1 [one time] A project manager or administrator installs Rational software. This individual sets up the ClearQuest and ClearCase LT environments, and creates a Rational project. (A *Rational project* associates the data created and maintained by Rational tools and enables integrations among them.) A project manager also creates a UCM project to associate with the Rational project.
- 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 includes an *integration stream*; you can create a companion *integration view* for your own use.

You use a *view* to select one version of each element in your workspace. In UCM, your *stream* provides these configuration instructions to the view and tracks your activities. When you join a project, UCM automatically configures your stream so that you see the right version for your workspace (see Figure 12).

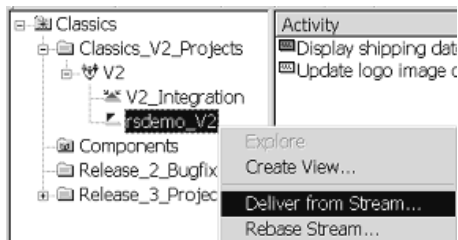
Figure 12 Exploring ClearCase LT Projects Using UCM



- 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. You use ClearCase LT in your private 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 in the artifacts. 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 After you finish work on the activity, you *deliver* changes for the entire activity (see Figure 13). 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 can also close the activity, depending on the policies the project manager has established.

Figure 13 Delivering Changes to the UCM Project Using ClearCase LT



- 7 After developers deliver a set of activities, the project manager makes a *baseline*, a new common starting place for all developers that includes the new activities or modified artifacts. On your project, a new baseline can be created regularly, perhaps even daily.
- 8 If the changes in the baseline are approved (through testing or through another review process), the project manager *promotes* it, making it the recommended baseline.
- 9 You *rebase* your development stream (work area) so that when you work on your next activity, you start 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, your team 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.

Unifying Code and Content for Web Development

As software applications and Web sites grow in size, complexity, and strategic value, so does the need to control changes to them. Beyond these common characteristics that Web-based and traditional software development applications share, there are important distinctions to make between. For example, in Web applications:

- Change happens at a considerably faster pace.
- Many stakeholders in both technical and non-technical roles contribute to Web sites.
- The frequency of change and the broader spectrum of stakeholders increases the likelihood of error.

Although most of this tutorial discusses development concepts for traditional software projects, many of the ideas, tools, and processes presented can also be applied to Web applications.

Summary

For More Information

To learn how to use ClearCase LT, start with the ClearCase LT tutorial. Go to **Start > Programs > Rational Software > Rational ClearCase > Tutorial**. In the Help Topics window, double click **Tutorial** and then double click **Rational ClearCase Tutorial**.

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 the *Rational ClearCase LT Introduction*.

What You Learned in This Chapter

In this chapter, you learned:

- 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 activities. It offers charting and reporting features to track and communicate project progress to all stakeholders.
- Under UCM, team members use ClearCase LT to manage artifacts under source control, they work on activities in their personal development workspaces, and deliver modified artifacts to the integration stream after 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, you will work on the Arrange Shipment use case for the enhancement request to ClassicsCD.com.

In this chapter, you use Rational RequisitePro and Rational Rose to create a use case for the enhancement you are implementing.

Audience

This chapter applies most directly to analysts, but is relevant for all team members.

Getting Your Bearings

In this chapter, you use RequisitePro and Rose. To determine whether these tools are installed on your computer, refer to the tool chart you filled out in Table 1, Rational Suite Tools, on page 27.

If they are *not* installed, you can still benefit from reading this chapter, but you will not be able to perform some of the exercises.

If they *are* installed, start RequisitePro now as follows (you open Rose later in this chapter):

Click **Start > Programs > Rational Software > Rational RequisitePro**. RequisitePro starts and the Open Project window appears. If the Let's Go RequisitePro window also appears, click **Close**.

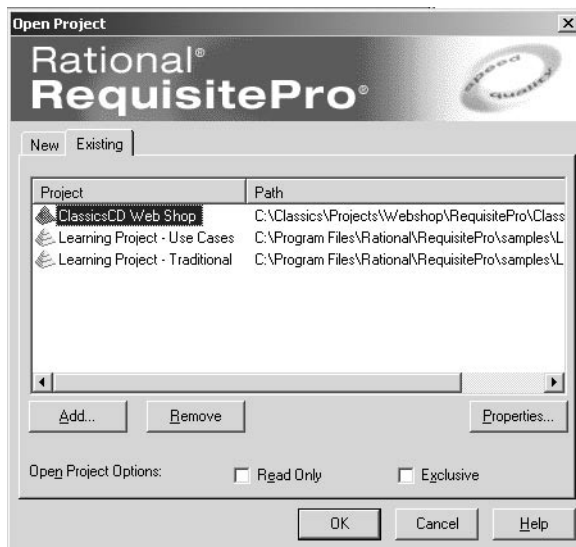
Note: If this is your first time starting RequisitePro, the Exploring Rational RequisitePro page appears. Read this page to learn about the RequisitePro user interface. You may want to print this page to keep as quick reference. Then close the window so you can select a RequisitePro project to work with.

Exercise: Open the ClassicsCD.com Webshop Project.

- 1 In the Open Project window, click **Add**.
- 2 From the Add Project window, go to C:\Classics\Projects\Webshop\RequisitePro.
- 3 Click ClassicsWebShop.rqs and then click **Open**.

The RequisitePro database associated with the ClassicsCD Web Shop project appear in the list of Existing projects (see Figure 14).

Figure 14 Opening the ClassicsCD Web Shop Project Using RequisitePro



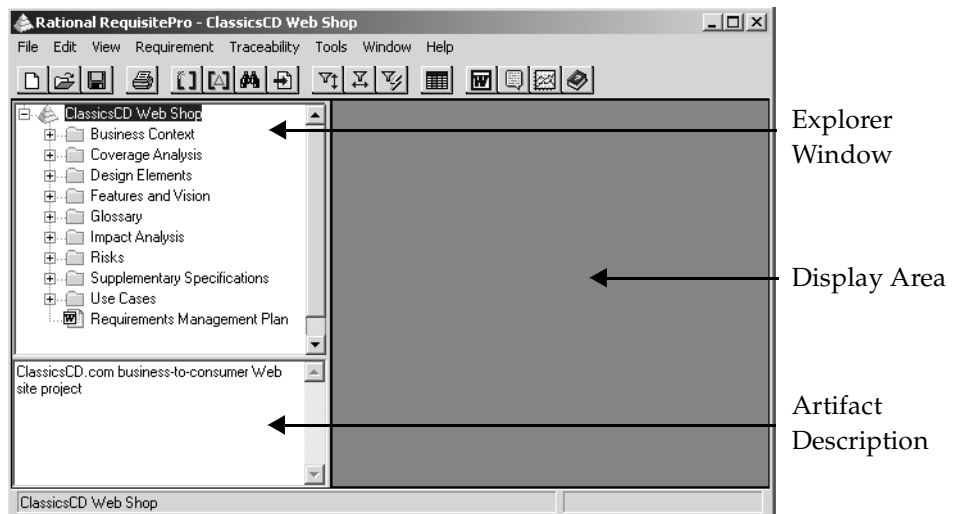
- 4 Click **OK**. The Project Logon dialog box appears.
- 5 In both the **Username** and **Password** boxes, type **pat** and then click **OK**.

Note: Throughout this tutorial, we will use **pat** for the Rational Logon username and password.

RequisitePro opens the project and displays project artifacts in a hierarchical tree browser in the left pane (the Explorer window). A description of the artifact selected in the Explorer window appears at the bottom. In the right pane, RequisitePro displays the view of a selected artifact. (see Figure 15).

You can adjust these windows to better view your selection.

Figure 15 Working with RequisitePro



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 your stakeholders — everyone who has an interest in your project. To determine those needs, you interview users and other stakeholders, analyze enhancement requests, and work with project team members. You then decide which of those needs will become project requirements.

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 requirements.

Requirements management is challenging because requirements change throughout a project. For example, users can change their minds about essential features, or they may not have articulated their wishes clearly in the first place. Competitors can release new versions of their software and you must respond by changing project plans midstream. Changing laws can affect your software. When you don't manage requirements, feature creep can slow down and complicate your project.

Using RequisitePro

RequisitePro makes it easy to write and manage requirements. It is integrated with Microsoft Word and is packaged with Word templates to help you 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 architecture and write more detailed specifications.
- *Testers* use requirements to design tests and check test coverage.
- *Project leaders* and *managers* use RequisitePro to plan project work based on available resources (for example, time, budget, and personnel).

Starting with a Use Case

In Chapter 2, *About This Tutorial*, you saw the enhancement request that was entered in response to your feedback. One of your team members has started work on the use case corresponding to the enhancement request.

Why Work with Use Cases?

Use cases describe system behavior in a common language that everyone on the team can understand. Working with use cases is a key unifying mechanism in the Rational Unified Process (RUP).

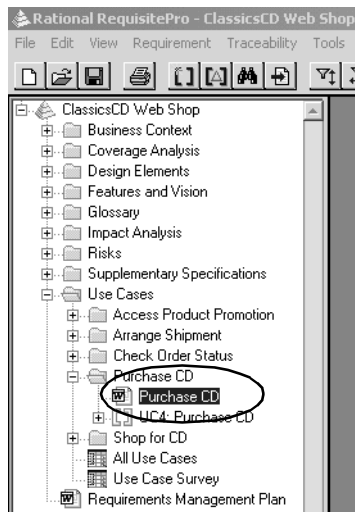
Use cases are important to everyone on the project:

- *Analysts* use them to express how the system should behave and to verify planned changes with stakeholders.
- *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.
- *System testers* use them to validate system behavior starting as early as the design phase.
- *Project leaders* and *managers* use them to formally verify that the results of requirements conform to the customer's view of the system.

Exercise: Open the use case document.

- 1 In the Explorer window, go to **ClassicsCD Web Shop > Use Cases > Purchase CD > Purchase CD** and then double-click this entry (see Figure 16).

Figure 16 Opening a Use Case Document with RequisitePro



The RequisitePro Word document appears and displays the Purchase CD use case. This document is based on a template provided with the RUP.

In Chapter 3, you saw, and may have clicked on, a link to an outline of this template. In the RUP, this outline can be found in the **Analyst** tree set: **Artifacts > Requirements Artifact Set > Use-Case Model > Use Case > Use Case Specification (Informal)**.

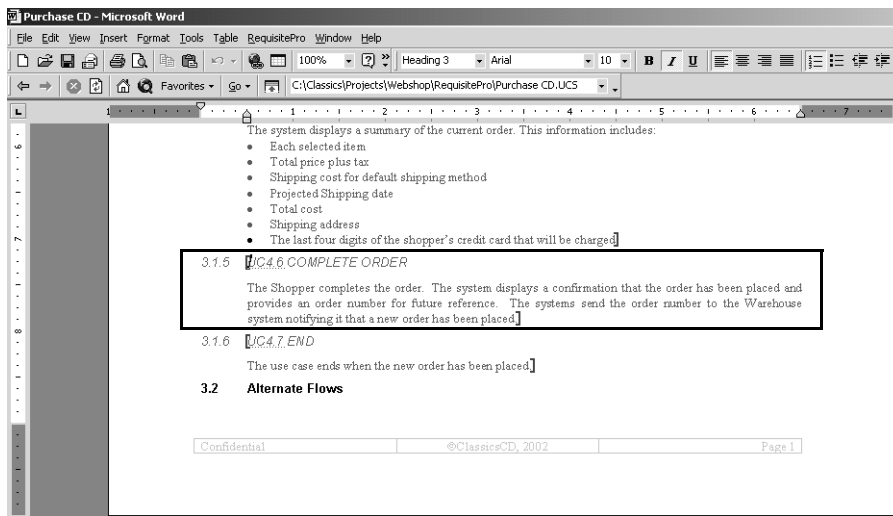
- 2 In the Table of Contents, go to Section 3, Flow of Events. Locate Section 3.1.5 Complete Order. Click the corresponding page number.

Note: Depending on which version of Microsoft Word is installed on your computer, you may need to press CTRL to activate the hyperlink to the page.

Microsoft Word automatically takes you to the corresponding use case specification: UC4.6 Complete Order (see Figure 17).

Capitalized text that also appears in teal colored, italicized font identifies *use case requirements*. These use case requirements (identified by the prefix UC) are high-level requirements that describe the system's behavior.

Figure 17 Working with Use Case Documents Using RequisitePro



- 3 Read the Brief Description and Flow of Events. Notice that when a shopper completes an order, it is not part of the original use case to provide the expected shipping date. In subsequent exercises, you will identify this feature as a requirement.

This is a typical way of starting requirements work. You use the familiar environment of Word to document your requirements. You use RequisitePro to identify and elaborate on your project requirements. You also indicate which requirements are related. RequisitePro then tracks how changes to the system affect your requirements and how changes to requirements affect your system.

- 4 If Rose *is not* installed on your computer, minimize Word and RequisitePro. If Rose *is* installed, quit Word and then click **No** if prompted to save your changes. Also quit RequisitePro. If you are prompted to close the project, click **Yes**.

How Does RequisitePro Handle Requirements?

RequisitePro is both document-centric and database-centric and relies on the strengths of the following:

- The *document* features provide a familiar environment (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 working in a RequisitePro Word document.
- 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 RequisitePro Views.

In the use case document, the requirements text (with double-underlined characters by default) exists in the document. The database also stores the requirements text, along with attributes (such as *priority* and *assigned-to*) that help track the requirement. Later in this chapter, you will work with RequisitePro database features.

Learning More About Use Cases

You frequently start requirements work by developing use cases. When working with use cases, you work in Rose to incorporate the use case in your visual model and then work in RequisitePro to add textual descriptions, attributes, and links.

The RUP describes how to write a use case. It includes details about the artifacts that you 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. You can use it as a starting point for defining use case requirements. 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.

Continuing Use Case Work Using Rose

Rational Rose helps analysts visualize the behavior of a system through *use case diagrams*. These diagrams help you manage complexity because they allow you to see the “big picture.” 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 between use cases and actors.

By using Rose to create use case diagrams, you provide a centralized, graphic representation of the system's use cases. This helps all stakeholders share a common understanding of the project 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 UML, an easily understood, industry-standard language for designing software. For example, analysts use Rose to create use case diagrams that describe a system at a high-level. Later on, you will see how architects continue this work by using Rose to design the system in more detail. As a result, your system diagram, architecture, and data are managed by one tool, Rational Rose, and with one language, UML.

Working with a Use Case Diagram

In this section, you continue work on the Purchase CD use case for ClassicsCD.com as the first step in implementing the enhancement.

Exercise: Start Rose.

- 1 From the **Start** menu, click **Programs > Rational Software > Rational Rose**.
Rose starts and the Create New Model dialog box appears. Make sure that the check box **Don't show this dialog in the future** is cleared so you can easily open the Rose model later on in this tutorial.
- 2 Click **Cancel** to close the Create New Model dialog box.
- 3 Rational Rose lets you work with models moved or copied among workspaces. To do this, you need to configure Rose:
 - a From the Rose menu bar, click **File > Edit Path Map**.
The Virtual Path Map window appears.
 - b In the **Virtual Symbol to Actual Path Mapping** list, look for \$CURDIR. If you see this symbol, close the Virtual Path Map window and proceed to Step 4.
 - c In the **Symbol** box, type CURDIR.
 - d In the **Actual Path** box, type &.
 - e Click **Add**.
 - f Click **Close**.

