Rational Suite Tutorial

Product Version	Rational Suite 2000.02.10
Release Date	April 2000
Part Number	800-023316-000

Rational®
the e-development company™

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

Contents

Preface

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.

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

Audience

Read this tutorial if you:

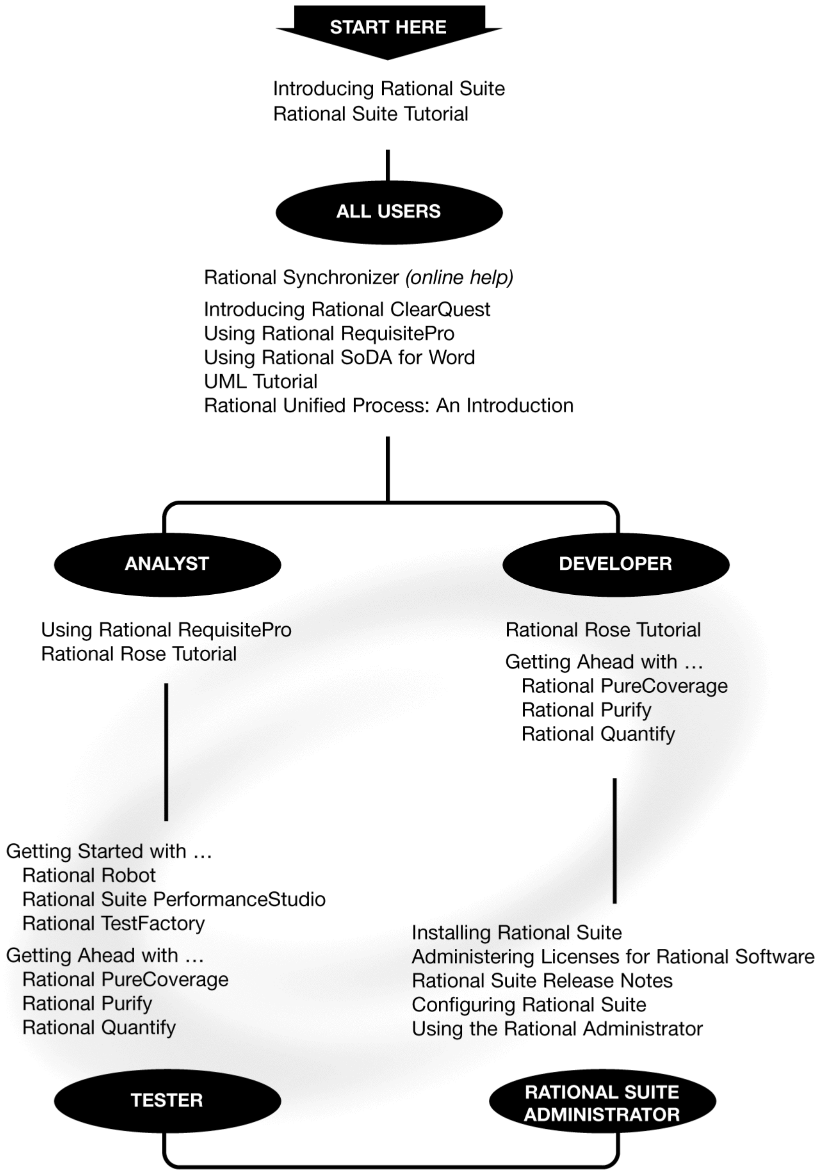
- Are a member of a development team – an analyst, developer, test engineer, or manager
- Have experience with some aspect of Windows application development

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

Other Resources

- Online Help is available for Rational Suite. From a Suite tool, select an option from the **Help** menu.
- All manuals are available online, either in HTML or PDF format. The online manuals are on the Rational Solutions for Windows Online Documentation CD.
- If you install Rational Suite DevelopmentStudio – RealTime Edition, PDF versions of the manuals for Rose RealTime are installed in `%ROSERT_HOME%\help`.
- For more information on training opportunities, see the Rational University Web site: `http://www.rational.com/university`.

Rational Suite Documentation Roadmap


START HERE
Introducing Rational Suite
Rational Suite Tutorial
ALL USERS
Rational Synchronizer *(online help)*
Introducing Rational ClearQuest
Using Rational RequisitePro
Using Rational SoDA for Word
UML Tutorial
Rational Unified Process: An Introduction
ANALYST
Using Rational RequisitePro
Rational Rose Tutorial
DEVELOPER
Rational Rose Tutorial
Getting Ahead with …
Rational PureCoverage
Rational Purify
Rational Quantify
Getting Started with …
Rational Robot
Rational Suite PerformanceStudio
Rational TestFactory
Getting Ahead with …
Rational PureCoverage
Rational Purify
Rational Quantify
TESTER
Installing Rational Suite
Administering Licenses for Rational Software
Rational Suite Release Notes
Configuring Rational Suite
Using the Rational Administrator
RATIONAL SUITE ADMINISTRATOR

Contacting Rational Technical Publications

To send feedback about documentation for Rational products, please send e-mail to our technical publications department at `techpubs@rational.com`.

Contacting Rational Technical Support

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

Rational Technical Support Information

Location	Contact Information	Notes
U.S. and Canada	800-433-5444 781-676-2450 support@rational.com	When sending e-mail: – Specify the product name in the subject line, for example, “Rational Suite”. – For existing issues, include your case ID in the subject line.
Europe	+31 (0) 20 4546 200 support@europe.rational.com	
Asia Pacific	+61-2-9419-0111 support@apac.rational.com	
World Wide Web	http://www.rational.com	Click the Technical Support link.

1 Welcome to Rational Suite

Does your organization focus on developing and delivering software?

If so, think about your last project. Was it on time? Within its budget? Was communication among team members clear and timely? Did your team maintain consistency throughout the project as it defined requirements, developed designs, 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 result often doesn't match the requirements. Many projects uncover serious design flaws late in the process. Defects are often found after the software ships, instead of during development.

How can you make your next project more successful?

Principles of Software Development

Rational Software Corporation, the e-development company, helps organizations develop and deploy software for e-business, e-infrastructure, and e-devices through a combination of tools, services and software engineering best practices. Rational's e-development solution helps organizations overcome the e-software paradox by accelerating time to market while improving quality.

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

Develop software iteratively. Iterative development means analyzing, designing, and implementing incremental subsets of the system over the course of a project. Early in a project, new executable files reduce project risk. As the project proceeds, later executable files are more robust or contain more features than previous executable files. Each new iteration moves you closer to the goal of delivering a product that meets its requirements.

Developing iteratively helps make your project more predictable, lets you collect feedback early, helps you identify and eliminate risks early in the project, and makes testing continuous throughout the project lifecycle.

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

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

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

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

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

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

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

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

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

Control changes to your software. It is important to manage changes to requirements and code in a trackable, repeatable, predictable way. Change management includes tracking and handling change requests; facilitating parallel work; and creating processes for reliably reproducing software builds.

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

Rational Suite Can Help

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

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

What's in Rational Suite?

Rational Suite provides sets of tools tailored for every member of your team. Each Suite edition contains team-unifying tools and tools designed for a specific practitioner. The following sections describe these tools and the Suite editions.

Tools that Unify Your Team

Each edition of Rational Suite contains the following tools to facilitate team communication and productivity:

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

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

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

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

Tools for Analysts

An analyst's role is to:

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

Rational Suite AnalystStudio

Rational Suite *AnalystStudio*, the Rational Suite edition designed for analysts, contains the team-unifying tools (**Rational Unified Process**, **RequisitePro**, **ClearQuest**, and **SoDA**) and:

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

Tools for Developers

A developer's role is to:

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

Rational Suite provides two editions for developers: Rational Suite DevelopmentStudio and Rational Suite DevelopmentStudio – RealTime Edition.

Rational Suite DevelopmentStudio

Rational Suite *DevelopmentStudio*, the Rational Suite edition designed for system developers and designers, contains the team-unifying tools (**Rational Unified Process**, **RequisitePro**, **ClearQuest**, and **SoDA**) and:

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

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

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

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

Rational Suite DevelopmentStudio – RealTime Edition

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

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

- Create executable models, allowing you to compile and observe simulations of your UML designs on the host or target platform. The result is that you can refine your design early and you can continually verify quality.

- Generate complete, deployable executables in C or C++ directly from UML design models targeted to real-time operating systems. Generating these applications eliminates the need for manual translation and avoids costly design interpretation errors.

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

Tools for Test Engineers

A test engineer's role is to:

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

Rational Suite provides two editions for test engineers: Rational Suite TestStudio and Rational Suite PerformanceStudio.

Rational Suite TestStudio

Rational Suite *TestStudio*, the Rational Suite edition designed for test engineers, contains the team-unifying tools (**Rational Unified Process**, **RequisitePro**, **ClearQuest**, and **SoDA**) and:

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

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

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

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

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

Rational Suite PerformanceStudio

Rational Suite *PerformanceStudio* is the Rational Suite edition designed for test engineers who develop and run performance tests, and for architects who want to test their architectural design early in the development process. PerformanceStudio contains the team-unifying tools (**Rational Unified Process**, **RequisitePro**, **ClearQuest**, and **SoDA**), the tools in TestStudio (**Robot**, **TestFactory**, **Purify**, **PureCoverage**, and **Quantify**) and:

Rational LoadTest. Helps you create real-world, multi-user performance tests for testing the performance and reliability of e-business, multi-tier, and database applications. Also helps you execute functional tests faster by distributing those tests to multiple computers and coordinating the results.

Rational PerformanceArchitect. Helps you test the performance of COM and DCOM applications. With Rational PerformanceArchitect, you create Rose sequence or collaboration diagrams, generate virtual user scripts from your diagrams, and then use PerformanceStudio to edit the script and run the performance tests.

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

To learn more about PerformanceStudio, see *Getting Started with Rational Suite PerformanceStudio*.

Rational Suite Enterprise

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

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

For More Information

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

For more information about the Unified Modeling Language, visit the UML Resource Center at:

`http://www.rational.com/uml`

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

2 About this Tutorial

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.

Prerequisites

Before continuing, make sure you have:

- An edition of Rational Suite installed on your computer
- A valid license for the software you've installed

Determining Which Rational Suite Tools Are Installed

Table 1, Rational Suite Tools, lists the Suite tools you will use in this tutorial.

Table 1: Rational Suite Tools

❑ **Rational ClearQuest**	❑ **Rational Robot**
❑ **Rational PureCoverage**	❑ **Rational Rose**
❑ **Rational Purify**	❑ **Rational SoDA for Word**
❑ **Rational Quantify**	❑ **Rational TestFactory**
❑ **Rational RequisitePro**	❑ **Rational Unified Process**

Exercise You may want to mark this page so you can refer back to it from later chapters. Place check marks in the table next to the tools that are installed on your system. To determine whether a tool is installed, click the **Start** button and see whether the tool's name is in the **Programs > Rational Suite** menu.

Some of these tools may not be installed on your computer. A tool may be excluded because of the Suite edition you bought. Someone in your organization may have chosen not to install other tools.

Installing the Tutorial Sample Application and Related Files

Before you try the tutorial exercises, you need to install and set up the files you will use. (The files are already on your system.) To install the files:

1. Make sure you have 30 MB of free disk space. If you are using Windows NT, make sure you have Administrator privileges so that you can complete the setup successfully.
2. From the **Start** menu, choose **Programs > Rational Suite > Rational Suite Tutorial > Setup**.
3. Follow the instructions that appear on your screen.

   The files are installed into the directory
   `c:\Program Files\Rational\Classics Demo`

Tip: Resetting the Tutorial

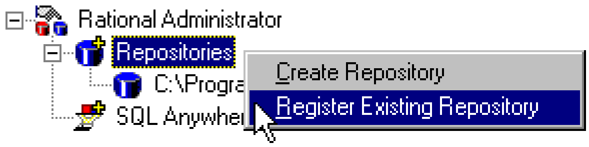
If you decide to restart the tutorial, reset the sample application and related files by running the installation procedure. For example, you might restart the tutorial to review material you've already worked on.

Registering the Repository

You use the Rational Administrator to group a set of projects associated with Rational Suite (for example, a RequisitePro database and a Rose model). The Rational Suite terminology for such a group is a *repository project*. To register the repository project:

1. Start Rational Administrator by clicking **Start** and choosing **Programs > Rational Suite > Rational Administrator**.
2. In the left pane of the Rational Administrator, right-click **Repositories**. From the shortcut menu, choose **Register Existing Repository**.

Registering the Repository

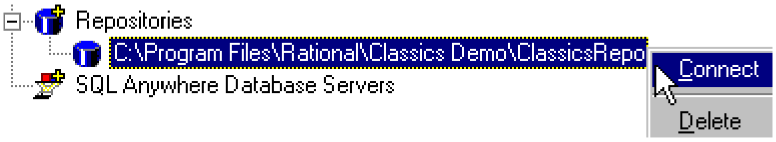


3 In the Register Existing Repository dialog box, browse to `c:\Program Files\Rational\Classics Demo\ClassicsRepo` and click **OK**.

The Administrator adds the path under the **Repositories** heading.

4 In the left pane of the Administrator, right-click the path you just added. From the shortcut menu, choose **Connect**.

Connecting to the Repository


Repositories
C:\Program Files\Rational\Classics Demo\ClassicsRepo
SQL Anywhere Database Servers
Connect
Delete

5 In the Rational Repository Login dialog box, use the default values (User ID is `admin`; Password is blank). Click **OK**.

You have now registered and connected to the repository you will work with during the tutorial. Keep the Rational Administrator open for the next task.

Attaching the ClearQuest Database to the Repository

In this tutorial, you work with a ClearQuest database that contains the tutorial project's change requests (defects and enhancement requests). Attach this database to the repository you just registered:

1 Check whether Rational ClearQuest is installed on your system by referring to Table 1, "Rational Suite Tools," on page 25. If it *is* installed, 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, choose **Tools > Rational ClearQuest Maintenance Tool** to display a ClearQuest wizard that helps with setup tasks.

3 On the wizard's first page, under Setup schema repository, use the default option, **Connect to Existing schema repository**. Click **Next**.

4 On the Connect to An Existing Schema Repository page:

- Make sure the value in the **Vendor** field is **MS_ACCESS**.
- In the Physical Database Name box, browse to `c:\Program Files\Rational\Classics Demo\ClassicsRepo\ChangeRequests\CQMaster.MDB`
- Click **Next**.

5 On the Save Current Setting for Later Use page, make sure that **Save currents settings as Default** is cleared. Click **Finish**.

6 Review messages to confirm that you have connected the ClearQuest database to the repository. Click **Done**.

7 Quit the Rational Administrator.

About the Work You Will Do

In this tutorial, you implement a small part of a large development project. Using Rational tools and process, you develop requirements, create a visual model, code, and test. This book guides you to the sections that are most appropriate for you and your role in your own organization.

Tutorial Background

In this tutorial, you work for *Classics Inc.*, a growing chain of retail stores that sells classical music CDs. Your organization is working on Version 2 of ClassicsCD.com, an online store where you can buy classical music CDs.

Your team uses Rational Suite. During the tutorial, you add one new feature to ClassicsCD.com.

A Note about the Application

In a real online store, Web pages would load dynamically, based on information stored in databases. The application you work with during the tutorial 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.

Ordering CDs

Start by becoming familiar with ClassicsCD.com

Exercise Start Classics and order two CDs:

1. Start the application: click **Start** and choose **Programs > Rational Suite > Rational Suite Tutorial > Classics CD Build 0**.

   Your Web browser displays the first page of ClassicsCD.com.
2. On the home page, click **Explore our storefront**.
3. On the storefront page, click **Catalog**.
4. Scroll to the Mozart section (composers are listed alphabetically) and click **Mozart: Symphonie Nr. 34** to view details of that album.
5. On the album's page, click the shopping cart 🛒 to add the album to your order.

   The catalog page reappears.
6. At the top of the page, click the shopping cart next to **Bach: Brandenburg Concertos 1 + 3**.

Finishing the Purchase

Exercise Now complete the purchase and provide feedback to *Classics Inc*.

1. In the left column of the page, click **Shopping Cart** (you may need to scroll to see it).
2. On the left column of the Shopping Cart page, click **Cashier**.

   Before you can complete the order, you need to log in.
3. In both the CustomerID and Password fields, type `jmoore` and click **Submit**.

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

4 Scroll to the bottom of the page and click **Place Order**.

5 Provide feedback to the company by clicking **Your Feedback** at the bottom of the page.

6 On the feedback form, under Dear ClassicsCD.com, type

```
When I place an order, I want to know when my order will ship.
```

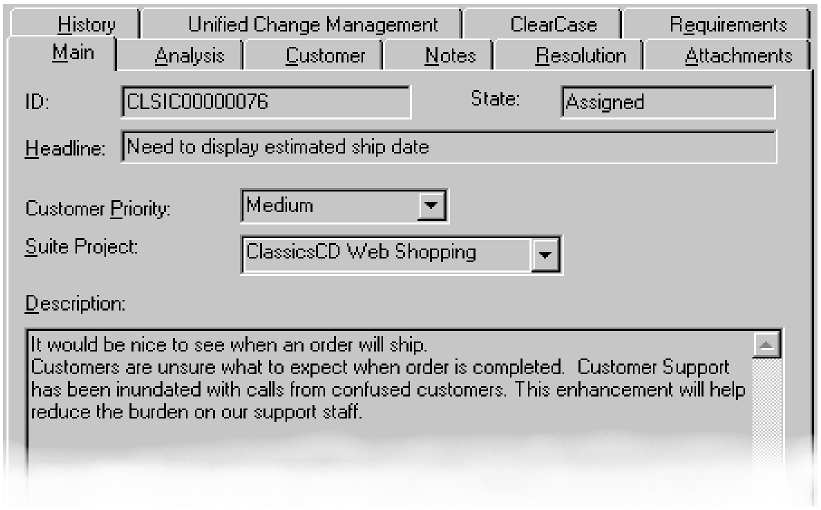
Under My e-mail, type `jmoore@clicker.com`.

Click **Send**.

7 When you have finished, quit ClassicsCD.com.

Discovering What to Build

In this tutorial, you will implement the enhancement you requested. Someone in marketing received your feedback and entered it into ClearQuest, the tool that manages change requests.



ID: CLSIC00000076
State: Assigned
Headline: Need to display estimated ship date
Customer Priority: Medium
Suite Project: ClassicsCD Web Shopping
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.

In Chapter 10, *Planning the Next Iteration*, you work with ClearQuest.

How to Use this Tutorial

You can work through the entire tutorial to understand how Rational Suite fits into your development environment. Or you can work through only those units most appropriate to your role. Table 2, Tutorial Roadmap, presents our recommendations.

Table 2: Tutorial Roadmap

Role	Your Main Tasks	Recommended Tutorial Units
Analyst	Represent the user Determine *what* the system does Specify and manage requirements	1 Welcome to Rational Suite 2 About this Tutorial 3 Learning About the Rational Unified Process 4 Creating Requirements 7 Creating a Use Case Report 10 Planning the Next Iteration
Developer	Determine *how* the system works Define architecture Create, modify, and manage code	1 Welcome to Rational Suite 2 About this Tutorial 3 Learning About the Rational Unified Process 4 Creating Requirements 6 Modeling the Enhancement 7 Creating a Use Case Report 8 Reliability Testing 10 Planning the Next Iteration
Test Engineer	Ensure requirements are met Create, manage, and execute tests Report results and verify fixes	1 Welcome to Rational Suite 2 About this Tutorial 3 Learning About the Rational Unified Process 4 Creating Requirements 5 Test Planning 7 Creating a Use Case Report 8 Reliability Testing 9 Functional Testing 10 Planning the Next Iteration

What’s Next

The next unit introduces you to the Rational Unified Process. You use this process to learn about work you will do in later units.

Let’s get started!

3 Learning About the Rational Unified Process

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

Audience

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

Getting Your Bearings

In this unit, you use the Rational Unified Process. Refer to the tool chart you filled out (Table 1, Rational Suite Tools, on page 25) to determine whether the Unified Process is installed on your system.

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

If the Unified Process *is* installed, start it now by clicking the **Start** button and then choosing **Programs > Rational Suite > Rational Unified Process**.

What Is the Rational Unified Process?

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

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

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

The Unified Process is implemented as an online guide and knowledge base, which you view with a Web browser.

The Unified Process and Rational Suite

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

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

This tutorial follows Unified Process guidelines.

Learning the Mechanics

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

Exercise From the Getting Started window, you can navigate to pages containing tips about the mechanics of using the Unified Process. You can also find pages that provide starting points for learning about the process itself.

1. On the Getting Started window, click **Browsing the Process**.

   Your Web browser displays the page, *Navigation Tools*, which contains a layout of the Unified Process.

2. Familiarize yourself with the user interface by moving your mouse over the picture and reading the tool tips that are displayed.
3. Optionally, return to the Getting Started window and read about other aspects of the Unified Process.
4. Leave your Web browser open.

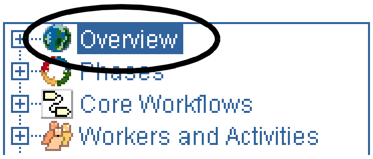
The Process at a Glance

The first page of the Unified Process presents a rich visual representation that can help you better understand the Unified Process.

Exercise Return to the first page:

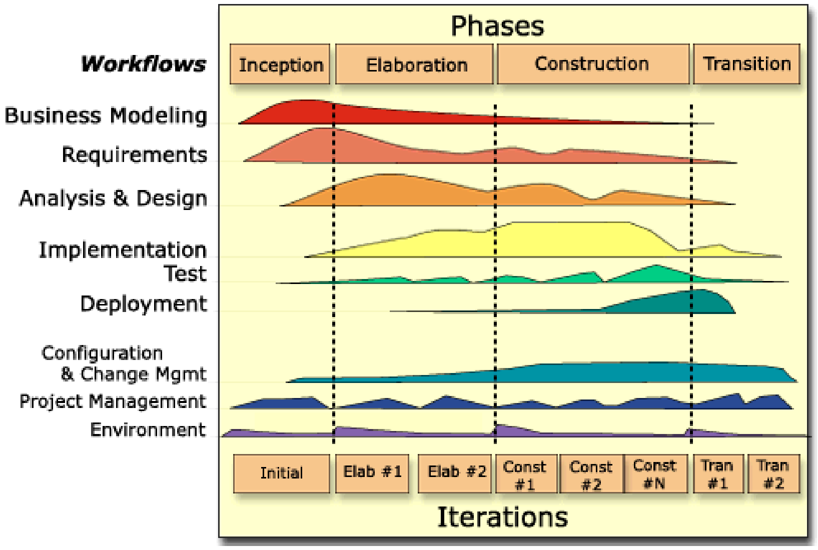
1 In your Web browser, click **Overview** in the upper left corner of the tree browser.

Returning to the Overview Page



The overview of the Unified Process appears. This diagram represents empirical data collected by Rational Software. It shows that software development is best performed in a series of iterations.

Rational Unified Process Overview



Over time, your software project will require different types of focus and resources. 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 becomes most intense at the end of construction.

Key Concepts

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

Exercise Read about the key concepts of the Unified Process.

1 In the tree browser, expand **Overview** by clicking the **+** to its left.

2 Under **Overview**, click **Key Concepts**. (From now on, we'll abbreviate these two steps as follows: "Navigate to **Overview > Key Concepts**.")

A new Web page appears. The diagram at the top of the page shows the relationships among Rational Unified Process concepts.

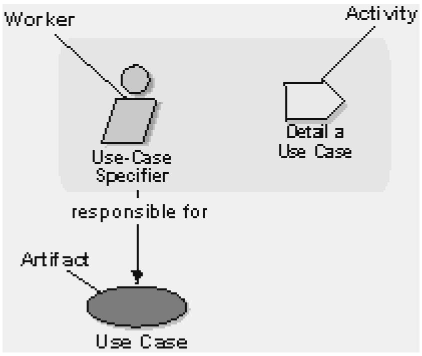
3 Under the diagram, scroll to the first section, *Software Engineering Process*, and read it for a quick summary of the Unified Process.