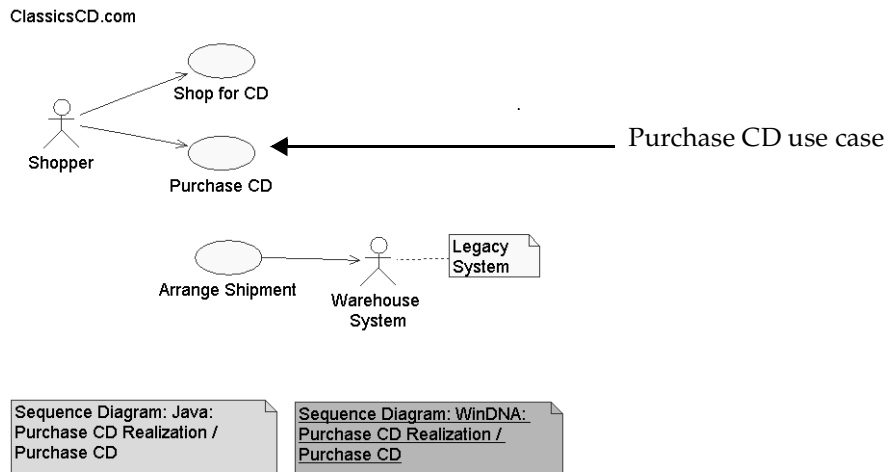
- 4 From the Rose menu bar, click **File > Open** and go to C:\Classics\Projects\Webshop\Rose.
- 5 Click ClassicsCD_WinDNA.mdl and then click **Open**.
- 6 Rose prompts you to load subunits. Click **Yes**.

Rose displays a hierarchical tree browser in the upper-left pane (the Rose browser). In the right pane (the diagram window), it displays the logical view of the architecture showing how the top-level packages in the system interact. If necessary, maximize the diagram window.

Exercise: View the use case diagram.

- 1 In the Rose browser, go to **Use Case View > System Models > Main**, and double-click **Main** to display the use case diagram (see Figure 18). If necessary, maximize the diagram window within Rose.

Figure 18 Viewing the ClassicsCD.com Use Case Diagram Using Rose



Associating the Rose Model with the RequisitePro Project

Earlier in this chapter, you opened the ClassicsCD.com Webshop project in RequisitePro. In this section, you learn how to view the association between the ClassicsCD.com Rose model with that project in RequisitePro. You also create the requirement and link the use case with a requirement.

Exercise: View the association between the Rose model and the RequisitePro project.

- 1 On the use case diagram, right-click the **Purchase CD** use case object. From the shortcut menu, click **View RequisitePro Association**.

If a login box appears type **pat** in both the **Username** and **Password** boxes and then click **OK**.

The RequisitePro Association dialog box appears and displays information about the Purchase CD use-case document and the Purchase CD requirement you saw earlier in this chapter.

- 2 After you finish reviewing the use case's RequisitePro association, click **OK**.

The Purchase CD use case is represented both in text and by a visual model. The use case is a single element because of the integration between Rose and RequisitePro.

Creating a New Requirement

Purchase CD is an established use case, but you still need to identify a requirement corresponding to the enhancement request to display an estimated ship date for a customer's order.

Exercise: Create the requirement.

- 1 If Rose *is not* installed on your computer, maximize Word and continue with Step 2. If Rose *is* installed, continue working with this Rose model: right-click the **Purchase CD** use case. From the shortcut menu, click **Use Case Document > Open**.

The RequisitePro Word document appears and you can work with the use case.

- 2 In the document, scroll to the use case specification: UC4.6 Complete Order. At the end of the second sentence, insert the following text before the period: "**as well as an expected ship date for each CD.**"

The second sentence should now read: "The shopper completes the order. The system displays a confirmation that the order has been placed and provides an order number for future reference as well as an expected ship date for each CD."

- 3 From the Word menu bar, click **RequisitePro > Document > Save**.

RequisitePro prompts you to enter a change description for UC4.6. Enter the following text “**Modified to correspond with the enhancement request to display an estimated ship date for customer orders.**” Then click **OK**.

RequisitePro saves the document and updates the requirement.

Exercise: Learn about requirement properties

- 1 From the Word menu bar, click **RequisitePro > Requirement > Properties**. The Requirement Properties dialog box appears.
- 2 Review the information about this requirement from the General tab.
- 3 Go to the Attributes tab. Review the requirement attributes to learn more about the Complete Order requirement, such as the priority and status.
- 4 Go to the Hierarchy tab. From this tab you learn that the Complete Order requirement that you modified to include the enhancement request for this tutorial is part of the Purchase CD use case.
- 5 When you have finished reviewing the properties of the requirement, click **OK**.

Looking at Requirements in the Database

In this section, you use the RequisitePro database to view other requirements related to the enhancement you are working on. Whenever you work in a database, you use a view, which filters data in a specific format. RequisitePro works the same way.

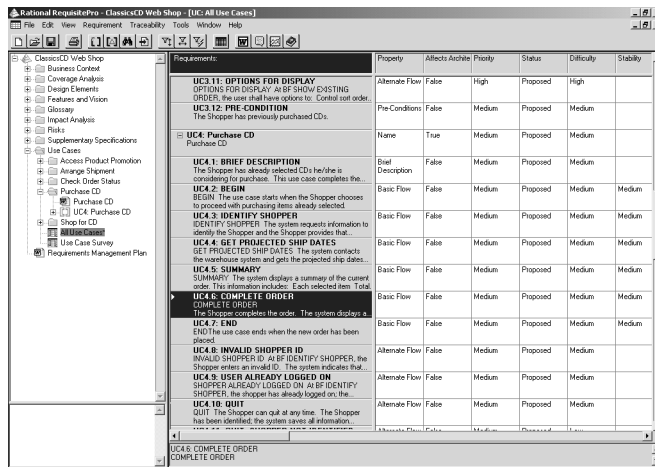
Exercise: View requirements using the RequisitePro database.

- 1 In the Word document, click **RequisitePro > Show RequisitePro Explorer**. RequisitePro appears.
- 2 In the Explorer window, expand **ClassicsCD Web Shop > Use Cases** and then double-click **All Use Cases**.

The use cases are organized hierarchically and each describes functional areas of ClassicsCD.com. Child requirement use cases are listed under their respective parent. Parent requirements are general, while child requirements describe specific areas. To see children of a use case, you may need to expand the parent requirement by clicking the **+** next to the use case’s name.

- 3 Scroll down to UC4.6: Complete Order. Review the requirement properties in this view (see Figure 19).

Figure 19 Viewing Requirements Using RequisitePro



- 4 To see more of a requirement's attributes and properties, scroll to the right.

Linking to Another Requirement

So far, you have:

- Decided to implement a new enhancement.
- In Rose, reviewed the use case diagram.
- Linked the model to the RequisitePro project.
- Created a new use case requirement.
- Added values to the requirement's attributes.

You now want to make sure the use case requirement is linked to another type of requirement, a *feature* requirement. A system's feature requirements are written at a very high-level and form a foundation for the entire system.

Exercise: View the relationship between a use case requirement and a feature requirement.

- 1 In the RequisitePro Explorer window, go to **ClassicsCD Web Shop > Coverage Analysis** and then double-click **Use Cases to Features** relationships.

RequisitePro displays this view, showing the entire hierarchy of requirements. To see more details, you may need to move the horizontal and vertical dividers of the matrix.


- 2 Scroll to UC4 and, if necessary, expand it to see the child requirements.

An arrow symbol appears in the cell, showing the relationship between the Purchase CD use-case requirement (including all of its child requirements, including the requirement you just modified) and the feature requirement, Feature1: Secure Payment Method. You will learn more about traceability in the next section.

- 3 Browse this view of the requirements using the scroll bars to learn more about the relationships between use cases and features.

Traceability and Suspect Links

The matrix in the RequisitePro View shows some of the links between requirements. These links describe dependencies between requirements.

An arrow with a line through it () indicates that the link is *suspect*. A link becomes suspect after a requirement in the link relationship changes. The suspect link alerts project teams that a requirement has changed. Analysts use this information to:

- Asses the impact of the change to the project.
- Manage the requirements more effectively.
- Work with managers and project leaders to more effectively manage the scope of the project.

Other Requirement Types

So far, we've discussed high-level feature requirements and more detailed use case requirements. Some requirements do not lend themselves to use cases, so RequisitePro supports other types of requirements. For example, you can define supplemental requirements for performance targets and platform support. Additional requirement types include design requirements, business needs, and glossary requirements.

You can also define new requirement types. 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, 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 finished.

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

Extended Help

Extended Help is a powerful feature of Rational Suite that provides information about the higher-level tasks you may want to accomplish. It gives you direct access to the RUP from Rational Suite tools. In addition, you can add your own organizational guidelines or standards to Extended Help.

Exercise: View Extended Help.

- 1 From the RequisitePro menu bar, click **Help > Extended Help**.

The Extended Help: Overview window appears within the RUP. An Extended Help tree set for this tool also appears.

- 2 In the **Extended Help - ReqPro** tree set, left pane, towards the bottom of the list, click **More Content > Tool Mentor: Managing Use Cases Using Rational Rose and Rational RequisitePro**.

Read this Tool Mentor to review the work you did in this chapter.

You can learn more about Extended Help from any Rational tool. From the tool's menu bar, click **Help > Extended Help**. The RUP displays Extended Help topics that are related to the Suite tool you are using.

Summary

For More Information

For more information on using RequisitePro, start with the tutorial. From the RequisitePro menu bar, click **Help > Tutorial**.

For more information about Rose, see *Modeling the Enhancement* on page 85.

Cleaning Up

Quit Extended Help.

Quit RequisitePro by clicking **File > Exit** from the RequisitePro menu bar. When RequisitePro asks if you're sure you want to close the project, click **Yes**. If prompted to save changes to any views, click **No**.

If necessary, quit Rose by choosing **File > Exit** from the Rose menu bar. If Rose prompts you to save changes, click **No**.

What You Learned in This Chapter

In this chapter, you learned:

- A requirement is a condition or capability to which the system must conform.
- Managing requirements is a systematic approach to finding, documenting, organizing, and tracking system features and attributes.
- RequisitePro helps you manage requirements and multiple requirement types.
- RequisitePro is both document-centric and database-centric, allowing your team to benefit from the strengths of both.
- When working with use cases, you work in Rose to incorporate the use case into your visual model and then work in RequisitePro to add textual descriptions, attributes, and links.
- Extended Help gives you immediate access to the RUP and information about using Rational Suite tools. You can add your own information to Extended Help.

What's Next

In the next chapter, you use the requirements identified in this chapter to get started on test planning.

So far, you have defined requirements for the ClassicsCD.com enhancement. You have not yet modeled or implemented code. However, you are ready to start test planning with Rational TestManager.

Audience

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

Getting Your Bearings

In this chapter, you use Rational TestManager. To determine whether TestManager is installed on your computer, refer to the tool chart you filled out in Table 1, Rational Suite Tools, on page 27.

If TestManager is *not* installed, you can still benefit from reading this chapter, but you will not be able to perform the exercises.

If TestManager *is* installed, start it now by clicking **Start > Programs > Rational Software > Rational TestManager**. The Rational Test Login dialog box appears.

Exercise: Open the ClassicsCD.com Webshop Project

In the Rational Test Login dialog box, use the following values:

- 1 In the **User Name** and **Password** boxes, type **pat**.
- 2 Make sure the **Project** box displays **Webshop**.
- 3 Make sure the **Location** box displays **C:\Classics\Projects\Webshop\Webshop.rsp**.
- 4 Click **OK**.

TestManager opens the Webshop project. Make sure the Planning tab at the bottom of the left pane (the *Test Asset Workspace*) is selected. You can now work with the project.

What Is Test Planning?

Test planning allows your team to effectively measure and manage test efforts over the course of the project. During test planning, you identify the types of tests to 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 must 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 can also use test inputs to help you define *test configurations*, attributes referring to a computer’s hardware and software. These attributes can apply to your test cases.

As with development, test planning is an iterative process. You continue to plan testing throughout a project 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 correctly.
- 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

You can use all types of project artifacts to plan, design, and run tests with Rational TestManager. You can use requirements, visual models, and source code to create a test plan so that you can test all aspects of your system, including product features, system architecture, and code.

The integration between Rational tools enables sharing of project assets to help 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.
- Helps team members share information about the testing progress.
- Helps you track how many tests have been planned, scripted, and carried out.
- 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

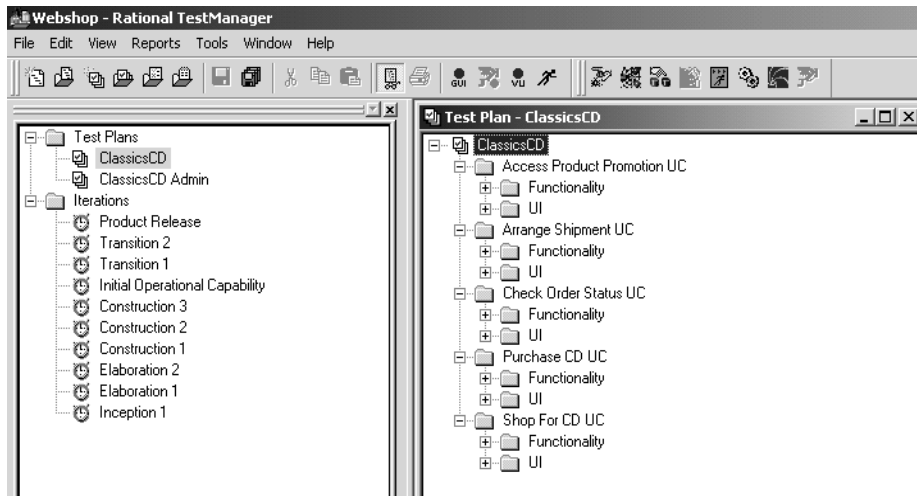
In TestManager, a *test plan* can contain information about the purpose and goals of testing within a Rational project, and the strategies to implement and run testing. You can have one or more test plans in a project. A test plan can include properties such as the test plan name, configurations associated with the test plan, and a time frame for when a test plan must pass.

You can generate reports based on a test plan's properties. For example, you can create reports to determine which test cases are part of a test plan. Reports like this can give you valuable information about the state of your testing project.

Organizing Your Test Plan

A test plan contains *test case folders*, which in turn contain test cases. A *test case* is a testable and verifiable behavior in a system. You can organize the test plan and test case folders in the way that makes sense for your organization. For example, you can create a test plan for each testing type, for each use case, or for each iteration or phase of testing. Alternatively, you can have a master test plan with test case folders that represent use cases, as in this tutorial (see Figure 20).

Figure 20 Sample Test Plan Organization in TestManager



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.

Exercise: Understand the structure of the ClassicsCD.com test plan.

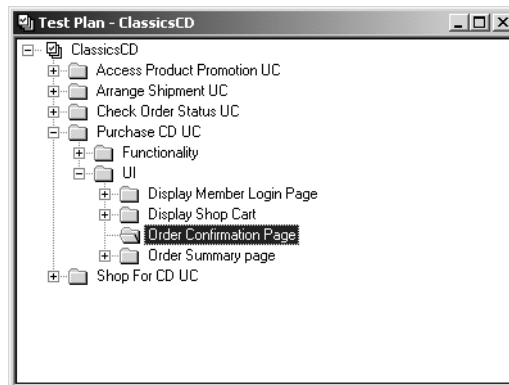
- 1 From the Planning tab in the Test Asset Workspace, expand **Test Plans** and then double-click **ClassicsCD**.

The right pane displays the test case folders for ClassicsCD. In the ClassicsCD.com Webshop project, test case folders are organized by use cases, as described earlier and seen in Figure 20.

- 2 Expand the test case folder for the **Purchase CD** use case. Two folders appear, **Functionality** and **UI**. In this tutorial, these test case folders hold test scripts that represent the system behavior (**Functionality**) and the Web site's presentation of this functionality, user interface (**UI**).
- 3 Expand the **UI** test case folder, and then the **Order Confirmation Page** folder.

Notice that there are no test cases associated with this folder, and a test script for this does not yet exist for the modification you made earlier to the Complete Order use case requirement (see Figure 21). You must create the test case so that you can later develop a test script for the enhancement to display ship dates for ordered CDs.