4 Continue reading to learn about other key concepts in the Process.

To summarize, the Rational Unified Process defines the following important concepts:

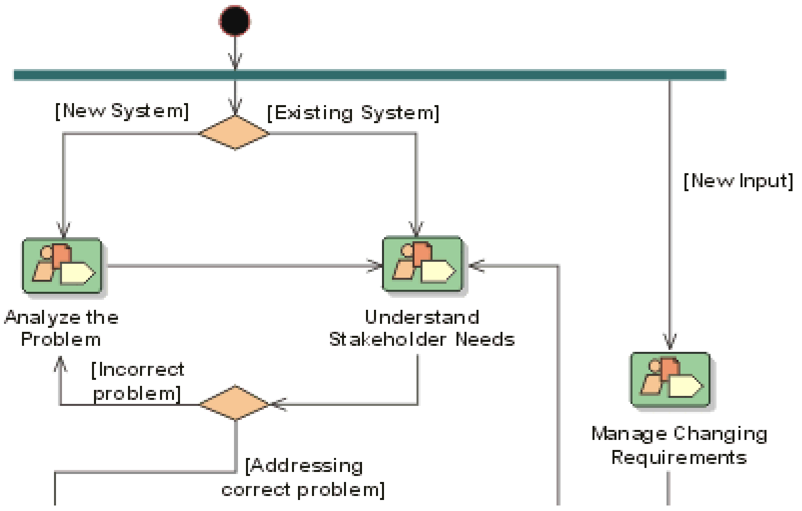
- A *worker* defines the behavior and responsibilities of an individual or a team of individuals. One person may act as many workers over the course of a project. A worker is responsible for a set of artifacts.
- An *artifact* is something a worker produces as the result of performing an activity. In the Unified Process, the artifacts produced in one activity are often used as input into other activities. An artifact can be small or large, simple or complex, formal or informal. Examples of artifacts are: a test plan, a vision document, a model of a system's architecture, a script that automates builds, or application code.
- An *activity* is a unit of work that a worker performs. It is a set of ordered steps, like a recipe, for creating an artifact.

Relationship between a Worker, its Activities, and Artifacts



- A *workflow* is the sequence of activities that workers perform toward a common goal. The following diagram shows a portion of the Requirements workflow.

Excerpt of Requirements Workflow


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

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

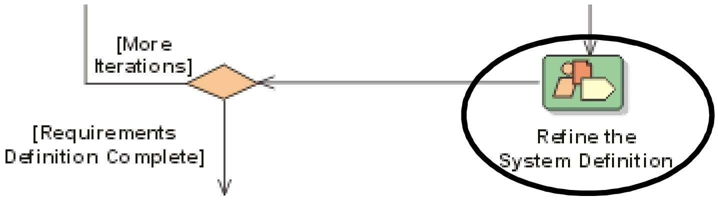
Exploring the Workflow

Exercise Display the workflow diagram for Requirements:

1 In the tree browser, navigate to **Core Workflows > Requirements** to display the Requirements workflow diagram.

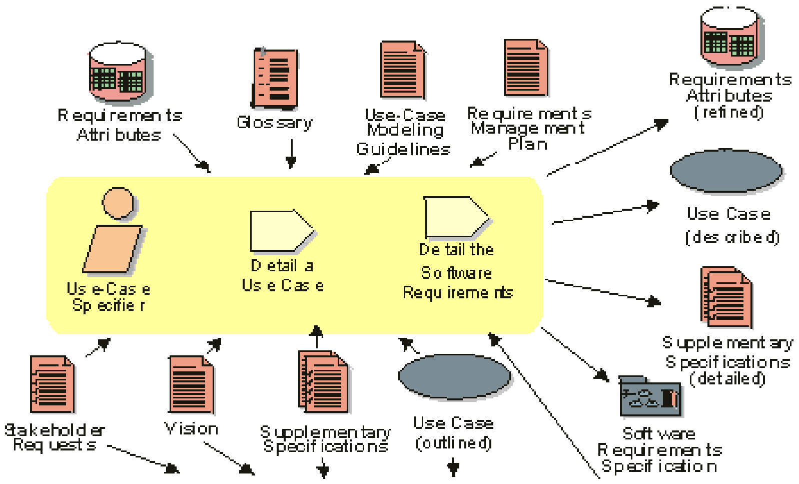
During this tutorial, you work on refining the existing application, ClassicsCD.com. Rational Unified Process provides guidance on how to enhance an existing system.

2 On the diagram, scroll to the bottom of the diagram and click **Refine the System Definition** to display workflow details.


[More Iterations]
[Requirements Definition Complete]
Refine the System Definition

The workflow detail shows the workers involved, the artifacts used as input, the resulting artifacts, and the activities that make up this part of the workflow. For more information about any of these elements, click in that area of the diagram.

Workflow Detail: Refine the System Definition (excerpt)


Requirements Attributes
Glossary
Use-Case Modeling Guidelines
Requirements Management Plan
Requirements Attributes (refined)
Use Case (described)
Supplementary Specifications (detailed)
Software Requirements Specification
Use-Case Specifier
Detail a Use Case
Detail the Software Requirements
Stakeholder Requests
Vision
Supplementary Specifications
Use Case (outlined)

Starting with Actors and Use Cases

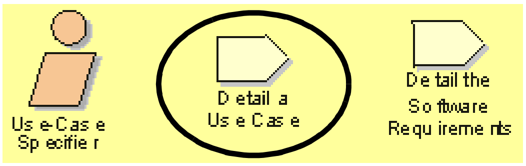
When you design or enhance a system, the Unified Process recommends that members of your team start by agreeing on the system's high-level behavior. To do so, you identify actors and use cases.

- *Actors* are the entities that interact with your system. An actor is usually a person (for example, an end-user or administrator). It can also be an external hardware or software system, for example, a cash register or warehouse system.
- *Use cases* describe how an actor uses and interacts with the system. More formally, a use case describes what services a system provides to a certain actor. You define a use case by describing a complete sequence of actions that yields observable results of value to an actor.

  Use cases are a key concept in the Unified Process. They enhance communication across your team and help you communicate clearly with your users.

Exercise Learn more about working with use cases.

1 On the Refine the System Definition workflow, click **Detail a Use Case**.



The Unified Process displays a page describing how to write a use case. It includes details about the artifacts that you'll need to get started and the artifacts that result from the activity. It then provides a step-by-step description of the activity.

2 Say you want to learn more about use cases, not just about how to write them. In the Resulting Artifacts section near the top of the page, click **Use Cases, completely described**.

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

Near the top of the page, there's a link to a use case template. We recommend that you use this template, or another template designed by your group, to ensure consistency and completeness. This makes it easy for all stakeholders to locate and understand important project information.

3 Click your browser's **Back** button to return to the Detail a Use Case page.

4 At this point, you might want to understand where you are in the Unified Process hierarchy. In the main window, click **Where am I**. The tree browser updates itself to show your current location – **Workers and Activities > Analysts > Use-Case Specifier > Detail a Use Case**.

Tool Mentors: Implementing the Process Using Rational Tools

The Unified Process provides guidelines for all phases of software development. This section shows how the Unified Process uses Tool Mentors to provide step-by-step instructions for implementing Unified Process practices with Rational tools. To illustrate, we show you the instructions for the work we will do in the next unit: defining use cases.

Exercise Read a Tool Mentor to see how the Unified Process integrates with Rational tools.

1 In the Unified Process, on the Detail a Use Case page, find the Tool Mentors section in the table near the top of the page.

2 Click **Using Rational RequisitePro to Detail a Use Case**.

The Tool Mentor starts with a statement of purpose and an overview, followed by a series of detailed steps.

3 In the overview, scroll to and click Step 3, **Mark requirements in the Detailed Use-Case Specification**.

Scan through the instructions; you will perform a subset of these steps in the next unit.

Learning about Developing for the Web

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

Exercise Read about developing for the Web.

1 In the Unified Process, navigate to **Overview > Roadmaps > Developing e-business Solutions**.

2 Scan the guidelines to learn more about each phase of the development process.

Summary

For More Information

To learn more about the Rational Unified Process, read the topics on the Getting Started page. (To return to the Getting Started page, click **Getting Started** on any Unified Process page.)

Cleaning Up

When you have finished using the Rational Unified Process, close it and the Getting Started window.

What You Learned in this Unit

In this unit, you learned:

- The Rational Unified Process contains best practices for software development. In Rational Suite, the Unified Process is recommended but optional.
- A workflow describes a set of related activities focused on meeting a goal. For each activity, a worker 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 Unified Process activity using the appropriate Rational tool.

What’s Next

In the next unit, you work on the use case for the enhancement request to ClassicsCD.com, Arrange Shipment.

4 Creating Requirements

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

Audience

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

Getting Your Bearings

In this unit, you use RequisitePro and Rose.

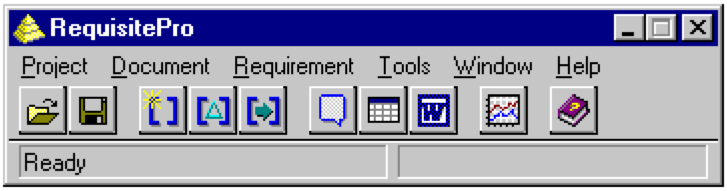
Refer to the tool chart you filled out (Table 1, Rational Suite Tools, on page 25) to determine whether RequisitePro and Rose are installed on your system.

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

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

Click the **Start** button and choose **Programs > Rational Suite > Rational RequisitePro**. The RequisitePro Tool Palette and Let's Go RequisitePro appear. Click **Close** on the Let's Go RequisitePro window.

The RequisitePro Tool Palette



Opening the ClassicsCD Web Shop Project

1. On the RequisitePro Tool Palette, click **Project > Open**.
2. In the Select a Project area, click **Add**.
3. In the Add Project dialog box, navigate to `c:\Program Files\Rational\Classics Demo\ClassicsRepo\Project\ClassicsCDW\requisite\Requisite.RQS`
4. Click **Open**.

   The dialog box now lists the *ClassicsCD Web Shop* project. The next time you start RequisitePro, the project will be in the list and you will not have to add it again.
5. Select **ClassicsCD Web Shop** and click **OK**.
6. If you see the Project Logon dialog box, choose or type **Pat** and click **OK**.

   The RequisitePro Tool Palette displays messages about its progress. Once the project opens, the Tool Palette's status bar displays the message *Ready*. You are now ready to work with the project.

Why Worry About Requirements?

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

Where Do Requirements Come From?

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

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

RequisitePro

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.
- *Test engineers* use requirements to design tests and check test coverage.
- *Managers* use RequisitePro to define project work based on available time, budget, and personnel.

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.

Starting with a Use Case

In Unit 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 requirement corresponding to your 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. Use cases are important to everyone on the project:

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

Exercise Use RequisitePro to open the use case document.

1 On the RequisitePro Tool Palette, click **Document > Open** to display the Open Project and Documents dialog box.

2 In the Select Documents list, select **Arrange Shipment**. Click **OK**.

The RequisitePro Word Workplace opens and displays the use case document. This document is based on a template provided with the Rational Unified Process. (In the previous unit, you saw, and may have clicked on, a link to this template.)

3 In the document, scroll to Section 1, *Arrange Shipment*.

Text with double-underlining identifies requirements. These *use case requirements*, identified by the prefix *UC*, are high-level requirements that describe the system’s behavior.

4 Read the Brief Description and Flow of Events. Notice that the shipping date enhancement has been recorded (it starts with “The warehouse...”) but it has not yet been identified 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’s requirements. You also indicate which requirements are related (through parent-child or traceability relationships). RequisitePro then tracks how changes to the system affect your requirements and how changes to requirements affect your system.

5 For now, minimize Word and the RequisitePro tool palette.

How Does it Work?

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

- 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 using the RequisitePro Word Workplace.
- The *database* features help you organize your requirements, prioritize your work, track requirements changes, and share information with other Rational tools. To work with database features, you use the Views Workplace.

How Does RequisitePro Handle Requirements?

In the use case document, the requirements text (double-underlined) 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 unit, we work with RequisitePro database features.

Continuing Use Case Work: Using Rose

In this section, you continue work on the use case – a high-level description of how an actor interacts with the system – as the first step in implementing the feature requirement.

Starting Rose

Rose is a visual modeling tool that uses the Unified Modeling Language (UML) and is integrated with common code development environments.

Exercise To start Rose:

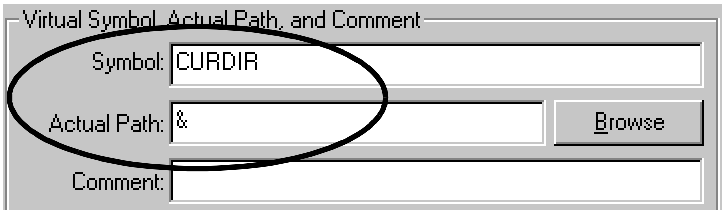
1 From the **Start** menu, click **Programs > Rational Suite > Rational Rose**.

Rose appears and displays the Create New Model dialog box.

2 Quit the Create New Model dialog box.

3 Configure Rose the first time you use it:

- From the Rose menu, select **File > Edit Path Map**.
- In the Symbol field, type `CURDIR`.
- In the Actual Path field, type `&`.
- Click **Add**.
- Click **Close**.



Symbol: CURDIR

Actual Path: &

These steps allow you to open a Rose model on your machine even if the model was created on a different machine.

4 From the Rose menu, select **File > Open** and navigate to `c:\Program Files\Rational\Classics Demo\ClassicsRepo\Project\ClassicsCDW\Rose`

5 Select `ClassicsCD_tutorial.mdl` and click **Open**.

6 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.

Creating a Use Case in Rose

Now that you know which feature to implement, you want to create a use case in Rose and add it to an existing use case diagram. A use case diagram shows:

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

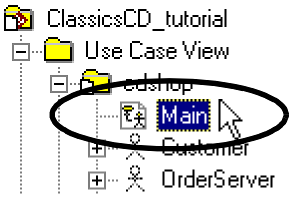
Exercise Add to the use case diagram.

1. In the Rose browser, expand **Use Case View** by clicking the **+** to its left. Then expand **cdshop**.

   Notice the actors (stick figures) and use cases (ovals) listed under **cdshop**.

2. Double-click **Main** under **cdshop** to display the use case diagram. Maximize the diagram within the Rose window.

   Double-click Main under cdshop

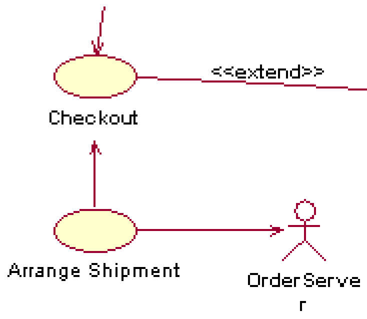


3. Create the Arrange Shipment use case:
   - In the Rose browser, right-click **cdshop**.
   - From the shortcut menu, choose **New > Use Case**.
   - Immediately type its name, `Arrange Shipment`, exactly as shown, and press ENTER.
   - If the use case does not appear on the diagram, drag it from the Rose Browser to the diagram, placing it below **Checkout**. (You can drag the blue note out of the way.)

4. On the diagram, select the arrow between **Checkout** and **Order Server**. Press DELETE to delete the arrow.

5. On the diagram toolbox (between the Rose browser and the diagram window):
   - Click the **Unidirectional Association** button ↱.
   - On the use case diagram, drag from the new **Arrange Shipment** use case to the **Checkout** use case.
   - Click the **Unidirectional Association** button ↱.
   - On the use case diagram, drag from the **Arrange Shipment** use case to the **Order Server** actor.

Portion of Updated Use Case Diagram



Associating the Rose Model with the RequisitePro Project

Earlier in this unit, you opened the Classics project in RequisitePro. In this section, you associate the Rose model you are modifying with that project in RequisitePro. You also link the use case you just created with the use case requirement that already exists in RequisitePro.

Exercise Associate the Rose model and the RequisitePro project:

1. On the use case diagram, right-click the **Arrange Shipment** use case. From the menu, click **Use Case Document > Associate**.
2. On the Associate Document to Use Case 'Arrange Shipment' dialog box, select **Arrange Shipment**. Click **OK**.

   RequisitePro appears, displays the Arrange Shipment use case document, and highlights the Arrange Shipment use case requirement you saw earlier in this unit.

   The Arrange Shipment use case is now represented both in words and by a visual model. The use case is now a single element.

Creating a New Requirement

Arrange Shipment is an established use case, but you still need to identify a requirement for the enhancement request.

Exercise Create the requirement.

1. In the RequisitePro Word Workplace, scroll to the end of the basic flow. Select the entire sentence beginning “The warehouse system responds...”
2. From the Word menu, select **RequisitePro > Requirement > Create**.

   RequisitePro displays the Requirement Properties dialog box.
3. On the Attributes tab, from the Property list, select **Basic Flow** to indicate that this new requirement is part of the basic flow of a use case.
4. On the Hierarchy tab:
   - From the Parent list, select **<choose parent...>**.
   - From the Parent Requirement Browser, select **UC7: Arrange Shipment**. Click **OK**.
5. On the Requirement Properties dialog box, click **OK**.

   RequisitePro underlines the requirement and marks it as pending.
6. From the Word menu, select **RequisitePro > Document > Save**.

   When RequisitePro saves the document, it finishes creating the requirement. Scroll up to the new requirement and verify that it has been assigned the number UC7.2.

Looking at Requirements in the Database

In this section, you use the RequisitePro database to view 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 Open a view into the database, using the Views Workplace.

1. In the Word Workplace, click **RequisitePro > Window > Show Views**.

   The RequisitePro Views Workplace opens.
2. From the RequisitePro Views menu, click **File > Open View**.

3 Under Attribute Matrix Views, select **UC: All Use Cases** and click **OK**.

The use cases in the left column describe the functional areas of ClassicsCD.com. The use cases are organized hierarchically, where each child requirement is listed under its parent. Parent requirements are more general, while child requirements describe more specific areas. To see children of a use case, click on the **+** next to its name.

4 Verify that the ship-date requirement was added by clicking the **+** next to UC7: Arrange Shipment.

5 Scroll to the right to see each requirement’s attributes and their properties.

Linking to Another Requirement

So far, you have:

- Decided to implement a new enhancement.
- In Rose, added a new use case to 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 link the use case requirement 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 Link the use case requirement to a feature requirement:

1 From the RequisitePro Views menu, click **File > Open View**.

2 Under Traceability Matrix Views, select **UC-Feat: Trace Matrix showing Features and Use Cases** and click **OK**.

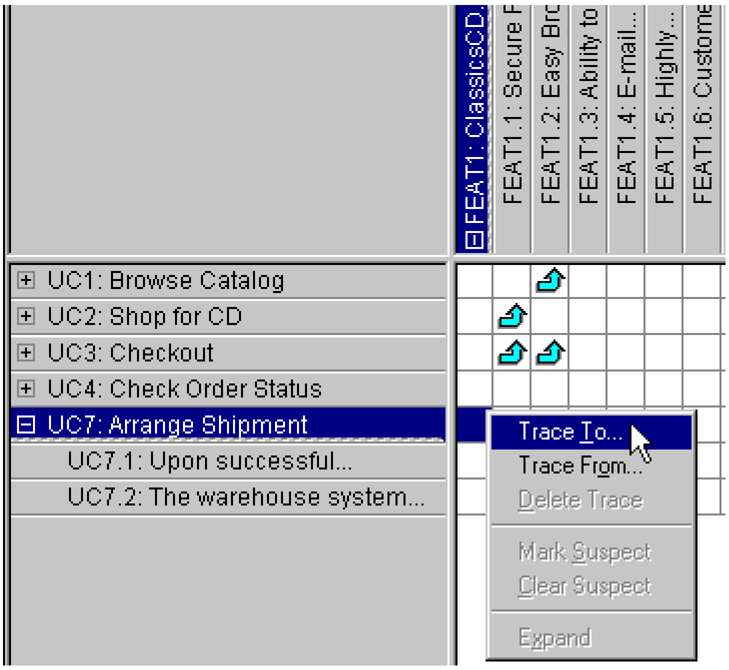
The view displays, showing the entire hierarchy of requirements.

3 From the Views workplace menu, choose **View > Collapse All**.

4 Along the top row, expand FEAT1 by clicking the **+** under it.

5 Expand UC7 by clicking the **+** to its left.

6 Right-click the cell at the intersection of FEAT1 and UC7. From the menu, select **Trace To**.



An arrow appears in the cell, showing the link relationship.

You have just used the RequisitePro traceability feature to create a link from the use case requirement, UC7, to the feature requirement, FEAT1. In the next section, you learn more about traceability.

Traceability Links and Suspect Links

The matrix in the Views Workplace 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 when a requirement in the link relationship changes. An analyst needs to examine the changes, and decide whether to edit the dependent requirements before clearing the suspect link.

Other Requirement Types

So far, we've discussed 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 requirements for performance targets and platform support.

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, funders, and so on). As you capture requirements, you check your work with the appropriate stakeholders. When the stakeholders and your team come to agreement, your initial job of gathering requirements is done.

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

Extended Help

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

Exercise You can see Extended Help from any RequisitePro window:

1 On the Views Workplace menu, click **Help > Extended Help**.

The Rational Extended Help window opens. The window has two panes. The left pane contains a tree browser and the right pane is blank.

2 Move the vertical divider to widen the left pane.

3 In the left pane, expand **Tool Mentors** (click the **+** next to it) and double-click **Detailing a use case**.

Extended Help displays the same tool mentor that you viewed in Unit 3, "Learning About the Rational Unified Process." Read this tool mentor to review the work you've done in this unit.

Extended Help provides information about the higher-level tasks you may want to accomplish. It gives you direct access to the Rational Unified Process from the Rational Suite tools. In addition, you can add your own organizational guidelines or standards to Extended Help.

For more information about this feature, on the Extended Help browser (the left pane), navigate to **About Rational Extended Help > Extended help overview**.

Summary

For More Information

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

For more information about Rose, see *Modeling the Enhancement* on page 67.

Cleaning Up

Quit Extended Help.

Quit RequisitePro by choosing **Project > Exit** from the RequisitePro Tool Palette. RequisitePro asks if you're sure you want to close the project. Click **Yes**.

Quit Rose by choosing **File > Exit** from the Rose menu. When Rose asks whether to save changes, click **Yes**.

What You Learned in this Unit

In this unit, 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 requirements.
- All members of your team benefit from using RequisitePro to manage requirements.
- RequisitePro is both document-centric and database-centric, allowing your team to benefit from the strengths of both.
- To create a use case, you work in Rose to add the use case to your model, then work in RequisitePro to add textual descriptions.
- You have finished writing the first set of requirements when your stakeholders and your team agree that you're done.
- Extended Help gives you immediate access to process and task information. You can add your own information to Extended Help.

What's Next

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

5 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.

Audience

This unit applies to test engineers.

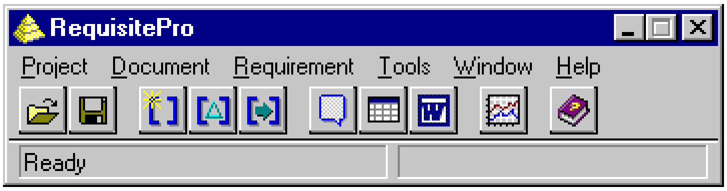
Getting Your Bearings

In this unit, you use Rational RequisitePro. Refer to the tool chart you filled out (Table 1, Rational Suite Tools, on page 25) to determine whether RequisitePro is installed on your system.

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

If RequisitePro *is* installed, start it now by clicking the **Start** button and choosing **Programs > Rational Suite > Rational RequisitePro**. The RequisitePro Tool Palette and Let's Go RequisitePro appear. Click **Close** on the Let's Go RequisitePro window.

The RequisitePro Tool Palette



Note This unit assumes you are familiar with RequisitePro concepts and benefits. If you are not, please read Unit 4, "Creating Requirements," before continuing.