Figure 21 Test Case Folder in TestManager



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 must 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 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 must 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 testable and verifiable behavior in a system. A test case can also describe the extent to which you will test an area of the application. Existing project artifacts, such as requirements, provide information about the application and can be used as test inputs for your test cases. 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.

After you identify your test inputs, you can create test cases and associate them with test inputs. These associations allow you to respond if test inputs change. These changes might require you to change the test cases or their *implementations*.

Exercise: Create a test case for the enhancement to the Complete Order requirement.

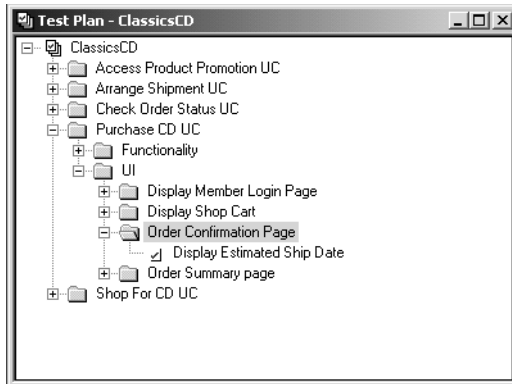
- 1 In the right pane of TestManager, right-click the **Order Confirmation Page** folder, and from the shortcut menu click **Insert Test Case**.

The New Test Case dialog box appears. Information you include in these boxes helps to define the test case which will be inserted under the test case folder.

- 2 On the General tab:
 - a In the **Name** box, type **Display Estimated Ship Date**.
 - b In the **Description** box, type:
Warehouse system gets order and responds with estimated ship date.
 - c In the **Owner** box, make sure that `pat` is selected.
- 3 Click **OK**.

The test case **Display Estimated Ship Date** appears in the test plan hierarchy (see Figure 22).

Figure 22 ClassicsCD.com Test Plan with New Test Case in TestManager



Test Inputs from Rational Rose

If you have Rational Rose installed and licensed, then you can register Rose models with TestManager and use Rose model elements as test inputs. You can view each individual model element in the TestManager Test Input window, and create an association between a model element and a test case.

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 TestManager Test Input window after you log on to that project. You can then create an association 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.

Exercise: Associate a test input with a test case.

- 1 In the right pane of TestManager, right-click the test case **Display Estimated Ship Date**, and from the shortcut menu click **Associate Test Input**.

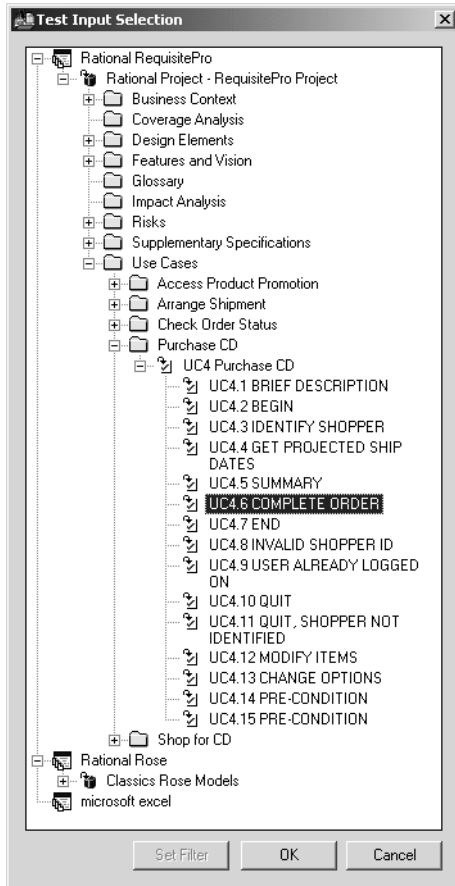
It may take a moment for TestManager to access artifacts associated with the Webshop project, be patient. Soon, the Test Input Selection dialog box appears. You use this feature to associate project artifacts to test cases.

- 2 Go to **Rational RequisitePro > Rational Project - RequisitePro Project**.

TestManager displays a progress indicator as it retrieves information from RequisitePro. After a pause, TestManager displays the associated RequisitePro assets in the Test Input Selection hierarchy.

- 3 Expand the tree nodes as follows: **Rational Project - RequisitePro Project > Use Cases > Purchase CD > UC4 Purchase CD > UC4.6 COMPLETE ORDER** (see Figure 23).

Figure 23 ClassicsCD.com Test Plan Input Selection in TestManager



- 4 Select **UC4.6 COMPLETE ORDER** and click **OK**.

The use case requirement you modified earlier in this tutorial is now associated with this test case.

- 5 From the TestManager menu bar, click **File > Save All**.

Traceability and Suspect Links

Once a test input is associated with a test case, a dependency between the two artifacts is created. Using this example, if the requirement UC4.6 COMPLETE ORDER were to change, the test case named Display Estimated Ship Date would become suspect.

Recall that in Chapter 5, *Creating Requirements*, you learned about traceability and suspect links. The integration between TestManager and RequisitePro builds on this idea, so that if a requirement's attributes change, any associated test cases will become suspect. TestManager alerts project teams that a requirement has changed and that test cases may be affected. Testers use this information to:

- Asses the impact of the change to test cases and test plans.
- Manage test plans more effectively.
- Work with other team members to more effectively manage project activities and artifacts.

Elaborating on Test Cases

As part of developing your test plan, you must design your tests. Test designs are elaborations of test cases. They provide the detail needed for understanding how the test case will be implemented. You can perform design work in conjunction with, or after, you plan your test cases, depending on the needs of your project.

You design tests using the Design Editor. During this step, you capture the most basic and probable flows in a test case and add validation criteria or verification points.

Understanding the Impact of Test Planning

So far, you have learned how to structure and organize a test plan. In Chapter 7, *Modeling the Enhancement*, you add the Arrange Shipment enhancement into your system's structure for the Checkout user interface. After implementation is complete, testing can begin. In Chapter 10, *Functional Testing*, you use the test case you created in this chapter to perform functional tests on the enhancement.

Test planning helps you identify strategies for testing early and to communicate the intent of testing activities to all stakeholders. Test planning in TestManager is designed to work for your entire team:

- *Analysts* use test plans to configure test inputs, define project iterations, and run test coverage reports.
- *Developers* can use test plans to perform *unit tests* and to verify that the test cases are consistent with the implementation and development plans.

- *Testers* use test plans to organize test cases (which are created from test inputs), develop and run tests, and analyze test results.
- *Project leaders* and *managers* use test plans to define project iterations, create custom reports, and run test coverage reports.

Continuing with Test Planning

Building 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. After you identify the available testing resources, you must 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, might consider not running it.
- **Might 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 to Perform

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

- **Reliability tests** – Can the application function without errors? Use Purify, Quantify, and PureCoverage for reliability testing.
- **Functional tests** – Does the application meet its functional requirements? Use Rational Robot for functional testing.
- **Performance tests** – Is the system's performance acceptable under varying workloads? Use Robot to record performance tests. Use TestManager to run these test scripts with different workloads, and to analyze the results of a test.

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 correctly when combined to run a test for a use case.
- **System testing** – Ensures that the software is functioning as a whole.
- **Acceptance testing** – Verifies that the software is ready for delivery and that it meets its requirements.

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

Project Scheduling

Part of creating a test plan involves developing 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 must 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 prioritize your tests as described in *Risks and Resources* on page 80, make sure you schedule at least the “must” (high priority) and “should” (medium priority) tests. If resources become 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.

TestManager comes with built-in iterations as defined in the RUP, or you can create your own. You can associate iterations with test cases and then run these test cases based on iterations.

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

- 1 Click **Start > Programs > Rational Software > Rational RequisitePro**. RequisitePro opens and the Open Project dialog box appears.

If, after starting RequisitePro, the Let's Go RequisitePro window appears, click **Close**.

- 2 In the Open Project window, click **Cancel**.
- 3 On the RequisitePro menu bar, click **Help > Contents and Index**.
- 4 In the RequisitePro Help Browser, on the Contents tab, go to (by double-clicking) **Wizards, Integrations and Components > RequisitePro Wizards > MS Project Integration Wizard**.

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

More on Test Artifacts

While working with every aspect of test planning is beyond the scope of this tutorial, this section provides pointers to help you learn more about the artifacts involved.

Exercise: Learn more about test artifacts.

- 1 If the RUP is still open, make it the active window. If the RUP is not already started, click **Start > Programs > Rational Software > Rational Unified Process**.
- 2 In your Web browser, select the **Tester** tree set and go to **Test > Artifact Overview** to display the Test: Artifacts Overview page.

This overview identifies the roles and artifacts involved in testing. For more information about any of these elements, click that element.

- 3 When you have finished using the RUP, you may quit the application. Or, if you would like to use the RUP as you work through this tutorial, learning more about the topics as you cover them, minimize the Rational Unified Process so you can easily use it when you like.

Summary

For More Information

For more information about test planning:

- Read about test plans in the *Rational TestManager User's Guide*.
- For a more in-depth treatment of test planning, read *Testing Computer Software* (Vnr Computer Library) by Cem Kaner and others (ISBN: 1850328471).

Cleaning Up

Quit TestManager by choosing **File > Exit** from the TestManager menu bar.

If necessary, close the Rational RequisitePro Online Help window and quit RequisitePro. If you are prompted to quit, click **Yes**.

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 include the Arrange Shipment enhancement request in the visual model for ClassicsCD.com.

So far, you have defined a use case requirement for the ClassicsCD.com enhancement. The test organization has started test planning. In this chapter, you continue to incorporate designs for the use case requirement into a ClassicsCD.com visual model using Rational Rose.

Audience

This chapter applies to software designers and developers.

Getting Your Bearings

In this chapter, you use Rational Rose. To determine whether Rose is installed on your computer, refer to the tool chart you filled out in Table 1, Rational Suite Tools, on page 27.

If Rose *is not* installed, you can still benefit from reading this chapter, but you will not be able to perform the exercises.

If Rose *is* installed, start it now:

- 1 Click **Start > Programs > Rational Software > Rational Rose**.
- 2 If you did not perform the exercises in *Creating Requirements* on page 55, perform the steps to start Rose page 62. Then skip to the following steps of this exercise.
- 3 In the Create New Model dialog box, click the Recent tab.
- 4 Click the ClassicsCD_WinDNA Rose model and then click **Open**. If Rose asks whether to load subunits, click **Yes**.

Rose displays a hierarchical tree browser in the upper-left pane (the *Rose browser*). In the right pane (the diagram window), it displays the logical view of the architecture showing how the top-level packages in the system interact.

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. This visual model:

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

Using Rational Rose

You can create visual models of architectures, components, and data using the industry-standard UML with Rational Rose. This helps you visualize, understand, and refine your requirements and architecture before committing them to code. Using Rose to develop visual models throughout the development lifecycle helps ensure that you're building the right system. The architecture model can be traced back to both the business process model and the system requirements.

Visual Modeling and the Tutorial

In *Chapter 5, Creating Requirements*, you used Rose and RequisitePro to create a use case requirement. In this chapter, you continue working on the design for the requirement using Rational Rose.

Working with a Sequence Diagram

A *sequence diagram* is a visual representation of the steps through one path in a use case. 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 objects.

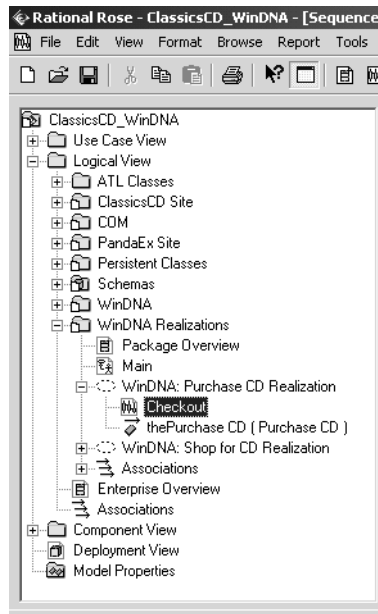
Opening a Sequence Diagram

Start by looking at an existing sequence diagram.

Exercise: Open the sequence diagram.

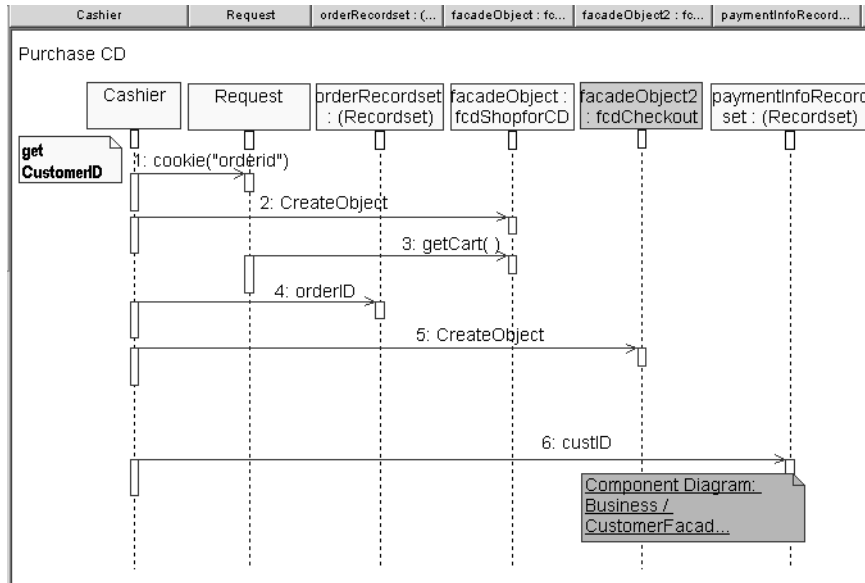
- 1 In the Rose browser, go to **ClassicsCD_WinDNA > Logical View > WinDNA Realizations > WinDNA: Purchase CD Realization > Checkout**. Double-click the sequence diagram **Checkout** to open it (see Figure 24).

Figure 24 Opening the Sequence Diagram Using Rose



Rose displays the Checkout sequence diagram (see Figure 25).

Figure 25 Checkout Sequence Diagram in Rose



- 2 If you cannot see the entire diagram, click **View > Fit in Window**.

This sequence diagram shows how the *actors* and other *objects* in the application communicate. A *message symbol*, a horizontal, solid arrow between two vertical, dashed *lifelines*, illustrates how objects in a sequence diagram communicate with each other. Items in a sequence diagram are arranged in chronological order.

The tutorial follows this naming convention for objects:

- Names begin with a lowercase letter.
- Names do not contain spaces.
- Within a name, the first letter of each word is capitalized.

The diagram also details how the ClassicsCD.com server handles the checkout process after a customer places an order. It shows that the Cashier Active Server Page calls upon the Checkout façade to verify member logins, get payment information, and confirm orders. Specifically, the first few messages mean that:


- 1 The Cashier Active Server requests a cookie to obtain the member’s ID.
- 2 The Cashier Active Server creates a façade object.
- 3 The Cashier Active Server sends the façade object a message to get the contents of the member’s shopping cart.

Adding Messages for the Enhancement

In this section, you add messages to the sequence diagram. Your messages show how objects in the system will communicate to implement the enhancement you are working on.

Exercise: Add to the sequence diagram.