Opening the ClassicsCD Web Shop Project

Open the ClassicsCD Web Shop project:

1. On the RequisitePro Tool Palette, click **Project > Open**.

   If the Select a Project list contains *ClassicsCD Web Shop*, skip to step 6. Otherwise continue with the next step to add the project to the list.

2. In the Select a Project area, click **Add**.
3. In the Add Project dialog box, navigate to `c:\Program Files\Rational\Classics Demo\ClassicsRepo\Project\ClassicsCDW\requisite\requisite.rqs`
4. Click **Open**.

   The dialog box now lists the project *ClassicsCD Web Shop*. The next time you start RequisitePro, the project will be in the list and you will not have to add it again.

5. If you see the Project Logon dialog box, choose or type **Pat** and click **OK**.
6. Select **ClassicsCD Web Shop** and click **OK**.

   Once the project opens, the Tool Palette's status bar displays the message *Ready*. You are now ready to work with the project.

What Is Test Planning?

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

Test planning starts early in the development cycle, as soon as you understand the initial set of requirements. As with development, test planning is an iterative process. You will continue to plan testing throughout a project's lifecycle, as analysts change or elaborate on requirements, and as developers design and implement code.

Risk Management

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

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

Test Planning and the Tutorial

In this unit of the tutorial, you perform a small piece of test planning – tracing test requirements. You also look at a test specification for the ClassicsCD.com enhancement.

Working with Test Requirements

You start test planning by identifying test requirements. You then associate test requirements with their corresponding use case requirements (for example, the use case requirement you worked with in Unit 4, “Creating Requirements”). A use case requirement focuses on the details of system behavior. A test requirement focuses on what to test in a specific area of the application.

Exploring Test Requirements

In this section, you look at the existing test requirements for Classics. You work with a traceability matrix, which shows the relationship between use case requirements and test requirements.

Exercise Look at the test requirements for ClassicsCD.com.

1. From the RequisitePro Tool Palette, click **Window > Show Views** to display the RequisitePro Views Workspace.
2. From the RequisitePro Views menu, click **File > New View** to display the New View dialog box.

3 In the View Type area, click **Traceability Matrix**.

4 In the Row Requirement Type list, select **UC: Use Case Requirement Type**.

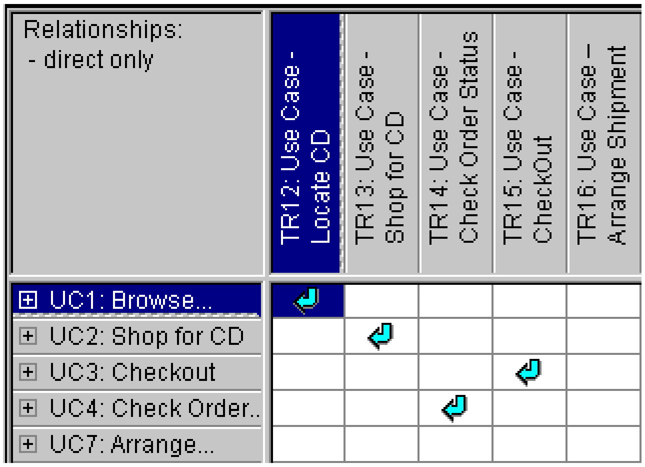
5 In the Column Requirement Type list, select **TR: Test Requirement Type**. Click **OK**.

The view displays an expanded list of use cases.

6 Collapse all the use cases by clicking **View > Collapse All**.

If you want, resize the table's heading rows.

Use Case Requirements Related to Test Requirements



Relationships: - direct only	TR12: Use Case - Locate CD	TR13: Use Case - Shop for CD	TR14: Use Case - Check Order Status	TR15: Use Case - CheckOut	TR16: Use Case – Arrange Shipment
⊞ UC1: Browse...	↲				
⊞ UC2: Shop for CD		↲			
⊞ UC3: Checkout				↲	
⊞ UC4: Check Order..			↲		
⊞ UC7: Arrange...					

This table shows how the test requirements are related to the use case requirements. The turquoise arrows show traceability relationships. For example, one arrow shows that TR12 is traced to UC1. Notice that no test requirement is traced to UC7.

If Use Case 1 changes, a red slash is drawn through the arrow (↲), indicating that the traceability relationship is suspect. A suspect relationship means that one or both requirements in the relationship have changed. An analyst needs to examine the changes, and decide whether to edit the requirements before clearing the suspect link.

You need to establish traceability at a high level only. During test planning, it is unnecessary to establish test requirements at lower levels of detail.

Exercise Add a traceability link for UC7.

1 On the view, right-click in the cell between UC7: Arrange Shipment and TR16: Use Case Arrange Shipment.

2 From the menu, select **Trace From**.

You have now established a link between the use case and the test requirement.

Designing Tests

Each test requirement describes a specific area of the application to test. However, each area can encompass a broad class of situations that you need to test. For example, in testing a cash sales transaction, you would probably test:

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

How do you determine what to test? This part of test planning – test analysis and design – requires you to rely on your own human intelligence and experience.

Starting with the Use Case

When you design tests, the first step is to examine the original use case to understand how the system is supposed to behave. You then create test requirements and trace them to the use cases. During analysis, you identify the conditions you need to test to verify that:

- Code implementing the use case does what *is* intended.
- Code implementing the use case does *not* do what is *not* intended.

Creating Test Cases

A test case describes the extent to which you will test an area of the application. It lists the preconditions for performing a test, the input to provide during testing, the variables you will examine, and the expected results of each test.

Many organizations find it useful to create a chart showing the results of the test analysis and design. By examining the chart, you quickly get information about areas that may require further testing.

Exercise Look at a sample test case.

1 On the RequisitePro Tool Palette, click **Document > Open** to display the Open Project and Documents dialog box.

2 In the Select Documents list, select **Test Cases for Use Case – Arrange Shipment**. (You may need to scroll to see it.) Click **OK**.

3 In the document, scroll to Section 1, *Functional Test Cases*.

   This document presents a test design for the *Arrange Shipment* use case. Read the first section for an overview of the tests to perform for this use case.

4 Scroll to the next section to view a table describing the tests. Each test is linked to a test case and describes any prerequisites and expected results.

Continuing with Test Planning

Writing a test plan is an iterative process that starts early in the project. It continues as analysts change requirements and elaborate on use cases, as developers design and write code, and as testers discover 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
- Scheduling

Risks and Resources

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

We recommend that you prioritize tests as follows:

- **Must test (high)** – You must run this test to avoid severe risk or to identify weak project areas early in the development cycle. You cannot complete project testing without completing this test.
- **Should test (medium)** – You should schedule this test, but in a resource crunch, could consider not running it.
- **Could test (low)** – This test might be useful to run, but is not essential to the project. Run this test if you cannot make further progress on other, more important, tests.
- **Won't test (low)** – This test is not part of the testing project. A test with this priority defines the boundaries of the test plan and helps focus attention on what will be tested.

Types of Tests

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

- **Functional tests –** Does the application meet its functional requirements? Use Rational Robot for functional testing.
- **Reliability tests –** Can the application run without errors? Use Rational TestFactory for reliability testing.
- **Performance tests** – Is the application's performance acceptable under varying loads? Use Rational Suite PerformanceStudio for performance testing.

Scheduling

Part of creating a test plan involves writing a schedule. You work with team leaders from other areas of the project to understand when their contributions will be ready for testing. You then need to balance your original schedule against the risks and resources you identified in order to arrive at the most effective schedule.

If you prioritized your tests as described in “Risks and Resources” on page 63, make sure you at least schedule the “must” (high priority) and “should” (medium priority) tests. If resources become more constrained over the course of the project, you can sacrifice tests of lower priority without compromising the absolute quality objectives expressed by the “must” tests.

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

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

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

Summary

For More Information

For more information about test planning:

- Read about test plans. Starting in the Rational Unified Process tree browser, click **Artifacts > Test Set > Test Plan**.
- Read “Planning Your Tests” in *Using Rational Robot*.
- For a more in-depth treatment of test planning, read *Testing Computer Software* (Vnr Computer Library) by Cem Kaner and others (ISBN: 1850328471).

Cleaning Up

Quit RequisitePro by choosing **Project > Exit** from the RequisitePro Tool Palette. RequisitePro asks if you’re sure you want to close the project. Click **Yes**. If you are prompted to save documents, click **Yes**.

What You Learned in this Unit

In this unit, 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 requirements and relating them to use case requirements.
- 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 unit, you use Rose to work on an architectural model for the enhancement you are working on.

6 Modeling the Enhancement

So far, you have defined requirements for the ClassicsCD.com enhancement. The test organization has started test planning. In this unit, you continue to incorporate designs for the enhancement into the ClassicsCD.com visual model.

Audience

This unit applies to software designers and developers.

Getting Your Bearings

In this unit, you use Rational Rose. Refer to the tool chart you filled out (Table 1, Rational Suite Tools, on page 25) to determine whether Rose is installed on your system.

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

If Rose *is* installed, start it now:

1 Click the **Start** button and choose **Programs > Rational Suite > Rational Rose**.

2 If you did not perform the exercises in Chapter 4, *Creating Requirements*, do the following:

   - Perform the steps under *Starting Rose* on page 47.
   - Resume this procedure at step 5.

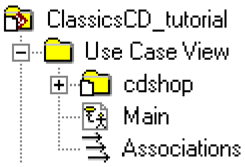
3 In the Create New Model dialog box, click the Existing tab, and navigate to:
`c:\Program Files\Rational\Classics Demo\ClassicsRepo\`
`Project\ClassicsCDW\Rose`

4 In the Model Files area, select `ClassicsCD_tutorial.mdl`. Click **Open**.

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.

5 In the Rose browser, expand Use Case View by clicking the **+** to its left. You should see **Main** and **cdshop**.

Use Case View, expanded



What Is Visual Modeling?

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

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

Visual Modeling and the Tutorial

Rational Rose supports visual modeling. In Unit 4, "Creating Requirements," you used Rose and RequisitePro to create a use case. In this unit, you continue working on the design for the enhancement.

Working with a Sequence Diagram

In this task you work with a sequence diagram, 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 classes

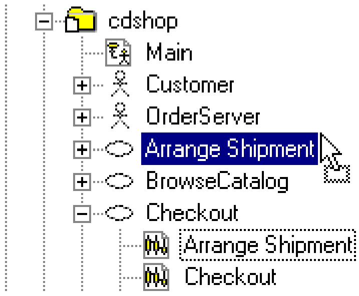
Opening a Sequence Diagram

Start by looking at an existing sequence diagram.

Exercise Open the sequence diagram now.

1. In the Rose browser, navigate to **cdshop > Checkout > Arrange Shipment**.
2. If you did not perform the exercises in Chapter 4, *Creating Requirements*, do the following:
   - In the Rose browser, right-click **cdshop**.
   - From the shortcut menu, choose **New > Use Case**.
   - Immediately type its name, `Arrange Shipment`, and press ENTER.

3 You want the sequence diagram to appear under the Arrange Shipment use case, so drag it to the use case:



4 Double-click the **Arrange Shipment** sequence diagram to open it.

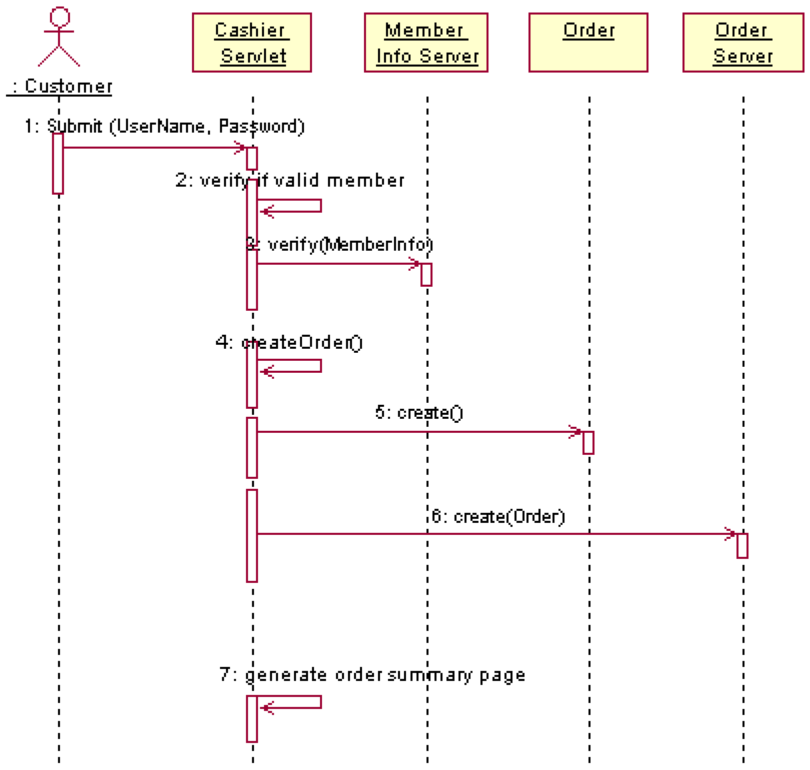
The sequence diagram appears.

Tip: Click the hand in the lower right corner of the diagram window to see a bird's eye view of the diagram and to easily reposition the window's contents.

5 Maximize the diagram, then resize it so you can it all at once. (You may need to select **View > Zoom Out** to view the entire diagram.)

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

Arrange Shipment Sequence Diagram



The first few messages mean that:

1 The Customer starts the use case by sending a Submit message to the Cashier Servlet.

2 The Cashier Servlet sends a message to itself to verify whether the Customer is a valid member.

3 The Cashier Servlet then sends a verify message to the Member Info Server.

4 The Cashier Servlet then sends a `createOrder` message to itself.

Adding Messages for the Enhancement

In this section, you add messages to the sequence diagram for the enhancement you are working on.

Exercise 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 on the diagram on the Cashier Servlet lifeline (the vertical line), just under Message 6.

Notice that Rose renumbers the messages on the diagram.

3 While the object is still selected, type `getShipDate`. Then click the background of the diagram.

We are following this naming convention for objects:

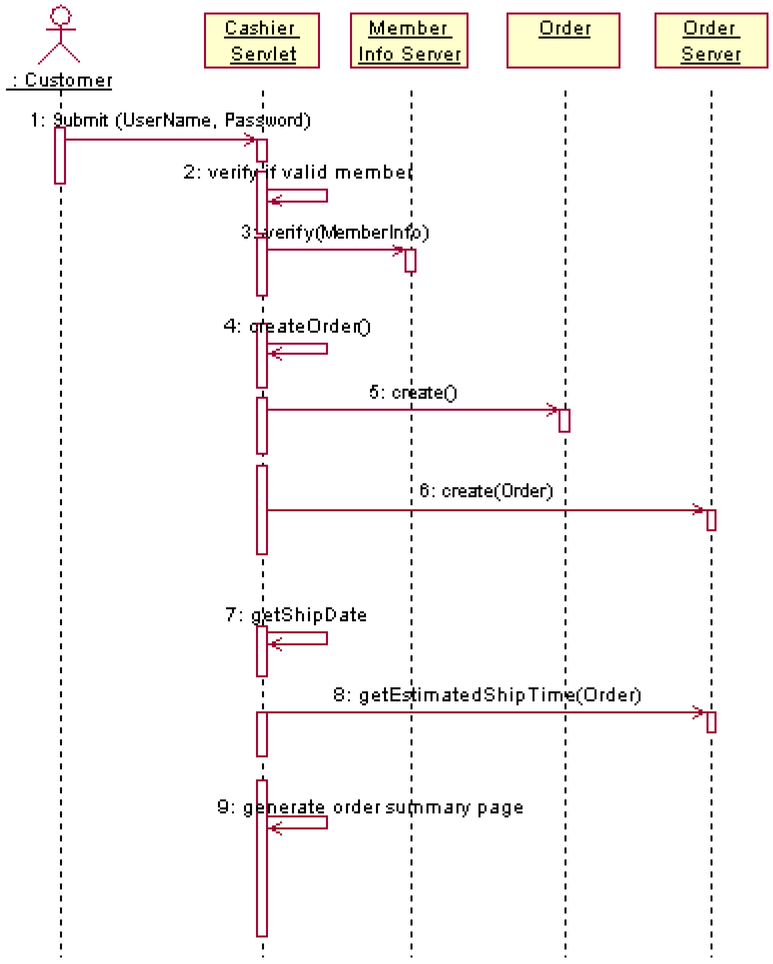
- Names begin with a lower-case letter.
- Names do not contain spaces.
- Within a name, the first letter of each word is capitalized.

4 Create the second message:

- On the diagram toolbox, click the **Object Message** button.
- Click the Cashier Servlet lifeline under getShipDate and drag the message line to the lifeline for Order Server.
- While the message line is still selected, type `getEstimatedShipTime(Order)`. Click in the background of the diagram. (You can resize the text box for the message name so that the entire message fits.)

5 From the Rose menu, click **File > Save** to save the model. If you are prompted to save subunits of the model, click **Yes**.

The Finished Sequence Diagram



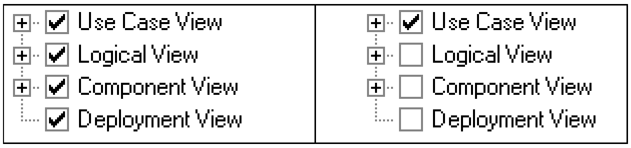
Publishing Part of the Model to the Web

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

Exercise Publish the model to the Web:

1 On the Rose menu, click **Tools > Web Publisher** to display the Rose Web Publisher dialog box.

2 In the Selections list, double-click the check boxes next to **Logical View**, **Component View**, and **Deployment View** to clear them. There should be a check next to **Use Case View**. The illustration shows a before (left side) and after (right side) view of the checkboxes.



3 Next to the HTML Root File Name field, click **...** and browse to `c:\Program Files\Rational\Classics Demo\Web Publish`

In the File name field, type `model` and click **Save**.

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

Rose displays a progress indicator. When it disappears, the Web files are ready.

5 On the Rose Web Publisher dialog box, click **Preview**.

A Web browser appears. It displays a window whose layout is similar to the one in Rose.

6 Explore the model. To see the sequence diagram you worked with in this unit, navigate in the left pane to **Use Case View > cdshop > Arrange Shipment > Arrange Shipment**.

7 When you have finished, minimize or close your Web browser. Then on the Rose Web Publisher dialog box, click **Close**.

After Completing 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 show you how 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.

To see a class diagram, use the Rose browser to navigate to **Logical View > com > rational > cdshop > business > Main**. Double-click **Main**. From the Rose menu, select **View > Fit in Window** to see the entire diagram. Each representation of a class shows you the class attributes and operations. (Double-click a class representation to see details about the class.)

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

Implementing the Code

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

To generate new code or to update existing code, choose an item from the Rose **Tools** menu. For example, to implement code for the enhancement you've been working on, you would choose **Tools > Java > Generate Java**. (Of course, if Java is not installed on your system, this operation does not complete successfully.)

As soon as you start changing code, your model will probably 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 can automate keeping the code and the model consistent. For example, from the Rose menu, you would choose **Tools > Java > Reverse Engineer Java**. This is called *reverse engineering*.

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

Note Rose Enterprise Edition can generate code, update code, and update models. Rose Modeler Edition, included in Rational Suite Analyst Edition, does not have these features.

Modeling Data

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

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

Advantages

Advantages of using Rose Data Modeler are:

- All your business, application, and data models are written in the same industry-standard language, UML, and tool, promoting consistency, traceability, and ease of communication.
- Data Modeler supports both forward and reverse engineering of relational databases, helping to automate the process of keeping code and models consistent.

Summary

For More Information

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

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

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

Cleaning Up

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

What You Learned in this Unit

In this unit, 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.
- You then identify classes and messages to prepare for code generation.
- Rational Rose helps you create visual models for code and data; generate code; and to keep the model synchronized with changed code. From Rose, you can also publish to the Web.
- Rose supports many languages, including Visual Basic, Visual C++, and Java.

What's Next

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

7 Creating a Use Case Report

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

Audience

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

Getting Your Bearings

In this unit, you use Rational SoDA. In addition, Rational Rose and Rational RequisitePro must be installed on your system. Refer to the tool chart you filled out (Table 1, Rational Suite Tools, on page 25) to determine whether these tools are installed on your system.

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

If these tools *are* installed, start SoDA now by clicking the **Start** button and then choosing **Programs > Rational Suite > Rational SoDA for Word**.

If you see a warning about enabling macros, make sure that you choose **Enable Macros**. Otherwise, you will be unable to use SoDA.

Microsoft Word, containing an additional SoDA menu, starts.

What Is SoDA?

SoDA automates the creation of software documentation. It extracts information from software engineering tools such as RequisitePro and Rose. Using Word templates, SoDA formats the information it extracts into a report. You can choose from the many templates provided with SoDA, or you can create your own templates with SoDA's easy-to-use template creation tool.

Adding information to a document is easy because you just use Word to add text. SoDA preserves your text, so it is there the next time you generate the document.

Why Create a Use Case Report?

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

- An *analyst* can show the report to customers and other stakeholders. Together, they can verify that the project is on the right track. These discussions can be held early in the project, so that the analyst can address problems or gaps before, rather than after, the project ships.
- A *software engineer* can use the report's description of expected system behavior to start writing engineering specifications.
- A *test engineer* can use the report to design tests for the use case. From the report, a test engineer can identify the steps to test and determine which conditions to test.
- A *technical writer* can start planning documentation, based on the report's descriptions of how actors interact with the system.
- A *usability engineer* can 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. The template gathers information and formats it into a report. Note that if you have not performed the exercises in Unit 6, "Modeling the Enhancement," you will not be able to create the use case report.

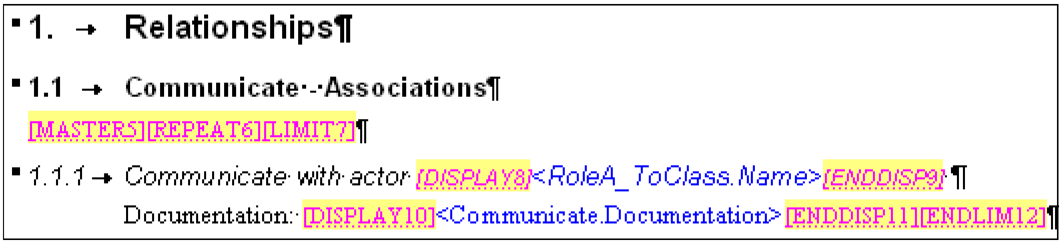
Exercise Create the use case report by starting with the template:

1. In Word, open
   `c:\Program Files\Rational\Classics Demo\ClassicsRepo\Soda\RUP Use Case Report.doc`

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

2. To see the entire template, click **Tools > Options**. On the View tab, under Formatting marks (Word 2000) or Nonprinting Characters (Word 97), select **All**. Click **OK**. (Remember to reset this setting if you prefer not to see all formatting marks when you work with Word.)

Excerpt of SoDA Template



▪1. → Relationships¶

▪1.1 → Communicate·-·Associations¶

[MASTER5][REPEAT6][LIMIT7]¶

▪1.1.1 → Communicate·with·actor·[DISPLAY8]<RoleA_ToClass.Name>[ENDDISP9]¶

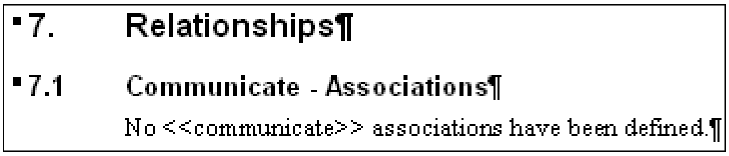
Documentation:·[DISPLAY10]<Communicate.Documentation>[ENDDISP11][ENDLIM12]¶

3. From the Word menu, click **SoDA > Generate Report**.

   Word minimizes and SoDA displays a progress indicator. After a short pause, Word restores itself and displays the report.

4. Browse through the report. You see the requirements you worked on in Unit 4, “Creating Requirements” and the visual model of the system you worked on in Unit 6, “Modeling the Enhancement.”