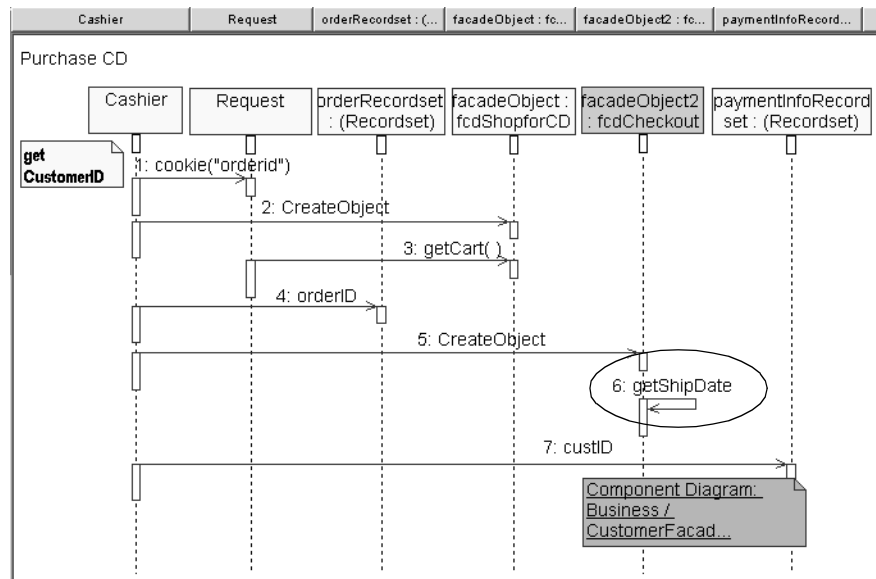
Starting on the sequence diagram:

- 1 On the diagram toolbox (between the Rose browser and the diagram window), click  (the Message to Self button).
- 2 To place the message, click the diagram on the facadeObject2:fcdCheckout lifeline (the vertical line associated with the green Rose Object), just under Message 5, CreateObject.

Notice that Rose rennumbers the subsequent message.

- 3 While the newly inserted **Message to Self** object is still selected, type **getShipDate** and then click anywhere in the background of the diagram (see Figure 26).
- 4 Click **File > Save**. If Rose prompts you to save subunits of the model, click **Yes**. If Rose prompts you to save subunits of the model to a new format, click **Yes**.

Figure 26 The Finished Sequence Diagram in Rose



Publishing Part of the Model to the Web

Now that you have finished working on this part of the model, we recommend that you create a Web version of it so that people on your team who have not installed Rose can review the model and give you feedback.

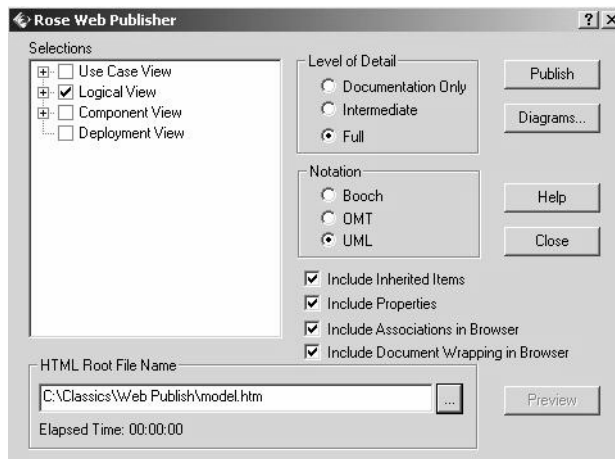
With Rose Web Publisher, you can create a Web-based (HTML) version of a Rose model that others can view using a standard Web browser. Rose Web Publisher recreates Rose model elements, including diagrams, classes, packages, relationships, attributes, and operations. After the version is published, you can use hypertext links to explore the model as you would in Rose.

Exercise: Publish the model to the Web.

- 1 From the Rose menu bar, click **Tools > Web Publisher** to open the Rose Web Publisher dialog box.
- 2 In the **Selections** list,
 - a Double-click to clear the **Use Case View** check box
 - b Double-click to clear the **Component View** check box
 - c Click to clear the **Deployment View** check box.
 - d Check **Logical View** check box.

So, of the selections available, only **Logical View** should be selected (see Figure 27).

Figure 27 Publishing Models to the Web Using Rose



- 3 Next to the **HTML Root File Name** box, click **...** and go to C:\Classics\Web Publish. Then type **model** in the **File name** box, and click **Save**.

The **HTML Root File Name** box displays this path (see Figure 27).

- 4 On the Rose Web Publisher dialog box, click **Publish**.

Rose displays a progress indicator as it accesses and converts the models to HTML. This process may take a few minutes. When the progress indicator disappears, the Web files are ready.

- 5 Click **Preview**.

Your Web browser displays a hierarchical tree browser in the upper-left pane (the Rose browser). Information about Rose Web Publisher appears in the right pane (the diagram window). Once a Rose model is selected, Rose displays it in the diagram window.

- 6 In the Rose Web Publisher browser go to **Logical View > WinDNA Realizations > <<use case realization>> WinDNA: Purchase CD Realizations > Checkout**.

Double-click this entry to explore the sequence diagram you worked with in this chapter.

- 7 Finish reviewing the diagram and then do one of the following:

- If the Rational Unified Process (RUP) was open before you started this exercise, click **Back** until you return to the RUP. In Rose, on the Rose Web Publisher dialog box, click **Close**.
- If the RUP was *not* open before you started this exercise, close your Web browser. Then on the Rose Web Publisher dialog box, click **Close**.

Continuing Work with the Sequence Diagram

Now that you have finished this part of the model, there are a few additional tasks to perform. In this tutorial, we summarize the tasks, but do not expect you to perform them.

Refining the Objects

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

For example, to see a class diagram, use the Rose browser to explore to **Logical View > ClassicsCD Site > Client Composition Diagrams**. Double-click **cashier_Client Diagrams**. Each representation of a class shows you the class attributes and operations. (Double-click a class representation to see details about the class.)

After 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 a command from the Rose **Tools** menu. For example, to implement Java code for the enhancement you've been working on, you would click **Tools > Java / J2EE > Generate EJB JAR File**, or **Generate WAR File**.

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 through a process called *reverse engineering*, where Rose automatically updates the model from changes in the code. To reverse engineer after updating source code, you would click **Tools > Java / J2EE > Reverse Engineer**.

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

Note: Rose Enterprise Edition and Rose Professional Data Modeler Edition can generate code, update code, and update models. For database schemas, both editions can generate code from a Rose visual model and update a Rose model from source code through a process called *round-trip engineering*. However, Rose Professional Data Modeler Edition only supports round-trip engineering to and from DDL scripts and database schemas – not for other languages such as Java and Visual C++.

Modeling Data

You can use Rose to model relational databases. Rose Professional Data Modeler Edition is a database modeling and design tool that uses UML. It helps you:

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

Benefits

The benefits of using Rose Professional Data Modeler are:

- All your business, application, and data models are written in UML, the same industry-standard language 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.

Summary

For More Information

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

For information about Rational Rose RealTime, see the online tutorials available from Rose RealTime 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 by choosing **Help > Extended Help**. In the **Extended Help - Rose** tree set, click **Tool Mentor: Managing Subsystems Using Rational Rose**.

Cleaning Up

If necessary, quit the Rose Extended Help.

Quit Rose. If you are prompted to save your changes, click **Yes**. If you are prompted to save subunits, click **Yes**.

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 from visual models, and keep models synchronized with changed code.
- You can use Rose to publish read-only copies of your models and diagrams to the Web. 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

The visual model for the enhancement is now complete. In the next chapter, you create a report about the use case for the enhancement.

Now that you have elaborated the Arrange Shipment use case with the new requirement, you want to communicate changes made to the requirements to all team members and to stakeholders. To do this, you might want to generate a report consolidating all the information about the use case. Such a report might contain the sequence diagram from the visual model in Rational Rose and the corresponding basic flow from the use case in Rational RequisitePro.

You also want to communicate the status of the project. You might want to gather project metrics from Rational Suite tools to determine the progress made in this iteration of the ClassicsCD.com project. Charts might help you determine trends and gauges might help you compare data to predefined threshold values.

In this chapter, you use Rational SoDA to produce a use case report and Rational ProjectConsole to view project artifacts, analyze metrics, and determine project status.

Audience

This chapter applies most directly to project leaders and managers, but is relevant for all members of a software development team.

Getting Your Bearings

In this chapter, you use SoDA and ProjectConsole. Rose and RequisitePro must also be installed on your computer. To determine whether these tools are installed on your computer, refer to the tool chart you filled out in Table 1, Rational Suite Tools, on page 27.

If any of these tools *are not* installed, you can still benefit from reading this chapter, but you will not be able to perform the exercises.

If these tools *are* installed, start SoDA now by clicking **Start > Programs > Rational Software > Rational SoDA for Word**.

Microsoft Word starts and automatically opens a blank document containing an additional SoDA menu. But first:

- If you see a warning about enabling macros, make sure that you click **Enable Macros**. Otherwise, you cannot use SoDA.
- If you do not see this warning, go to **Tools > Macro > Security**. The Security dialog box opens. On the Security Level tab, make sure that your macro security level is set to either **Medium** or **Low**. Otherwise, you cannot use SoDA. After you finish the exercises in this chapter, you can reset your macro security level to a higher setting.

Managing Project Status

Rational ProjectConsole and Rational SoDA provide tool support for managing project status.

What Is SoDA?

SoDA automates software documentation, like status reports. It is tightly integrated with many Rational tools so you can easily extract information to report on requirements, designs, tests, and defect status. For example, you can use SoDA to report on:

- Versioning information with ClearCase LT
- Reported defects with ClearQuest
- Requirements with RequisitePro
- Visual models and designs with Rose
- Test scripts with TestManager
- Information extracted from other documents created in the Windows environment
- Information extracted from multiple Rational tools and other information sources into a single, integrated document.

Using SoDA Templates

To generate reports and other documents, SoDA relies on a predefined template for Word. The template gathers information and formats it into a report.

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. SoDA preserves your text 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 the easy-to-use template creation tool in SoDA.

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 diagrams that show how the system interacts with actors. Use case reports are helpful to your entire team:

- *Analysts* 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.
- *Developers* use the report's description of expected system behavior to start writing engineering specifications and designing the system architecture.
- *Testers* use the report to design tests for the use case. From the report, a tester can identify the steps to test and determine which conditions to test.
- *Technical writers* start planning documentation based on the report's descriptions of how users interact with the system.
- *Usability engineers* use the report to design usability tests, possibly starting with paper prototypes.

Creating the Use Case Report

To create the use case report, SoDA relies on a predefined template that gathers information and formats it into a report. The report you create in this chapter includes information from RequisitePro and Rose. Therefore, if you have not performed the exercises in Chapter 5, *Creating Requirements*, and Chapter 7, *Modeling the Enhancement*, you cannot create the use case report.

Exercise: Create the use case report by starting with the template.

- 1 In Word, open C:\Classics\Projects\Webshop\SoDA\RUP Use Case Report.doc.

You may see a warning about enabling macros if you reset your macro security levels at the beginning of this chapter. Make sure that you click **Enable Macros**. Otherwise, you cannot use SoDA.

Word displays the SoDA template containing text, macro commands, and annotations.

- 2 View the entire template including SoDA commands by doing one of the following from the Word toolbar:
 - If you are using Word 2000 or later:
 - Click **Tools > Options**. On the View tab, under **Formatting marks** select the **All** check box. Click **OK**.
 - If you are using Word 2002:
 - Click **SoDA > Show/Hide Commands**, and display comment markers by clicking **View > Markup**.

Pink and yellow Word annotations used by SoDA appear in the document to store the SoDA commands (see Word 2000 example of display in Figure 28). It is important to see this hidden text when working with SoDA for Word.

Figure 28 Viewing the Use Case Report Template Using SoDA (Excerpt)

```

[OPEN1]
[OPEN2][OPEN3]::[The title and table of contents will come from the use case specification]
[DISPLAY4]<UseCaseSpecification.FormattedText>[ENDDISP5]
* 1. → Relationships
* 1.1 → Communicate--Associations
[MASTER6][REPEAT7][LIMIT8]
* 1.1.1 → Communicate with actor [DISPLAY9]<RoleA_ToClass.Name>[ENDDISP10]
Documentation: [DISPLAY11]<Communicate.Documentation>[ENDDISP12][ENDLIM13]
* 1.1.2 → [LIMIT14]Communicate with actor [DISPLAY15]<RoleA_FromClass.Name>[ENDDISP16]
Documentation: [DISPLAY17]<Communicate.Documentation>[ENDDISP18]
[ENDLIM19][ENDREF20][ENDMAST21][LIMIT22]No <<communicate>> associations have been
defined.[ENDLIM23]

```

Note: Hidden text may display differently depending on which version of Word you are using. Still, when working with SoDA, you should always have the Show/Hide function turned on to display all hidden text. If you prefer not to see all formatting marks when you work with Word outside of SoDA, remember to reset this option later.

- 3 Click **SoDA > Generate Report**.

Word closes and SoDA displays the Progress Indicator while retrieving information from the use case document and the Rose model that you worked with earlier. This process may take a few moments. Then, SoDA for Word displays the report.

- 4 Browse through the report.
 - a Go to page 4 to see the requirement you worked on in Chapter 5, *Creating Requirements*.
 - b Go to page 6 of the report to see the visual model you worked with in Chapter 7, *Modeling the Enhancement*.

Working with SoDA Templates

Although working with SoDA templates is beyond the scope of this tutorial, this section provides pointers to help you get started.

Optional Exercise: View the template tool for this SoDA template.

- 1 In Word, display the use case template, RUP Use Case Report.doc by clicking **Window > RUP Use Case Report**.
- 2 Click **SoDA > Template View** to display the SoDA Template View tool.
You use this Template View to create or enhance SoDA templates.
- 3 Click **Help > Help on SoDA** to learn about the elements in this Template View.
Use this information to understand how each line translates into the SoDA commands for the RUP Use Case Report template.
- 4 Close SoDA Help and the Template View.
- 5 Then quit SoDA for Word by choosing **File > Exit**. If you are prompted to save any of your changes, click **No**.

What Is ProjectConsole?

ProjectConsole simplifies access to project information and helps you measure progress and quality. It automatically collects project artifacts from all Rational Suite tools and structures them into a project Web site that all Web-enabled team members can use. In addition, ProjectConsole collects metrics about requirements, designs, tests, and defect status so that you can easily understand the *true* status of your project.

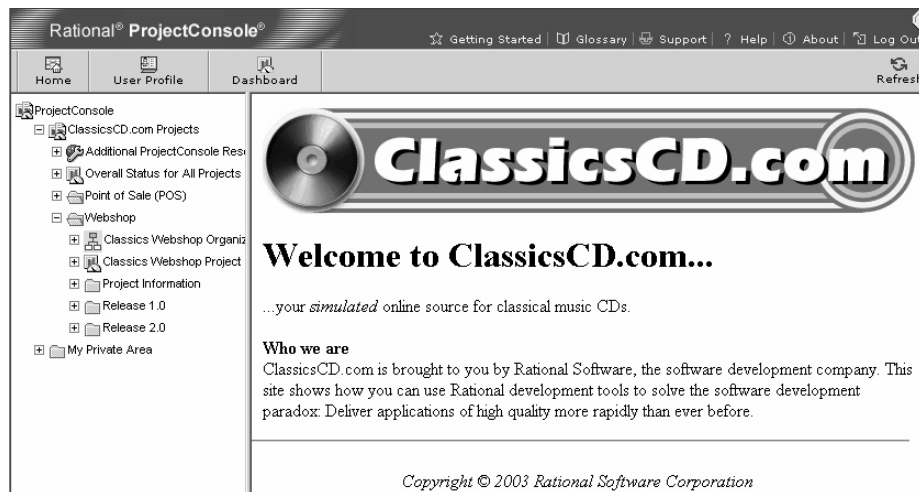
Using the Project Web Site

ProjectConsole improves team communication by providing one project Web site that hosts project artifacts such as use case reports, defects, and metric reports.

ProjectConsole automatically extracts artifacts like these from Rational Suite or select third-party tools. ProjectConsole displays the artifacts in a project Web site according to the structure you have defined (see Figure 29).