Excerpt of SoDA Report



▪7. Relationships¶

▪7.1 Communicate - Associations¶

No <<communicate>> associations have been defined.¶

Working with SoDA Templates

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

Exercise To view the template tool for the template you worked with earlier:

1. In Word, re-display the file `RUP Use Case Report.doc`.
2. From the Word menu, click **SoDA > Template View**.
3. Word displays the SoDA Template View tool. You may want to study how each line translates into the template's macros.

Summary

For More Information

To get started with SoDA, from the Word menu, click **Help > Help on SoDA**. In the Help Topics window, click **SoDA Tutorial**. Follow the instructions displayed on your screen.

Cleaning Up

Close the SoDA documents. If you'd like, quit Word also.

What You Learned in this Unit

In this unit, you learned:

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

What's Next

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

8 Reliability Testing

The ClassicsCD Administrator enhancement is implemented and test planning has occurred. You are now ready to test the enhancement. In this unit, we discuss testing for reliability. In the next unit, you perform functional testing.

Audience

This unit applies to testers and developers.

Reliability Testing Tools

This chapter describes the following automated testing tools:

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

What Is TestFactory?

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

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

Overview of Process

You use TestFactory as follows:

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

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

Instrumenting the Application

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

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

Mapping the Application

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

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

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

Running a Pilot

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

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

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

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

Test Suites: Putting It All Together

Use a Test Suite to organize scripts and to run them automatically as a group. In a Test Suite, you can include:

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

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

Using TestFactory with Rational Robot

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

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

Additional Testing Tools

Rational Suite provides additional testing tools that both developers and testers use throughout the development lifecycle. These tools run on Windows NT and Windows 2000; they do not run on Windows 95 or Windows 98.

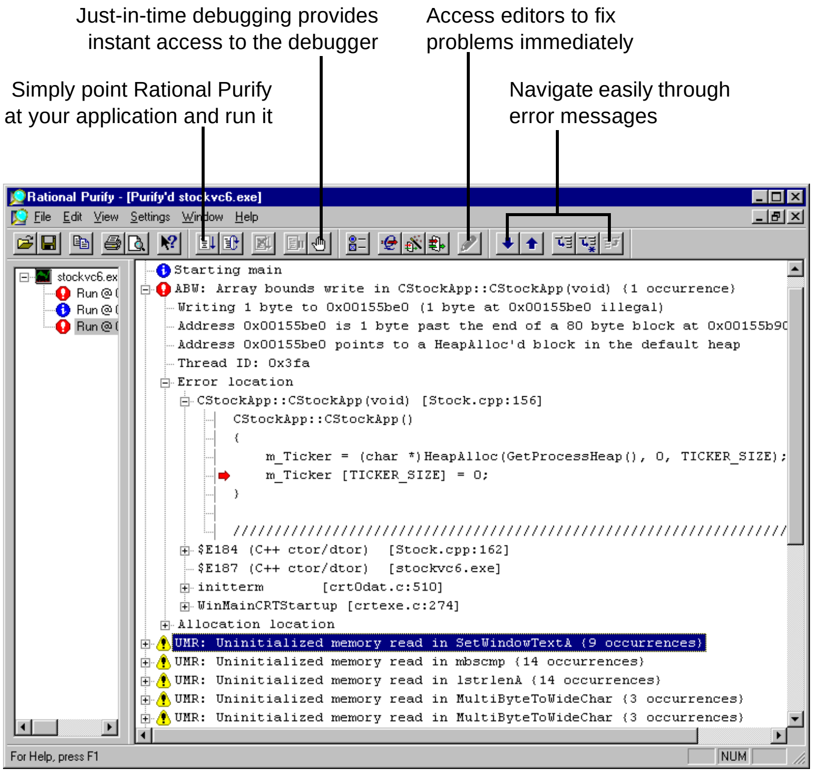
Rational Purify

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

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

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

Purify Features


Just-in-time debugging provides instant access to the debugger
Access editors to fix problems immediately
Simply point Rational Purify at your application and run it
Navigate easily through error messages
Rational Purify - [Purify'd stockvc6.exe]
File Edit View Settings Window Help
stockvc6.ex
Run @ (
Run @ (
Run @ (
For Help, press F1
NUM

```
Starting main
ABW: Array bounds write in CStockApp::CStockApp(void) {1 occurrence}
  Writing 1 byte to 0x00155be0 (1 byte at 0x00155be0 illegal)
  Address 0x00155be0 is 1 byte past the end of a 80 byte block at 0x00155b90
  Address 0x00155be0 points to a HeapAlloc'd block in the default heap
  Thread ID: 0x3fa
  Error location
    CStockApp::CStockApp(void) [Stock.cpp:156]
        CStockApp::CStockApp()
        {
            m_Ticker = (char *)HeapAlloc(GetProcessHeap(), 0, TICKER_SIZE);
            m_Ticker [TICKER_SIZE] = 0;
        }

        ///////////////////////////////////////////////////////////////////
    $E184 (C++ ctor/dtor)  [Stock.cpp:162]
    $E187 (C++ ctor/dtor)  [stockvc6.exe]
    initterm       [crt0dat.c:510]
    WinMainCRTStartup [crtexe.c:274]
  Allocation location
UMR: Uninitialized memory read in SetWindowTextA {9 occurrences}
UMR: Uninitialized memory read in mbscmp {14 occurrences}
UMR: Uninitialized memory read in lstrlenA {14 occurrences}
UMR: Uninitialized memory read in MultiByteToWideChar {3 occurrences}
UMR: Uninitialized memory read in MultiByteToWideChar {3 occurrences}
```

Easy to understand error symbols:

Error
Warning
Information
Error Location

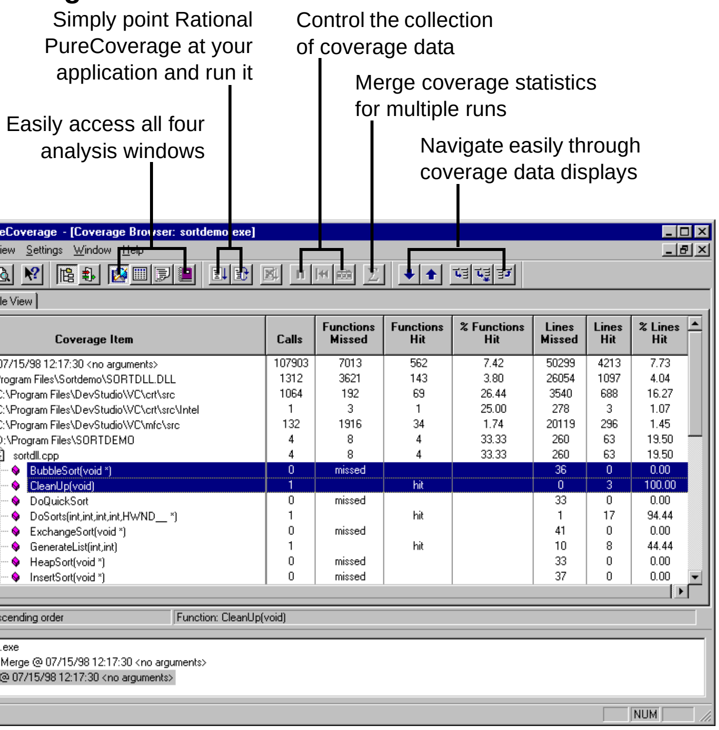
Rational PureCoverage

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

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

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

PureCoverage Features


Simply point Rational PureCoverage at your application and run it
Control the collection of coverage data
Merge coverage statistics for multiple runs
Easily access all four analysis windows
Navigate easily through coverage data displays
Rational PureCoverage - [Coverage Browser: sortdemo.exe]
File Edit View Settings Window Help
Module View
File View

Coverage Item	Calls	Functions Missed	Functions Hit	% Functions Hit	Lines Missed	Lines Hit	% Lines Hit
Run @ 07/15/98 12:17:30 <no arguments>	107903	7013	562	7.42	50299	4213	7.73
D:\Program Files\Sortdemo\SORTDLL.DLL	1312	3621	143	3.80	26054	1097	4.04
C:\Program Files\DevStudio\VC\crt\src	1064	192	69	26.44	3540	688	16.27
C:\Program Files\DevStudio\VC\crt\src\Intel	1	3	1	25.00	278	3	1.07
C:\Program Files\DevStudio\VC\mfc\src	132	1916	34	1.74	20119	296	1.45
D:\Program Files\SORTDEMO	4	8	4	33.33	260	63	19.50
sortdll.cpp	4	8	4	33.33	260	63	19.50
BubbleSort(void *)	0	missed			36	0	0.00
CleanUp(void)	1		hit		0	3	100.00
DoQuickSort	0	missed			33	0	0.00
DoSorts(int,int,int,int,HWND__ *)	1		hit		1	17	94.44
ExchangeSort(void *)	0	missed			41	0	0.00
GenerateList(int,int)	1		hit		10	8	44.44
HeapSort(void *)	0	missed			33	0	0.00
InsertSort(void *)	0	missed			37	0	0.00

Coverage Item: Ascending order
Function: CleanUp(void)

sortdemo.exe
Auto Merge @ 07/15/98 12:17:30 <no arguments>
Run @ 07/15/98 12:17:30 <no arguments>

Ready NUM

The Coverage Browser provides an easy to use outline view for fast analysis of coverage data

Rational Quantify

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

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

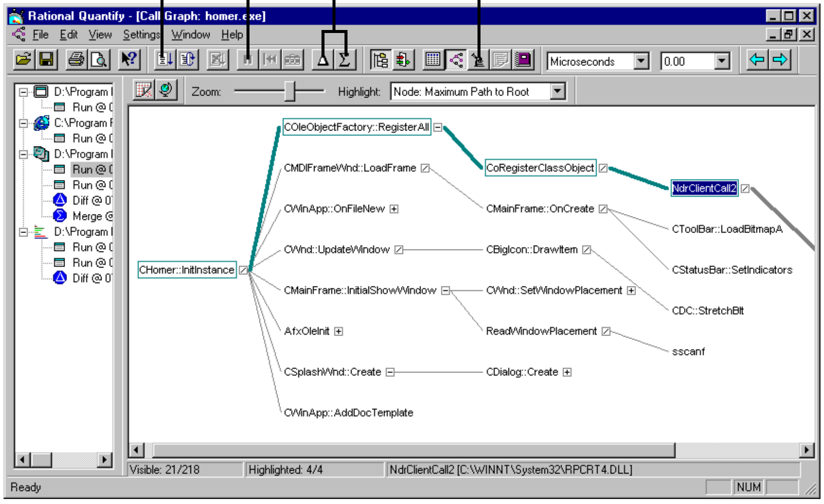
Quantify Features

Simply point Rational Quantify at your application and run it

Control the collection of profiling information

Compare execution time between two runs or merge execution time for multiple runs

Drill down to detailed performance data



Summary

For More Information

For more information about:

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

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

What You Learned in this Unit

In this unit, you learned:

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

What's Next

In the next unit, you continue testing the ClassicsCD Administrator 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.

9 Functional Testing

At this point in the development process, developers have implemented the ClassicsCD.com enhancement (this work was done outside of the tutorial). Test engineers have run initial reliability tests. In this unit, you perform functional tests on the enhancement.

Audience

This unit applies to test engineers.

Getting Your Bearings

In this unit, you start by using Rational TestManager, which is installed with Rational Robot. Refer to the tool chart you filled out (Table 1, Rational Suite Tools, on page 25) to determine whether Rational Robot is installed on your system.

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

If Rational Robot *is* installed, start TestManager now (later in this unit, you also work with Robot):

1 Click the **Start** button and choose **Programs > Rational Suite > Rational TestManager**.

2 If the Rational Repository Login dialog box appears, use these values:

- In the User ID and Password boxes, type `pat`.
- Make sure the Repository Path box displays `c:\Program Files\Rational\Classics Demo\ClassicsRepo`
- Ignore the value in the Project field.
- Click **OK**.

The Rational TestManager appears.

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 its GUI to validate that the system responds appropriately to user input. Testing can focus on both the operation and the appearance of GUI objects.

Start with Requirements

To determine how a system is intended to behave, you start with requirements. In Unit 5, "Test Planning," you worked with test requirements that were based on use case requirements. In the first part of this unit, you use Rational TestManager to work with those same test requirements.