This flexibility allows project leaders to easily adapt the Web site's information structure to best fit the needs of their team or organization. Project leaders decide what information should be displayed and how it should be organized so that all team members can easily find and access project artifacts. The project Web site can be refreshed on demand and on a schedule, making sure that all team members are always viewing the most current artifacts.

Figure 29 Example of ProjectConsole Web Site



Working with Project Metrics

ProjectConsole extracts information from data produced throughout a software development project and automatically generates graphics, such as charts and gauges, either predefined or ones that you customize. These metrics allow you to automatically collect information about the status of your project and share it with members of your team. With ProjectConsole, you can also analyze data in a single, integrated view collected from several Rational tools.

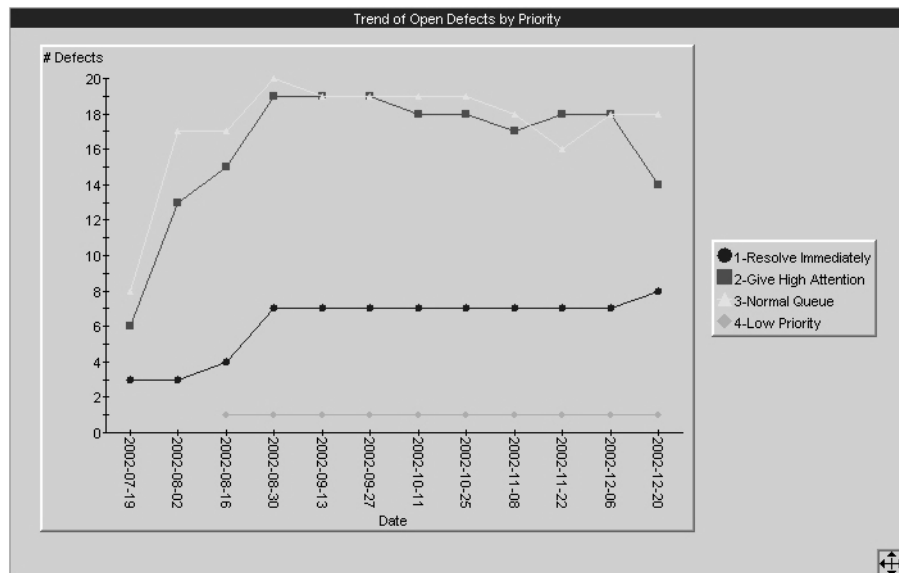
Table 3 shows you how ProjectConsole is used to measure progress and quality.

Table 3 Using ProjectConsole Metrics to Understand Project Status

Use ProjectConsole to	by gathering data about	using
Determine whether your application or product is stabilizing	Lines of code being added, modified, or deleted	ClearCase LT
	Visual model elements being added, modified, or deleted	Rose
	Reported defects	ClearQuest
See how many additional tasks must be performed in this iteration	Open defects	ClearQuest
	Open features, use cases, and requirements	RequisitePro
	Open test cases	TestManager
Assess the quality of your application or product	Open defects by severity	ClearQuest
	Trend of test results	TestManager

Figure 30 shows you how ProjectConsole can be used to assess the trend of open defects over time.

Figure 30 Working with Trend Charts Using ProjectConsole



From this trend chart, wherein we see that high priority defects are decreasing over time, we could determine that developers are successful at stabilizing the application by the end of the iteration.

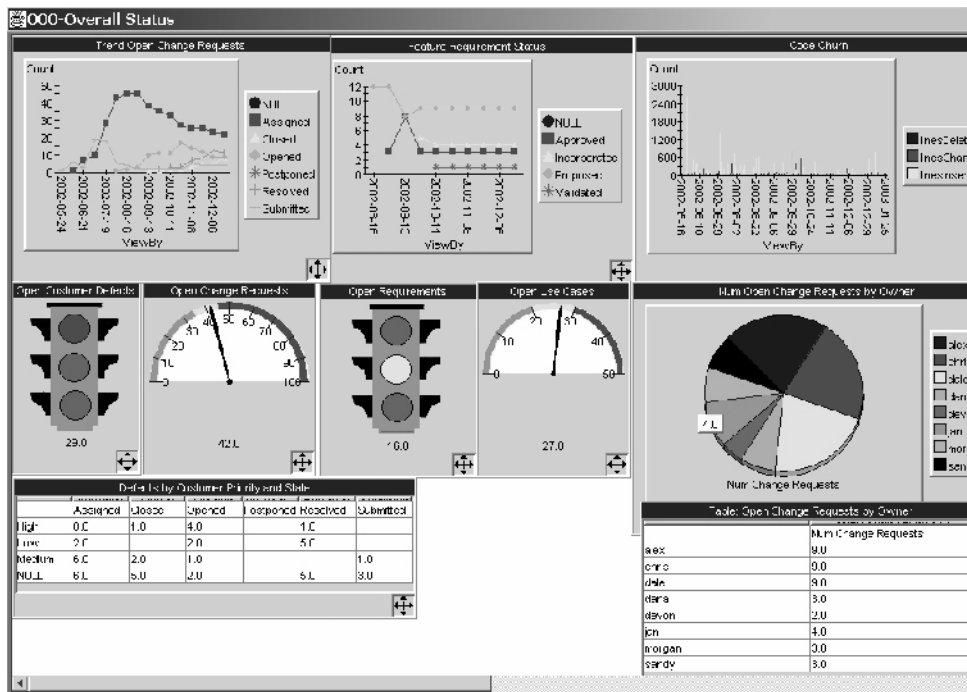
Analyzing Metrics

ProjectConsole also provides a dynamic metric analysis tool, the Dashboard, which helps you drill-down and perform root-cause analysis.

Figure 31 shows you how the ProjectConsole Dashboard can be used to assess overall project status using indicators and charts. Figure 31 also demonstrates how the Dashboard allows you to:

- View metrics across different disciplines (requirements, design, testing).
- Understand how the data corresponds to the current project phase.
- Analyze project status.

Figure 31 Assessing Overall Project Status Across Disciplines



Summary

For More Information

To learn more about SoDA, in SoDA for Word, click **Help > Help on SoDA**. A window showing a list of SoDA topics opens. Choose your topic of interest.

For more information on using ProjectConsole, start with the tutorial. Click **Start > Program Files > Rational Software > Rational ProjectConsole > Rational ProjectConsole Tutorial**.

For more information on using ProjectConsole Dashboard, read *Getting Started: Rational ProjectConsole*.

To learn more about building templates with ProjectConsole, you must first install the Rational ProjectConsole Template Builder. Then start ProjectConsole by clicking **Start > Program Files > Rational Software > Rational ProjectConsole Template Builder**. Microsoft Word starts and automatically opens a blank document which contains an additional ProjectConsole menu. From the Word menu bar, click **Help > Help on Template Builder**. A window showing a list of topics opens (you may need to maximize this window). Choose your topic of interest.

Cleaning Up

If necessary, quit Word. If you are prompted to save your changes, click **No**.

What You Learned in This Chapter

In this chapter, you learned:

- SoDA automates software documentation by creating reports based on templates. It contains an easy-to-use tool that assists you with template creation.
- A use case report is useful to all members of your project.
- ProjectConsole automatically collects project artifacts from Rational Suite tools and structures them into a project Web site for Web-enabled team members to use.
- ProjectConsole automatically generates project metrics by extracting information from data produced during software development. This provides you with an accurate and objective assessment of the project status.

What's Next

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

The ClassicsCD.com enhancement is implemented and testing has been planned. You are now ready to test the enhancement. In this chapter, we discuss testing for reliability.

Audience

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

Reliability Testing Tools

This chapter describes the following automated testing tools:

- **Rational Purify.** Pinpoints run-time errors and memory leaks in Visual C++ application code, and errors related to garbage-collection in Java and .NET managed application code.
- **Rational PureCoverage.** Identifies the parts of your Java, Visual C++, Visual Basic, Visual Basic.NET, or Visual C#.NET program that 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++, Visual Basic, Visual Basic.NET, or Visual C#.NET application to help you identify performance bottlenecks in your code.

Run-Time Analysis Tools in Rational Suite

Rational Purify

Run-time memory corruption 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 not.

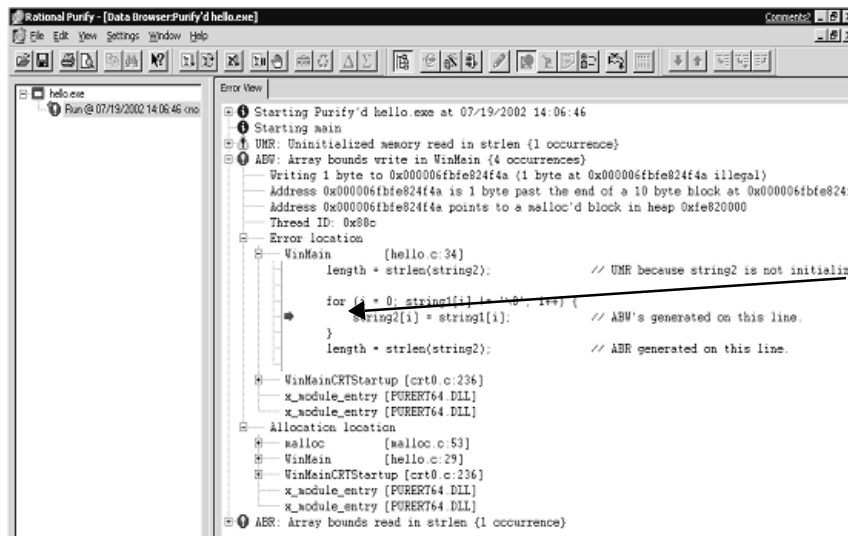
Rational Purify is a comprehensive run-time tool used to check for:

- Memory corruption and memory leaks in C/C++ code applications.
- Memory profiling and memory leaks in Java and .NET managed code applications.

Purify can find memory errors in every component of your program, even if 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 and .NET managed code applications, 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. You can use Purify's coverage data to make sure that all your code is free of errors (see Figure 33).

Figure 32 Sample Error Detection Results from Rational Purify



Find errors

Rational PureCoverage

To effectively test an application, you need to know which parts of the application were exercised during a test 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 C/C++, Visual Basic 6, Visual Basic.NET, Visual C#.NET and Java code.

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 (see Figure 33). It is essential to an automated testing environment.

Figure 33 Sample Test Coverage Results from Rational PureCoverage

Coverage Item	Calls	Methods Missed	Methods Hit	% Methods Hit	Lines Missed	Lines Hit	% Lines Hit
Run @ 12/13/2001 12:59:31 (no arguments)	13096	506	198	23.83	74	288	79.56
blocks.vb	963	3	21	87.50	74	288	79.56
Tetrts_Gogo.Block.UndoLeft()	238	1	7	87.50	29	57	66.29
Tetrts_Gogo.Block.Rotate()	0	missed	5	0	5	0	0.00
Tetrts_Gogo.Block.ReturnBlock()	5	hit	0	0	9	100.00	100.00
Tetrts_Gogo.Block.MoveRight()	175	hit	0	0	14	100.00	100.00
Tetrts_Gogo.Block.MoveLeft()	3	hit	0	0	3	100.00	100.00
Tetrts_Gogo.Block.MoveDown()	45	hit	0	0	3	100.00	100.00
Tetrts_Gogo.Block.Create()	42	hit	0	0	3	100.00	100.00
Tetrts_Gogo.Block.Close()	4	hit	24	24	24	50.00	50.00
Tetrts_Gogo.Block.Close()	4	hit	0	0	1	100.00	100.00
Frm1.vb	715	2	14	87.50	45	231	83.70
Tetrts_Gogo.Frm1.subDrawBox()	248	hit	0	0	11	100.00	100.00
Tetrts_Gogo.Frm1.subCloseBox()	220	hit	0	0	9	100.00	100.00
Tetrts_Gogo.Frm1.cmdStat_Click()	1	hit	0	0	9	100.00	100.00
Tetrts_Gogo.Frm1.cmdExit_Click()	0	missed	3	0	0	0.00	0.00
Tetrts_Gogo.Frm1.Timer1_Tick()	45	hit	4	16	80.00	80.00	
Tetrts_Gogo.Frm1.Main()	1	hit	0	0	1	100.00	100.00
Tetrts_Gogo.Frm1.InvokeCompo.	1	hit	0	0	61	100.00	100.00
Tetrts_Gogo.Frm1.InvokeGame()	2	hit	0	0	13	100.00	100.00
Tetrts_Gogo.Frm1.Frm1_Load()	1	hit	0	0	2	100.00	100.00
Tetrts_Gogo.Frm1.Frm1_KeyDown	13	hit	12	22	84.77	84.77	
Tetrts_Gogo.Frm1.DrawBlock()	117	hit	0	0	0	100.00	100.00
Tetrts_Gogo.Frm1.Dispose()	0	missed	4	0	0	0.00	0.00
Tetrts_Gogo.Frm1.CreateViewBox.	4	hit	0	0	7	100.00	100.00
Tetrts_Gogo.Frm1.CheckForLine3.	3	hit	20	17	45.95	45.95	
Tetrts_Gogo.Frm1.CanMove()	58	hit	2	33	94.29	94.29	
Tetrts_Gogo.Frm1.Close()	1	hit	0	0	9	100.00	100.00
[Unknown Dictionary]	12403	502	137	21.44	0	0	0.00
[Unknown File]	12403	502	137	21.44	0	0	0.00
WndProc.Invoke()	0	missed	0	0	0	0.00	0.00
WndProc.EndInvoke()	0	missed	0	0	0	0.00	0.00
WndProc.BeginInvoke()	0	missed	0	0	0	0.00	0.00
WndProc.Close()	0	missed	0	0	0	0.00	0.00
WindowClass.UnregisterClass()	4	hit	0	0	0	0.00	0.00

See coverage summary

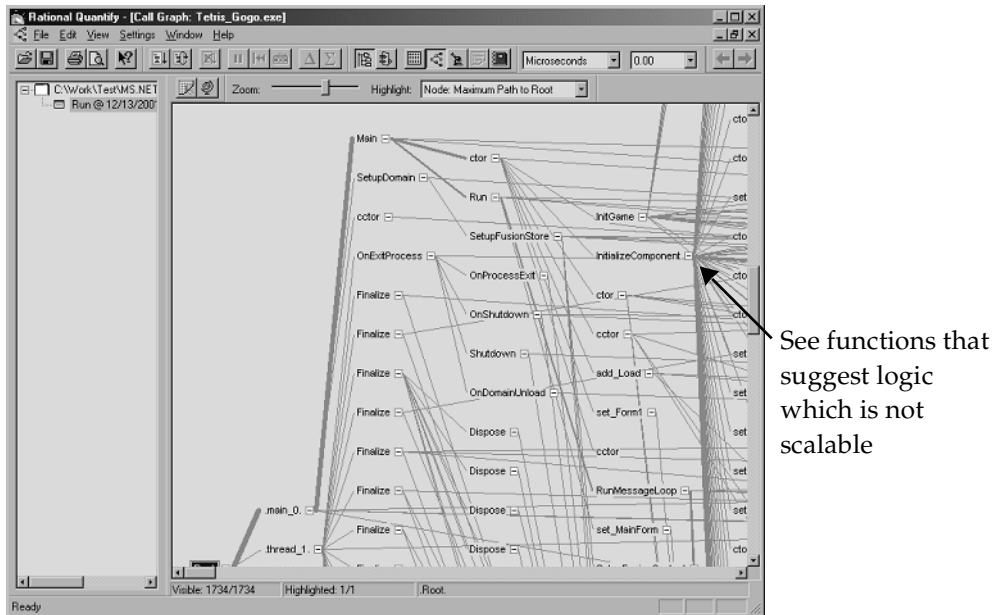
Find untested code

Rational Quantify

Rational Quantify quickly locates performance bottlenecks in C/C++, Visual Basic 6, Visual Basic.NET, Visual C#.NET and Java code. It takes the difficulty and guesswork out of performance tuning by delivering accurate, repeatable timing data for all the components of your program, even if you don't have the source code.

Quantify gives you the insight you need to write more efficient code and make any program work faster (see Figure 34). It can turn everyone on your team into a performance engineer.

Figure 34 Sample Performance Analysis from Rational Quantify



Using Purify, PureCoverage, and Quantify with Other Rational Tools

Rational Purify, PureCoverage, and Quantify integrate with:

- ClearQuest, so you can submit defects directly from your run-time analysis tool and communicate reliability findings with other team members.
- Robot, so you can run test scripts and profile your application at the same time, simultaneously testing an application's reliability and functionality. This helps your team work more effectively and efficiently.

Summary

For More Information

For more information about:

Purify, **PureCoverage**, and **Quantify**. Read *Getting Started: Rational PurifyPlus*, *Rational Purify*, *Rational PureCoverage*, *Rational Quantify*.

Rational books are available on the *Rational Solutions for Windows – Online Documentation* CD-ROM, or online at <http://www.rational.com/documentation>.

For more information about Purify, PureCoverage, and Quantify, read their respective Help systems.

What You Learned in This Chapter

In this chapter, you learned:

- Developers and testers use the following Rational run-time analysis tools:
 - Purify, used to find run-time memory corruption errors for C++ code and profile memory usage for Java and .NET managed code.
 - PureCoverage, used to analyze code coverage.
 - Quantify, used to pinpoint performance bottlenecks.

What's Next

In the next chapter, you perform functional testing on the ClassicsCD.com enhancement. You 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.

At this point in the development process, developers have implemented the ClassicsCD.com enhancement, and testers have run initial reliability tests (this work was completed outside of the tutorial). In this chapter, you use Rational Robot to implement the test case you created in Chapter 6, *Test Planning*, for functional testing of the enhancement requirement. You also use Rational TestManager to analyze the results of your implementation.

Audience

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

Getting Your Bearings

In this chapter, you start by using Rational TestManager. To determine whether TestManager is installed on your computer, refer to the tool chart you filled out in Table 1, Rational Suite Tools, on page 27.

If TestManager *is not* installed, you can still benefit from reading this chapter, but you will not be able to perform the exercises.

If Robot *is not* installed, you can still benefit from reading this chapter, but you will not be able to perform the exercises.

If TestManager *is* installed, start it now:

- 1 Click **Start > Programs > Rational Software > Rational TestManager**.
- 2 If the Rational Test Login dialog box opens, use these values:
 - a In the **User Name** and **Password** boxes, type **pat**.
 - b Make sure the **Project** box displays **Webshop**.
 - c Make sure the **Location** box displays **C:\Classics\Projects\Webshop\Webshop.rsp**.
 - d Click **OK**.

TestManager opens the Webshop project and displays the Test Asset Workspace in the left window pane. Make sure you can read the tabs at the bottom of the Test Asset Workspace. Move your pointer over the symbol on each tab to see its name and then expand the workspace using the vertical divider until you can clearly see the tab names. You are now ready to work with the project.

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 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. TestManager provides built-in support for implementing and running functional tests created in Robot.

Working with Test Scripts

During test planning, you write test cases, as described in Chapter 6, *Test Planning*. A test case describes the extent to which you will test an area of the application. It can list 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 a test script. A test script has the following components:

- A set of properties, such as the name and purpose of the script.
- A file containing scripting language commands. You generate a script file when you record activities with Robot or other scripting languages and tools.

Scripts and Modularity

You can record a test script that starts an application and proceeds through several steps to achieve a certain end result. If a particular activity must be performed in many test scripts, it makes sense to create a script that performs only this common activity.

Instead, you can create a set of test scripts that all start with the same steps and conclude by testing different parts of the application. Using Robot, you can create short modular test scripts, which you can then combine and sequence into a *suite* in TestManager. With this technique, you can reuse the same script in different tests, or run these suites repeatedly against successive builds of your product. Or, you can reuse a test 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 a test script for each of the following:

- Logging onto the system
- Selecting an item to purchase
- Completing a purchase
- Logging off the system

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

Getting to a Starting Point

Recall that in Chapter 6, *Test Planning*, you created the **Display Estimated Ship Date** test case for your enhancement requirement. In this chapter, you reuse a test script that was recorded to get the ClassicsCD.com application to an appropriate starting place for recording a new test script for your test case.

Exercise: Prepare to use Robot.

- 1 Start Robot from TestManager by clicking **Tools > Rational Test > Rational Robot**.
- 2 In Robot, click **Tools > GUI Playback Options**.
- 3 On the GUI Playback Options dialog box, on the Log tab, make sure that the following options are selected:
 - **Output playback results to log**
 - **View log after playback**
 - **Specify log information at playback**
- 4 Click **OK**.
- 5 Quit Robot by choosing **File > Exit**.

Working with Test Scripts

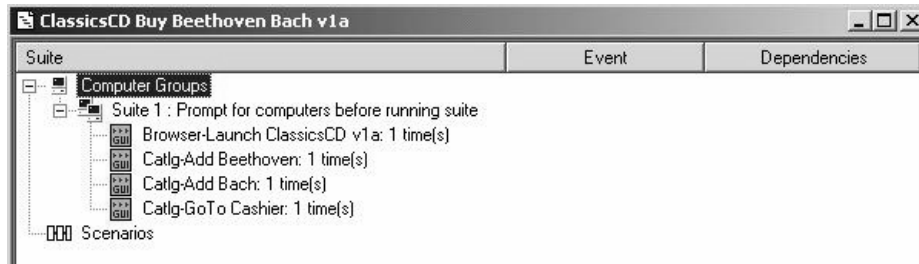
You can record and playback a test script for this tutorial only if Microsoft Internet Explorer is installed on your computer. (It does not need to be the default browser.) If Microsoft Internet Explorer is not installed on your computer, continue reading this chapter and resume performing the exercises starting with *Playing Back the Script on a New Build* on page 120.

Exercise: Playback a script that completes a sale.

- 1 From the Execution tab in the Test Asset Workspace in TestManager (the tree browser in the left pane), expand **Suites**, and double-click **ClassicsCD Buy Beethoven Bach v1a**.

Test scripts for this suite of the ClassicsCD.com Webshop project appear in the right window (see Figure 35).

Figure 35 Viewing Test Scripts Using TestManager



- 2 In the Test Asset Workspace, right-click **ClassicsCD Buy Beethoven Bach v1a**. From the shortcut menu, click **Run**.

The TestManager Run Suite dialog box opens.

In the **Build** list, make sure that **Build 1** is selected. (Use the defaults for the other values.) If **Build 1** is not selected, click **Change**, and from the **Build** list click **Build 1**.

- 3 In the TestManager Run Suite dialog box, click **OK**. If TestManager prompts you to overwrite a test log, click **No** and either save or delete any open test logs. Then resume this exercise at Step 2.

Do not interact with the ClassicsCD.com application while TestManager processes your request to run the script and Robot plays back the scripts in the suite! If you see a Windows message box that starts *Do you want Windows to remember...*, *wait. Robot will eventually continue.*

The script transacts a sale with two line items and a payment. At certain points, the script compares values in the application to a baseline value. When the script finishes, TestManager displays the results of the test in the Test Log window. (If necessary, minimize ClassicsCD.com to see the test results.) All or almost all the comparisons (the verification points) pass.

In the Test Log window, you can learn more about the results by expanding the scripts in the Event Type hierarchy. In some cases, you may see a warning next to a line that says *Unexpected Active Window*. This warning means that during the

playback, an extra window opened on your screen (for example, the message box that starts `Do you want Windows to remember...`). Robot noticed the window but the window did not interfere with the test results. If Robot returned the warning, double-click the warning line to see a screenshot of the unexpected displayed in the Image Comparator window. After you finish, close the Image Comparator window.

- 4 Close the Test Log window.

Recording the Script

You use Robot to record a script while exercising parts of your application's GUI. During recording, 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 and in suites.

Starting to Record the Script

You have just run a prerecorded script to get the application to a known, consistent starting place. You are now ready to create and record a new test script for the test case *Display Estimated Ship Date* that you created in Chapter 6, *Test Planning*.


Exercise: Get ready to record the script.

Start recording the script from the point where the ClassicsCD Buy Beethoven Bach v1a suite finished.

- 1 Open the ClassicsCD.com Web site. Go to `C:\Classics\ClassicsCD_com_sites\v1a` and then open `Cashier.htm`.

Your Web browser displays the *Cashier* page of ClassicsCD.com v1a, an updated version of the Web site.

- 2 Scroll to the bottom of the Checkout page so that you can see **Place Order**. Notice that the estimated ship date appears.

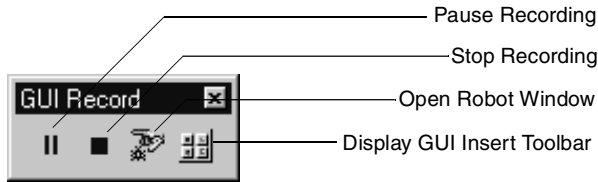
- 3 In TestManager, click  (the Record GUI Script button).

Rational Robot starts and the Record GUI dialog box opens.

- 4 In the **Name** box, type **Display Estimated Ship Date** and then click **OK**.

The GUI Record toolbar appears (see Figure 36).

Figure 36 The GUI Record Toolbar



- 5 Make the Checkout page of ClassicsCD.com the active window.

Creating a Verification Point

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

Exercise: Record the script and create the verification point.



- 1 On the GUI Record toolbar, click .
- 2 On the GUI Insert toolbar, click  (the Object Data button). The Verification Point Name dialog box appears.
- 3 In the **Name** box, type **Verify Ship Date** and then click **OK**.
The Select Object dialog box appears.
- 4 Click the check box command so that this dialog box will automatically close after you choose a verification point.
- 5 Drag the hand pointer to the line on the Checkout page that starts: `We estimate that your order...` (see Figure 37) using a drag-and-drop operation.

Figure 37 Creating a Test Verification Point Using TestManager and Robot

We estimate that your order will be shipped in 4 business days.

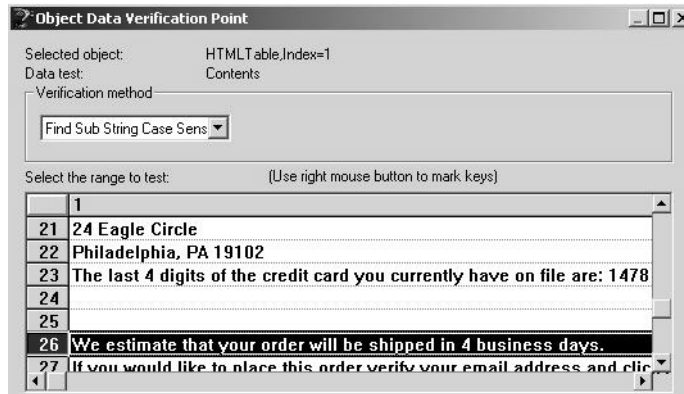
If you would like this order verify your email address and click here:

HTML table Html

- 6 The Object Data Tests dialog box appears, indicating that you have captured the **Contents** of the HTML table. Click **OK** to view the captured text.

- 7 On the Object Data Verification Point dialog box:
 - a From the **Verification method** list, click **Find Sub String Case Sensitive**.
 - b Under **Select the range to test**, scroll to line 26 (We estimate that...) and then click it so that it is the only line selected (see Figure 38).
 - c Click **OK** to close the dialog box.

Figure 38 Working with Test Verification Points Using TestManager




You have now created a verification point and you have almost finished recording this test script. The next time this script runs, it will verify that the text you captured is still displayed. You can include any number of verification points in a script. This script has only one.

Finishing the Recording Session

You can now finish the recording session.

Exercise: Perform the final steps in the script.

- 1 On the Checkout page, click **Place Order**.
- 2 On the GUI Record toolbar, click  to stop recording.
Robot shows the Display Estimated Ship Date script you just recorded.
- 3 Read the script and notice how the commands correspond to the actions you performed as you recorded the script.
- 4 Quit Robot and close ClassicsCD.com.

Adding a Test Script to a Suite

Recall that before you recorded the Display Estimated Ship Date script, you set up the application by running the test suite ClassicsCD Buy Beethoven Bach v1a. In future testing, you need to replay the new Display Ship Date script repeatedly. However, you want to avoid going through manual steps to set up the application each time.

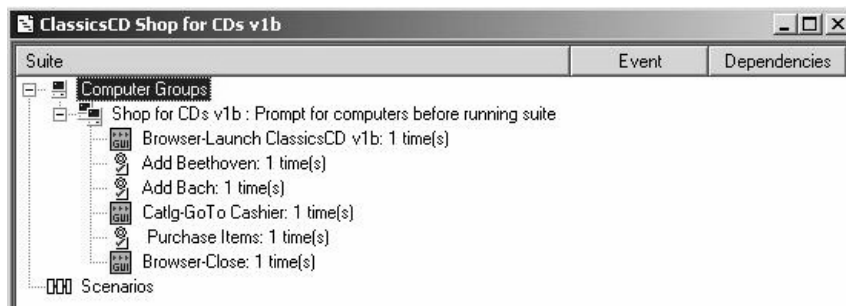
Instead, you can add the Display Ship Date script to an existing test suite that calls other scripts to set up and shut down the application.

Exercise: Add the new script to a test suite.

- 1 From the Execution tab in the Test Asset Workspace in TestManager, go to **Suites > ClassicsCD Shop for CDs v1b** and then double-click this entry.

Test cases and scripts for this suite of your project appear in the right pane (see Figure 39).

Figure 39 Viewing a Test Suite Hierarchy Using TestManager



- 2 In this view, right-click the **Purchase Items: 1 time(s)** test case. On the shortcut menu, click **Insert > Test Script**.

The Run Properties of Test Script dialog box appears.

- 3 In the **Test script source** box, click GUI- (Rational Test Datastore).
- 4 In the **Select** section of the dialog box, scroll to and click the test script **Display Estimated Ship Date**.
- 5 Click **OK**.

The Display Estimated Ship Date test script appears in the hierarchy.

- 6 Choose **File > Save**.

The test script is now associated with this test suite.

The scripts and suites you develop form a set of regression tests that you run after every software build. The outcome of a particular test can change during subsequent iterations as old defects are fixed and new defects and other changes are introduced.

Incorporating a Test Script into Your Test Plan

After you create and record a test script, and associate it with one or more suites, it is important that you incorporate it into your test plan. Recall that in the ClassicsCD test plan, test case folders are organized by use cases. In Chapter 6, *Test Planning*, you created the test case Display Estimated Ship Date. In the next exercise, you will associate the new test script with this test case.

Exercise: Add the new script to a test case.

- 1 On the Planning tab of the Test Asset Workspace in TestManager, go to **Test Plans > ClassicsCD** and then double-click **ClassicsCD**.

The ClassicsCD test plan hierarchy appears.

- 2 In this window, go to **ClassicsCD > Purchase CD UC > UI > Order Confirmation Page > Display Estimated Ship Date**.
- 3 Right-click **Display Estimated Ship Date** and then click **Properties** on the shortcut menu.

The Properties dialog box for your test case opens.

- 4 On the Implementation tab, under **Automated implementation** click **Select** and then click **GUI - (Rational Test Datastore)** from the drop-down menu.

The Select Test Script dialog box appears.

- 5 Scroll to and click **Display Estimated Ship Date** and then click **OK**.
- 6 In the Test Case Properties window, click **OK**.
- 7 Choose **File > Save All**.

The test script is now associated with the test case.

Playing Back the Script on a New Build

Testers and developers can work in parallel. So, while testers build test scripts and suites, developers are typically creating new builds of the application. You recorded a script against Build 1, but now the developers have delivered Build 2 with changes to the UI. It is important that you run your suites on the newest build.

Testers or other members of your team may want to run test cases early in the project. Team members can right-click any test case in your test plan and run it from the shortcut menu.

For the purpose of this exercise, and because we are in the testing phase of Version 2 of the ClassicsCD.com development project, we will run the test script from the suite itself.

Exercise: Run the test suite on the newest build.

- 1 From the Execution tab in the Test Asset Workspace of TestManager, right-click **ClassicsCD Shop for CDs v1b** and then click **Run** on the shortcut menu.

The TestManager Run Suite dialog box appears. Notice that Build 1 is selected in the **Log Information** area.

- 2 Click **Change**. From the **Build** list, click **Build 2**. (Use the defaults for the other values.) Click **OK**.
- 3 Click **OK** in the Run Suite dialog box.

Note: Do not interact with the ClassicsCD.com application while TestManager processes your request to run the script and Robot plays back the scripts in the test suite! If you see a Windows message box that starts `Do you want Windows to remember . . .`, *wait. Robot will eventually continue.*

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 on the Details tab of a new test log.

- 4 In the Test Log window, click the Details tab to view the test results.

Analyzing the Results

TestManager shows which verification points passed and which failed. If a verification point fails and then its script also fails. You can inspect the test log in TestManager and decide how to handle any failures.

The script you recorded, Display Estimated Ship Date, passes despite the UI changes in Build 2. However, notice that other scripts have failed on this new build.

Handling Failures

The outcome of a particular test can 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.
- A real error is one in which the script fails with a correct baseline. To report a real error, submit a defect record using Rational ClearQuest, which is integrated with Rational TestManager.

Handling an Intentional Change

Exercise: Inspect the first failure.

- 1 In the Details tab of the Test Log window, go to **Computer Start (Shop for CDs v1b [1]) > TestCase Start (Purchase Items) > Script Start (Catlg-Purchase Items)** (see Figure 40).

Figure 40 Selecting the First Failure Using TestManager

Event Type	Result
Suite Start (ClassicsCD Shop for CDs v1b)	Fail
Computer Start (Shop for CDs v1b [1])	Fail
Script Start (Browser-Launch ClassicsCD v1b)	Pass
TestCase Start (Add Beethoven)	Pass
TestCase Start (Add Bach)	Pass
Script Start (Catlg-GoTo Cashier)	Pass
TestCase Start (Purchase Items)	Fail
Script Start (Catlg-Purchase Items)	Fail
Verification Point (Page Header - Object ...)	Fail
Verification Point (Order Summary - Objec...	Fail
Script End (Catlg-Purchase Items)	Fail
TestCase End (Purchase Items)	Fail
Script Start (Display Estimated Ship Date)	Pass
Script Start (Browser-Close)	Pass
Computer End	Fail
Suite End (ClassicsCD Shop for CDs v1b)	Fail

- 2 Double-click the failure, **Verification Point (Page Header - Object Data)**.

The Grid Comparator for the page header appears, showing that the page header changed from *Checkout* to *Cashier*. It turns out that this was a planned change and results from UI modifications.

In this case, you want to change the baseline so that the next time the script plays back, it compares the page header to the new value. (You change the baseline if a test fails because of an intentional change in the application.)

- 3 In the Grid Comparator window, click **File > Replace Baseline with Actual**.

The Grid Comparator displays a box asking if you want to confirm the replacement.

- 4 Click **Yes**.

The Grid Comparator updates the baseline and reports that there are no differences.

- 5 Close the Grid Comparator.

Handling a Real Error

Exercise: Inspecting another failure

- 1 In the Test Log window, double-click the next failure, **Verification Point (Order Summary - Object Data)**.

The Grid Comparator appears.

On line 2, the baseline shows that you were expecting to purchase a Beethoven Symphony, but the actual item placed into your shopping cart was a Mozart Symphony. This is a real error. Because these CDs are priced differently, this error generates a different purchase amount and the corresponding lines are also flagged as differences in the Grid Comparator.

- 2 Close the Grid Comparator.

Reporting the Error

To report the error, use ClearQuest and its integration with TestManager.

Exercise: Report the error

- 1 Right-click **Verification Point (Order Summary - Object Path)** and then choose **Submit Defect** on the shortcut menu.

If this menu command appears dimmed, ClearQuest is not installed on your computer and you cannot complete this exercise.

If ClearQuest is installed on your computer, the ClearQuest Login dialog box appears.

Note: Before you continue, make sure you completed the exercise on page 32, when you attached the change request database for the Webshop project.

- 2 In the ClearQuest Login dialog box:
 - a In both the **User Name** and **Password** boxes, type **pat**.
 - b From the **Database** list, ensure that **CLSIC** is the selected Database. If not, click **CLSIC: Rational Demo** from the list.

CLSIC is the name of the database that contains the change requests (defects and enhancement requests) for ClassicsCD.com.

- c Click **OK**.

ClearQuest opens a Submit Defect dialog box and automatically assigns a number to your defect. Red items indicate boxes where an entry is required: you cannot submit a defect until all required boxes contain valid values.

- 3 In the **Headline** box, type **Wrong item in shopping cart**.
- 4 From the **Severity** list, click **1-Critical** (see Figure 41).

Figure 41 Submitting a defect using ClearQuest

The screenshot shows a 'Submit Defect' dialog box with the following details:

- Title:** Submit Defect CLSIC00000130
- Tab:** Test Data
- ID:** CLSIC00000130
- State:** Submitted
- Headline:** Wrong item in shipping cart
- Suite Project:** Webshop
- UCM Project:** (empty)
- Owner:** (empty)
- Priority:** (empty)
- Severity:** 1-Critical
- Customer Priority:** (empty)
- Description:** Auto-Generated by TestManager
- Buttons:** OK, Cancel

- 5 Go to the Test Data tab. Notice that TestManager has already filled in boxes related to the test script for this defect.
- 6 Click **OK** to close the Submit Defect dialog box.
- 7 In the Test Log window, scroll to the right until the Defects column is visible. Notice that the defect you just entered has been automatically associated with the verification point.

You have finished testing this iteration of ClassicsCD.com.

Summary

For More Information

For more information about testing strategy, click **Help > Extended Help** from any Rational Test tool. In the Extended Help browser, read the content of interest to you.

To get started with Rational Test tools, see the *Rational TestManager User's Guide* and the *Rational Robot User's Guide*, both available on the *Rational Solutions for Windows – Online Documentation* CD-ROM.

Cleaning Up

Quit TestManager. If prompted to save any test results, click **Yes**.

If necessary, quit ClassicsCD.com.

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 Rational Robot and including verification points in your scripts.
- You can develop modular scripts and then use suites to call those scripts. You reuse scripts each time developers deliver a new software build.
- Robot makes it easy to address problems that are discovered during testing.
- The Rational ClearQuest integration with TestManager automates much of the error reporting process.

What's Next

Now that you have learned how to test an application for an iteration of development, you are nearly finished with the tutorial! In the next chapter, you plan the next iteration of ClassicsCD.com.

The ClassicsCD.com enhancement is now complete. You have finished work on this iteration. This chapter describes the steps you take to begin planning the next iteration.

Audience

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

Getting Your Bearings

In this chapter, you use Rational ClearQuest. To determine whether ClearQuest is installed on your computer, refer to the tool chart you filled out in Table 1, Rational Suite Tools, on page 27.

If ClearQuest *is not* installed, you can still benefit from reading this chapter, but you will not be able to perform the exercises.

If ClearQuest *is* installed, make sure you completed the exercise on page 32, when you attached the change request database for the Webshop project. Then:

- 1 Click **Start > Programs > Rational Software > Rational ClearQuest**.
- 2 In the ClearQuest Login dialog box:
 - a In both the **User Name** and **Password** boxes, type **pat**.
 - b From the **Database** list, ensure that **CLSIC** is the selected Database. If not, click **CLSIC: Rational Demo** from the list.

CLSIC is the name of the database that contains the change requests (defects and enhancement requests) for ClassicsCD.com.
 - c Click **OK**.

ClearQuest displays two panes. The left pane lists a hierarchy of charts and reports you can view. The right pane is blank.

Assessing the State of your Project

In Chapter 10, *Functional Testing*, you used 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.

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

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

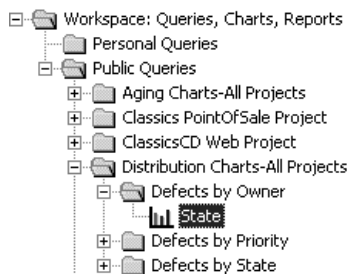
Showing the Workload

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. From a workload chart, you can drill down to information about a specific team member's workload. This feature can be helpful if you are interested in learning about the defects and related details assigned to an individual.

Exercise: Display a chart showing workload.

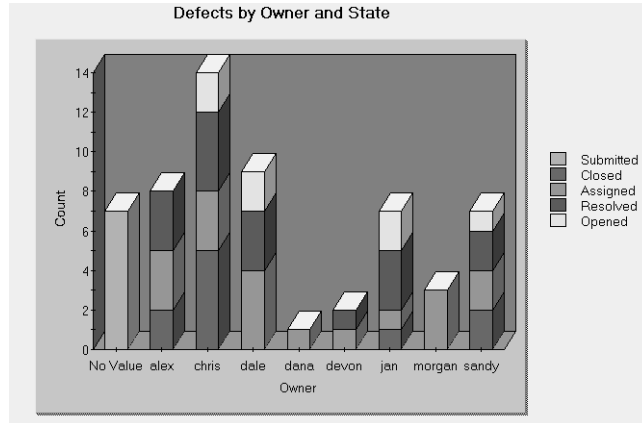
- 1 In the left pane of ClearQuest, go to **Public Queries > Distribution Charts-All Projects > Defects by Owner**.
- 2 Double-click the **State** entry (see Figure 42).

Figure 42 Choosing Query Views Using ClearQuest



ClearQuest displays the workload chart. If necessary, maximize the chart to see the details more clearly. The turquoise bar on the left represents unassigned defects (see Figure 43).

Figure 43 Viewing Defects and Workloads Using ClearQuest



- 3 Click once in the various regions of Jan's bar to learn more about the state and number of defects.
- 4 Double-click the turquoise bar (the leftmost bar) to list the defects summarized in that bar. These are defects that have not yet been assigned to team members.

ClearQuest displays a confirmation asking if you want to create a query.

- 5 Click **OK**.

ClearQuest lists the defects on the Result Set tab and displays details about the selected defect, so you can quickly drill down to details about a specific defect, and modify it as needed. This ClearQuest feature helps you plan a new iteration. For example, using this ClearQuest query, you can:

- View the unassigned defects
- Assign a defect (from the Main tab)
- Link them to requirements (from the Requirements tab).

- 6 Notice that defect ID CLSIC00000130, submitted in Chapter 10, *Functional Testing*, is included in this list (see Figure 44).

Figure 44 Viewing Defects Using ClearQuest

id	Owner	State
CLSIC00000073		Submitted
CLSIC00000074		Submitted
CLSIC00000084		Submitted
CLSIC00000088		Submitted
CLSIC00000089		Submitted
CLSIC00000091		Submitted
CLSIC00000092		Submitted
▶ CLSIC00000130		Submitted

Result set Query editor Display editor /

<input type="checkbox"/> PQC	<input type="checkbox"/> Test Data	<input type="checkbox"/> Environment	
<input type="checkbox"/> Unified Change Management	<input type="checkbox"/> ClearCase	<input type="checkbox"/> Requirements	Apply
<input type="checkbox"/> Main	<input type="checkbox"/> Analysis	<input type="checkbox"/> Notes	Revert
<input type="checkbox"/> Resolution	<input type="checkbox"/> Attachments	<input type="checkbox"/> History	Print Record

ID: State:

Headline:

Suite Project: Keywords:

UCM Project:

Owner:

Priority:

Severity: Symptoms:

Customer Priority:

Description:

***** Auto-Generated by TestManager *****

Actions ▼

◀ ID: 00000130 ▶

Working with Enhancement Requests

Recall that you started the tutorial by looking at an enhancement request that you later implemented. In the next exercise, you find more enhancement requests to implement in the next iteration.

Exercise: Examine the enhancement requests.

- 1 In the left pane of ClearQuest, go to **Public Queries > ClassicsCD Web Project** and then double-click **All Project Enhancement Requests**.
- 2 In the right pane, click *CLSIC00000036, Need to notify customer via email when order ships* to display details about this enhancement request. Notice that the enhancement request is in the **Submitted** state, but it has not yet been assigned. Click the History tab to learn more about the request and its history.
- 3 Now click *CLSIC00000031, Need to notify customer via e-mail when order ships* to display its details. Notice that the enhancement request is in the **Assigned** state. To see who is assigned to this request, click the Analysis tab.

Notice that both ClearQuest entries request the same enhancement for the online store. It is typical for several entries with a very similar, if not identical, enhancement request or defect to be stored in a change request database. This allows project teams to evaluate, or *triage*, each ClearQuest entry. Team members can determine more than just to whom requests or defects should be assigned, but also identify those entries which are duplicates of existing or new requirements, modify them to detail the duplication, and close them with this resolution description.

Exercise: Modify an enhancement request.

- 1 Go back to the previous enhancement request you looked at (*CLSIC00000036, Need to notify customer via email when order ships*). Since we know that a similar request has already been assigned, we'll close this request as a duplicate.
- 2 On the Main tab, click **Actions** and then click **Duplicate**.
- 3 In the Mark as Duplicate dialog box, enter **31** and then click **Find**.

Notice that the value in the Enter ID field changed to CLSIC00000031, which is the full ID of the enhancement request.

- 4 Click **Show more details** to make sure you have identified the correct record to which CLSIC00000036 is a duplicate.
- 5 When you are finished reviewing the data, click Cancel to return to the Mark as Duplicate dialog box.

- 6 Click **OK** to mark CLSIC00000036 as a duplicate of CLSIC00000031.
On tabs where you must fill in a box, a red square appears.
- 7 Go to the Analysis tab.
 - a The **Owner** box is marked red, indicating that a value is mandatory. Recall that you logged in as pat and you are determining the resolution for this entry. So from the **Owner** list, click **pat**.
- 8 Go to the Resolution tab. Notice that TestManager has already filled in boxes related to the duplicate of this record.
- 9 Click **Apply**. You may need to click **Run Query** from the toolbar to refresh the query results.

You will see that enhancement request *CLSIC00000036* is now in the *Duplicate* state.

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. You can also use ProjectConsole to automatically generate charts and gauges with metrics gathered from tools like RequisitePro and ClearQuest. This report helps you analyze the status of your next project and share it with members of your team.

What Will Happen in the Next Iteration?

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

- *All team members* use the Rational Unified Process (RUP) throughout the project as a guide for software development best practices, and as a source of information about software engineering.
- *Project leaders and managers* use the RUP Builder to refine their RUP configuration to better meet the needs of the next iteration.

- *All team members* use Unified Change Management (UCM) throughout the project to manage change in their system's development.
- *All team members* use the Rational Developer Network to access targeted development content, skill-building resources, and an online community of Rational Suite users.
- *Analysts* discuss 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.
- *Testers* use Rational TestManager to plan the tests for this iteration. The engineers create test plans, develop test requirements, and design tests.
- *Developers* use visual modeling techniques available in Rose to describe how planned enhancements fit within the system architecture.
- *All team members* use Unified Software Project Management (USPM) to compile information so they can assess status, trends, quality, and other aspects critical to project management and reporting.
- *All team members* use ProjectConsole and SoDA to gather project metrics and create project reports. Information presented in charts, gauges, and reports is useful during discussions with stakeholders and in design sessions.
- *Developers* use Rose to initiate implementation of the enhancement.
- *Developers and testers* use Purify, Quantify, and PureCoverage to verify the iteration's reliability.
- *Testers* use TestManager and Robot to verify that the enhancements meet requirements, that defects are fixed, and that no regression failures have occurred.
- *All team members* use ClearCase LT to make changes to project artifacts. Project members each work in private development workspaces. When team members finish their work, they deliver artifacts to the team's public integration workspace.
- *Project leaders and managers* use ProjectConsole, ClearQuest, SoDA, and RequisitePro to assess that state of the project from requirements through release. Later, they use these tools to plan subsequent iterations.

Summary

For More Information

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

Cleaning Up

When you are ready, quit ClearQuest.

If the Rational Unified Process is still open, either close it now or leave it open and use it as a supplement to learn more about Rational Suite on your own.

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.
- How the next iteration will proceed, and the activities that will transpire after planning the project.

What You Learned in This Tutorial

- 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 lifecycle and the primary roles on a development team – analysts, developers, testers, and managers.

What's Next

Congratulations! You have finished the Rational Suite tutorial. To send us any comments and suggestions that will help us improve our product documentation, go to <http://www.rational.com/contact/techpub.jsp>.

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.

Your next job is to learn more about the tools you will use on your next project and get to work!

Optional Activity: Join the Rational Developer Network

To access articles, discussion forums, and Web-based training courses on developing software with Rational Suite, we recommend that you join the Rational Developer Network.

- 1 If a current version of Rational Suite is installed on your computer, click **Start > Rational Software > Logon to the Rational Developer Network** and follow the registration instructions.
- 2 If a current version of Rational Suite is *not* installed on your computer, from your Web browser go to <http://www.rational.net> and follow the registration instructions.

After you register and log in to the Rational Developer Network, scan the Web site to learn more about the targeted resources, communication, and collaboration resources available. Using the Rational Developer Network will help you get started on your own projects using Rational Suite.

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.

baseline. A consistent set of artifact versions that represent a stable configuration for a project's components.

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

component. A nontrivial, 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.

discipline. The summary of all activities you can go through to produce a particular set of artifacts.

edition. Sets of Rational Suite tools that are customized for each functional area of a software development team.

element. An object that encompasses a set of versions for software project artifacts. Elements can be either files or directories.

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.

feature creep. A term used by software development teams to describe the tendency to add unplanned changes to product features throughout (and often late in) the development process.

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

implementation. The process of testing developed components as units and integrating the results into an executable system.

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 analysis and design, the implementation of an operation or procedure.

metrics. The measurements of project activity.

object. In object-oriented analysis and design, 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.

project. A project is a temporary endeavor undertaken to create a unique product or service. Temporary means that every project has a definite beginning and a definite ending. Unique means that the product or service is different in some distinguishing way from all similar products and services. Projects are often critical components of the performing organizations' business strategy. Projects are performed by people, constrained by limited resources, and planned, executed, and controlled.

project leader. A person who allocates resources, shapes priorities, coordinates interactions with the customers and users, and generally tries to keep the project team focused on the right goal. A project leader also establishes a set of practices that ensures the integrity and quality of project activities and artifacts.

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 change request management tool that helps users track any type of change activity – defects and fixes, enhancement requests, documentation changes, and so on – throughout the software development lifecycle. The ClearQuest Web interface allows users to perform all major ClearQuest operations.

Rational ClearQuest MultiSite. A highly customizable Windows and Web-based change request management tool that helps geographically distributed users track any type of change activity – defects and fixes, enhancement requests, documentation changes, and so on – throughout the software development lifecycle by easily replicating a centralized database to each remote site and then synchronizing the changes made at each site with changes made at other sites.

Rational Developer Network. A Web-enabled, searchable knowledge base that aggregates best practices, reusable artifacts and assets, and Web-based training to help software professionals expand their professional skills. The Rational Developer Network is available to Rational customers as a component of Rational Suite.

Rational Process Workbench. A highly customizable, Web-enabled, searchable knowledge base that enhances team productivity and delivers company-specific best practices using guidelines, templates, and Tool Mentors for critical software development activities.

Rational ProjectConsole. A highly customizable project management tool that helps users select and deploy best practices, plan and carry out iterative projects, and measure progress and quality throughout the software development lifecycle.

Rational PureCoverage. Automatically pinpoints areas of code in software applications that have not been tested.

Rational Purify. Automatically pinpoints hard-to-find run-time memory errors in software applications.

Rational Quantify. Automatically pinpoints performance bottlenecks in software 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, multiuser database. The RequisiteWeb interface allows users to perform all major RequisitePro operations.

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

Rational Rose. The world's leading visual component modeling and development tool; helps 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. Helps 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* and Rational Rose (Professional Data Modeler Edition).

Rational Suite DevelopmentStudio. Edition of Rational Suite optimized for software development. Contains the *Team Unifying Platform* 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* 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 edition includes the Rational Unified Process, RequisitePro, ClearCase LT, ClearQuest, SoDA, TestManager, and ProjectConsole.

Rational Suite TestStudio. Edition of Rational Suite optimized for testers. Contains the *Team Unifying Platform* and Rational PureCoverage, Rational Purify, Rational Quantify, Rational Robot, and Rational TestFactory.

Rational TestManager. Provides management and control of all test activities from a single, central point, including the ability to control and view legacy and proprietary test assets. It improves team productivity by making test results and progress toward goals immediately available to all team members.

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

real-time application. An application or system with stringent requirements for latency, throughput, reliability, and availability. Typically understood as representing operations which happen at the same rate as human perceptions of time.

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 creating or updating a Rose visual model from existing code, so that the visual model and code are kept in sync. 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. Traditional roles on a software development team include analysts, developers, testers, and managers or project leaders.

round-trip engineering. The ability to generate code from a Rose visual model (see *forward engineering*), and to update a Rose model file from source code (see *reverse engineering*).

stream. In UCM, this provides configuration instructions for your view (see *view*), and tracks activities and baselines (see *baselines*).

test case. A set of test inputs that describe a testable and verifiable behavior in a system, the extent to which you will test an area of an application, and the results of each test.

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

tester. 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 testing.

Tool Mentor. Step-by-step instructions on how to use specific Rational tools to perform activities as described in the Rational Unified Process.

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

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

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 the main, alternate, and exception flows.

Unified Software Project Management (USPM). The Rational approach to managing software projects, from requirements to release. USPM spans the development lifecycle, focusing on compiling information to assess status, trends, quality, and other aspects critical to project management and articulation of progress.

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

view. A ClearCase LT object that provides a work area for one or more users to modify source versions.

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 within a discipline to attain an observable value.

Index

A

- activity 42, 49, 137
 - and ClearCase LT 52
- actor 44, 137
- analyst 21, 137
 - tools 63
- AnalystStudio 21, 139
- architecture
 - component-based 17
 - visual modeling 86
- artifact 42, 49, 61, 72, 137
 - and ClearCase LT 52
 - managing change 49
- artifacts 30
- automated testing 105, 137

B

- baseline 52, 137
 - promoting 52
- budget and predictability 38
- builds 18

C

- change control 18
- change set 52
- change, managing 18
- child requirement 65
- class 137
 - class diagram 91
 - identifying (Rose) 91
- ClassicsCD.com
 - installing 28
 - overview 28
 - running 33
- ClearCase LT 20, 49, 50, 96, 101, 138
 - and ClearQuest 51

- and UCM 51
 - Web interface 20
- ClearQuest 20, 34, 50, 96, 101, 127, 138
 - and ClearCase LT 51
 - and Robot 121, 123
 - assessing project status 128
 - attaching database to a Rational project 31
 - schema 128
 - starting 127
 - Web interface 20
- ClearQuest MultiSite 20, 138
- code, implementing 92
- component 17, 137
- component-based architecture 137
 - designing (Rose) 17
- configuration management 20, 49, 137
- Customer Support xiii

D

- database (RequisitePro) 61
- defect reporting 121, 123
- designing
 - tests 79
- designing component-based architecture 17
- developer 21, 137
 - tools 63
- developing software
 - See* software development
- development stream (ClearCase LT) 51
- development view (ClearCase LT) 51
- DevelopmentStudio 21, 139
 - RealTime Edition 139
- diagram window (Rose) 63, 85, 91
- discipline 42, 137
- disciplines 39
- document (RequisitePro) 61
- documentation
 - resources xi

E

edition 137
element 137
Enterprise Edition
 Rational Suite 139
 Rose 22
error reporting 121, 123
Extended Help 68, 137

F

feature creep 137
forward engineering 137
functional testing 111, 112

G

GUI Record toolbar 115

H

Help
 Rational Customer Support xiii
 resources xi
Help, Extended 68, 137

I

implementations, in test 76, 138
implementing code 92
installing tutorial sample application 28
integration stream (ClearCase LT) 51
integration view (ClearCase LT) 51
iteration 41
iterative development 16, 138
 and Rational Unified Process 40

J

Java
 and PureCoverage 107
 and Quantify 108
 and Rose 92
joining a project 51

L

link
 requirements and defects 129
 suspect, RequisitePro 67
 traceability, RequisitePro 67

M

managing 96
managing change 18, 49
managing requirements
 See requirement, managing
managing risk 72
managing software changes 18
measuring
 project status 96
memory leaks 106
method 138
metrics 138
Microsoft Project (RequisitePro) 82
Microsoft Visual Basic
 and PureCoverage 107
 and Quantify 108
 and Rose 92
Microsoft Visual C++
 and PureCoverage 107
 and Purify 106
 and Quantify 108
 and Rose 92
Microsoft Word
 and RequisitePro 59

O

object 138
 identifying in Rose 91

P

parent requirement 65
performance testing 23
 code 22
 system 23, 81
performance testing, code 108
phase 41, 138

- planning a script, TestManager 115
- playing back a script 120
- prerequisites of tutorial 27
- process
 - See* Rational Unified Process
- Professional Data Modeler Edition
 - Rose 21
- project 138
- project leader 138
- project metrics (ProjectConsole) 99
- project state, assessing 128
- project status 96
- ProjectConsole 99, 139
- PureCoverage 22, 81, 105, 107, 139
 - and Java 107
 - and Microsoft Visual Basic 107
 - and Microsoft Visual C++ 107
- Purify 22, 81, 105, 106, 139
 - and Microsoft Visual C++ 106

Q

- quality engineer, role of 23
- quality, verifying
 - See* testing
- QualityArchitect 22
- Quantify 22, 81, 105, 108, 139
 - and Java 108
 - and Microsoft Visual Basic 108
 - and Microsoft Visual C++ 108

R

- Rational 24
- Rational Administrator 138
 - attaching change request database to a Rational project 31
 - registering a Rational project 30
- Rational ClearCase LT
 - See* ClearCase LT
- Rational ClearQuest
 - See* ClearQuest
- Rational Developer Network 21, 135, 138
- Rational Process Workbench 25, 138
- Rational project 30, 51
 - datastore 32
 - register 30

- Rational PureCoverage
 - See* PureCoverage
- Rational Purify
 - See* Purify
- Rational Quantify
 - See* Quantify 22
- Rational Robot
 - See* Robot
- Rational Rose 63
 - See* Rose
- Rational SoDA
 - See* SoDA
- Rational Software, mission 15
- Rational Suite 139
 - AnalystStudio 21, 139
 - benefits 18, 19
 - ContentStudio 139
 - DevelopmentStudio 21, 139
 - DevelopmentStudio - RealTime Edition 139
 - documentation roadmap xii
 - Enterprise Edition 139
 - summary table 25
 - Team Unifying Platform 19, 139
 - TestStudio 23, 139
 - tools 19, 27
- Rational Synchronizer 139
- Rational TestFactory
 - See* TestFactory
- Rational TestManager 139
 - See* TestManager
- Rational Unified Process 18, 19, 37, 140
 - Extended Help 68
 - overview 38
 - phases and iterations 40
 - starting 37
 - Tool Mentor 45
- real-time application 140
- rebasing a stream (UCM) 52
- registering a Rational project 30
- reports, creating (SoDA) 96
- requirement 57, 140
 - and change 58
 - and vision document 58
 - child 65
 - managing 17, 55, 58, 140
 - parent 65
 - types 67

- RequisitePro 55, 77, 96, 101, 139
 - and Rose 63
 - database features 61
 - document features 61
 - Explorer 56
 - integration with Microsoft Project 82
 - starting 55, 82
 - test planning 71
 - Tool Palette 55
 - Views 61
 - Web interface 19
 - Word document 59, 61
- resetting tutorial 30
- reverse engineering 140
- risk management 72, 140
- roadmap
 - Rational Suite Documentation xii
- roadmap,
 - tutorial 35
- Robot 23, 81, 107, 111, 139
 - and ClearQuest 121, 123
 - GUI Record toolbar 115
 - playing back a script, Robot 120
 - reviewing test results 121
 - starting 113
- role 42, 140
- roles 41
- Rose 63, 77, 85, 92, 96, 101, 139
 - and RequisitePro 63
 - browser 63, 85, 91
 - class diagram 91
 - diagram window 63, 85, 91
 - Enterprise Edition 22
 - Java 92
 - Microsoft Visual Basic 92
 - Microsoft Visual C++ 92
 - Professional Data Modeler Edition 21
 - RealTime 23
 - starting 62, 85
 - Web publishing 90
- round-trip engineering 140
- run-time errors 106
- RUP
 - See* Rational Unified Process

S

- schedule
 - predictability 38
 - test efforts 81
- schema 128
- script
 - planning (TestManager) 115
 - playing back, Robot 120
 - shell 118
- sequence diagram 86
 - and use case 86
 - class 87
 - message 87, 88
 - object 88
- shell script 118
- SoDA 20, 95, 96, 139
 - starting 95
 - template 99
- software 18
- software development
 - common problems 15
 - component-based architecture 17
 - controlling change 18
 - iterative development 16
 - managing requirements 17
 - verifying quality 18
- software engineer, role of 21
- SQABasic 115
- stream, in ClearCase LT 140
- suspect link (RequisitePro) 67
- system performance 23
- system testing 23

T

- Team Unifying Platform 19, 139
- test
 - suite 112
- test case 72, 73, 75, 76, 112, 140
 - designing 79
 - folder 73, 75
- test configuration 72, 140
- test input 72, 140
- test plan 73, 140

- test planning 71
 - creating scripts 115
 - identifying risks and resources 80
 - scheduling 81
 - test types 81
- test script 112
- tester 23, 140
- TestFactory 24
- testing
 - code performance 22, 108
 - coverage 107
 - functional 111, 112
 - memory leaks 106
 - reliability 105
 - run-time errors 106
 - system performance 23, 81
 - verification point 116
 - verifying quality 18
- TestManager 20, 72, 76, 81, 96, 101, 111, 112, 139
 - and Robot 121
 - starting 111
- tests
 - types of 81
- TestStudio 23, 139
- Tool Mentor 45, 140
- Tool Palette (RequisitePro) 55
- tools (Rational Suite) 19
- tools in Rational Suite 27
- traceability 140
 - links 67
- tutorial
 - ClassicsCD.com 28
 - prerequisites 27
 - resetting 30
 - roadmap 35
 - sample application 28
 - setting up 28
 - tool checklist 27

U

- UML, *See* Unified Modeling Language 21
- Unified Change Management 20, 140
- Unified Change Management (UCM) 49
- Unified Modeling Language 17, 21, 26, 62, 86, 141
- Unified Process, *See* Rational Unified Process
- Unified Software Project Management 96
- use case 44, 141
 - and sequence diagram 86
 - and visual modeling 59
 - benefits to team 59
 - report 95, 97
- use case diagram 61
 - working with 63
- use case report 97
- use case requirement 61

V

- verification point 116
- verifying software quality, *See* testing
- version control 141
- view 141
- view (RequisitePro) 65
- Views (RequisitePro) 61
- vision document 141
 - and requirements 58
- visual model 141
 - Web version (Rose) 90
- visual modeling 85, 86
 - implementing code 92
 - maintaining consistency with code 92

W

- Web Development 53
- Word document (RequisitePro) 61
- workflow 42, 141