Creating a Script

During testing, you focus on planning, recording, and running scripts. A script has the following components:

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

Determining Which Script to Work On

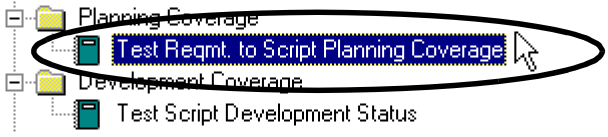
The ClassicsCD.com system has several test requirements, all related to use case requirements. In Unit 5, you traced the Arrange Shipment test requirement to its use case requirement. Now use TestManager to determine which test requirements still need scripts.

Exercise Use TestManager to report on which test requirements do and do not have associated scripts.

1 From the TestManager menu, click **Reports > Run**.

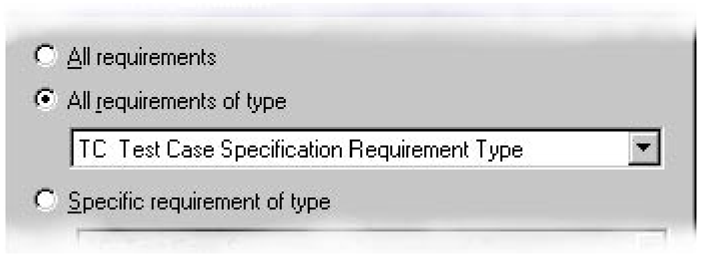
2 On the Run Report dialog box, select **Test Reqmt. to Script Planning Coverage** and click **OK**.

Selecting a Report to Run


Planning Coverage
Test Reqmt. to Script Planning Coverage
Development Coverage
Test Script Development Status

3 On the Select Requirement dialog box, click **All requirements of type** and then select **TC Test Case Specification Requirement Type** from the list. (Scroll up to see it.) Click **OK**.

Selecting a Requirement Type


All requirements
All requirements of type
TC Test Case Specification Requirement Type
Specific requirement of type

The report appears, showing the status of test planning. The left column shows the test requirements. As you can see, some test cases have scripts planned; others do not.

4 We will focus on test case *TC12 Arrange Shipment Functionality*. Scroll to that test case now. Notice that there are no scripts planned for the test case.

Planning the Script

Now that you know which test case needs a script, go to the requirements view in TestManager. In the following exercise, you use the integration between RequisitePro and Rational Test to plan the script.

Exercise Use TestManager to look at test requirements and to plan a script.

1 From the TestManager menu, click **File > Open Requirements**.

2 From the TestManager menu, click **View > Select Requirement Type > TC Test Case Specification Requirement Type** to display test case specifications.

The Requirements Hierarchy, listing test case requirements (TC), appears. The test requirements you see here are the same requirements you saw in RequisitePro. In fact, TestManager loads the requirements directly from the RequisitePro database.

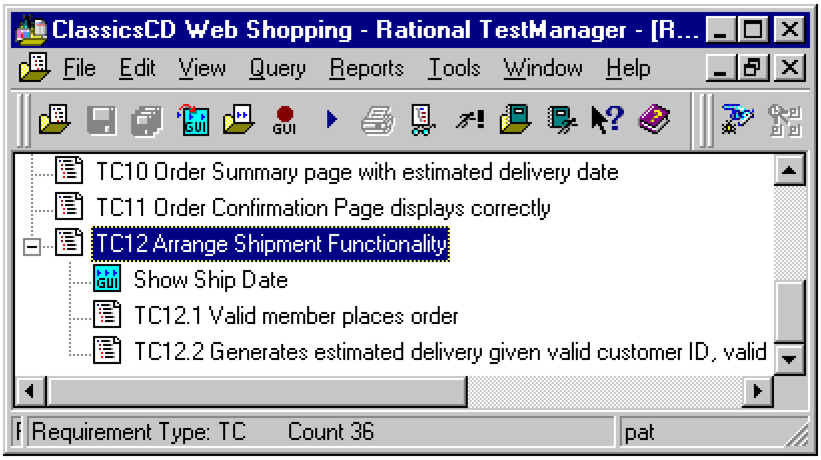
3 Right-click requirement *TC12 Arrange Shipment Functionality*. From the shortcut menu, click **Plan > GUI Script**.

4 On the Plan Script dialog box:

- In the **Name** box, type `Show Ship Date`.
- In the **Description** box, type: `Shows date that order will ship`.
- Click the Related Assets tab and verify that there is a link to TC12.
- Click **OK**.

In the Requirements Hierarchy, a script icon appears below the test requirement. You have now planned a script. It has properties associated with it (for example, the name and description), but it does not yet have code.

Requirements Hierarchy after Planning a Script


ClassicsCD Web Shopping - Rational TestManager - [R...
File Edit View Query Reports Tools Window Help

TC10 Order Summary page with estimated delivery date
TC11 Order Confirmation Page displays correctly
TC12 Arrange Shipment Functionality
Show Ship Date
TC12.1 Valid member places order
TC12.2 Generates estimated delivery given valid customer ID, valid
Requirement Type: TC   Count 36   pat

Recording the Script

After planning a script, you record it. When you record a script, Robot translates the activities you perform into scripting language commands. (Robot uses SQABasic for its scripting language. SQABasic resembles Microsoft Visual Basic and contains additional commands tailored for automated testing). After you record a script, you can reuse it, for example, in regression tests.

Scripts and Modularity

The script you will record tests the ClassicsCD.com enhancement, Arrange Shipment. You could record a script that starts the application, enters a sale, and then verifies the reorder. However, you can't reuse such a script without modifying it.

Instead, you can create a set of scripts that all start with the same steps and conclude by testing different parts of the application. Fortunately, Robot lets you create short modular scripts. You can combine a series of scripts into one shell script (a *Test Suite*). With this technique, you can reuse the same script in different tests.

Starting and Preparing Robot

Exercise Prepare to use Robot.

1. From the TestManager menu, click **Tools > Rational Test > Rational Robot** to start Robot.
2. From the Robot menu, click **Tools > GUI Playback Options**.
3. On the GUI Playback Options dialog box, on the Log tab, select the following:
   - **Output playback results to log**
   - **View log after playback**
   - **Specify log information at playback**
4. Click **OK**.

Is Internet Explorer Installed?

You can record the new script only if Internet Explorer is installed on your system. (It does not need to be your default browser.) If Internet Explorer is not installed on your system, continue reading this chapter and resume performing the exercises starting with *Adding the Show Ship Date Script to a Shell Script* on page 100.

Getting to a Starting Point

In this section, you reuse a script that has already been recorded to get the application to an appropriate starting place for recording the Show Ship Date script.

Exercise Replay a script that completes a sale:

1. From the Robot menu, choose **File > Playback**.
2. On the Playback dialog box, select **Tutorial – Shell Buy Beethoven And Bach** from the list (type the first few letters to find it more easily). Click **OK**.

   Robot displays a Specify Log Information dialog box, which allows you to specify how to store test results.
3. From the Build list, select **Build 1**. (Use the defaults for the other values.) Click **OK**.
4. If Robot displays a confirmation dialog box asking if you want to overwrite the log, click **Yes**.

   Do not interact with the ClassicsCD.com application while Robot plays back the Cash Sale script! If you see a 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, the Rational LogViewer appears, showing the results of the test. All or almost all the comparisons (the verification points) pass.

   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 appeared on your screen. Robot noticed the window but the window did not interfere with the test results.

If Robot returned the warning, double-click on the warning line to see a screenshot of the unexpected window. When you have finished, close the Image Comparator window.

5 Close the LogViewer window.

Starting to Record the Show Ship Date Script

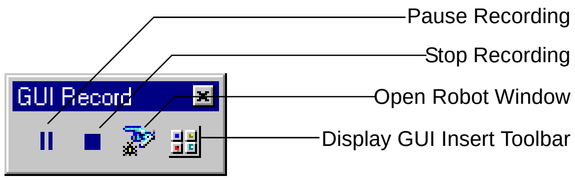
You are now ready to record the Show Ship Date script.

Exercise Start recording the script from the point where the Buy Beethoven and Bach script finished.

1 In TestManager, on the Requirements Hierarchy window, right-click the **Show Ship Date** script. From the shortcut menu, select **Record**.

The GUI Record toolbar appears.

The GUI Record Toolbar



2 Click the title bar of Internet Explorer to make it the active window.

3 Scroll to the bottom of the Internet Explorer window. Notice that the estimated ship date is now displayed.

Creating a Verification Point

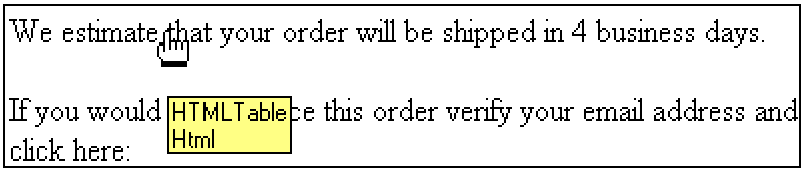
In the following steps, you create a *verification point*, which establishes a baseline value for a specific part of the application. When you play back the script, Robot compares the value it finds to the baseline value you establish.

Exercise Create the verification point.

1. On the GUI Record toolbar, click the **Display GUI Insert Toolbar** button.
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`, then click **OK**.
4. From the Select Object dialog box, drag the hand pointer to the line on the Internet Explorer window that starts "We estimate that your order..."

   The Select Object dialog box briefly disappears and then reappears.

   Dragging the Hand Pointer to the Internet Explorer

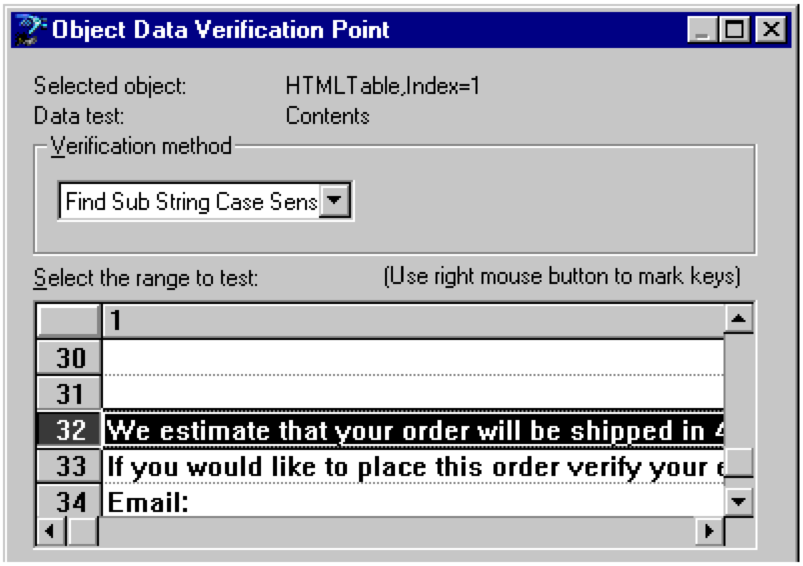
   We estimate that your order will be shipped in 4 business days.

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

5. Click **OK** to close the Select Object dialog box.
6. The Object Data Tests dialog box appears. Click **OK**.

   The Object Data Verification Point dialog box appears, showing the text that was captured.
7. On the Object Data Verification Point dialog box:
   - From the Verification method list, select **Find Sub String Case Sensitive**.
   - Under Select the range to test, scroll to line 32 ("We estimate that ...") and click it so that it is the only line selected.
   - Click **OK** to close the dialog box.

Working with the Object Data Verification Point Dialog Box



Selected object: HTMLTable,Index=1
Data test: Contents
Verification method
Find Sub String Case Sens
Select the range to test: (Use right mouse button to mark keys)

	1
30	
31	
32	We estimate that your order will be shipped in 4
33	If you would like to place this order verify your e
34	Email:

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

8 On the Internet Explorer window, click **Place Order**.

Finishing the Recording Session

You can now finish the recording session.

Exercise Perform the final steps in the script:

1 On the GUI Record toolbar, click **Stop Recording** ■.

Robot displays the Show Ship Date script.

2 Optional: read the script commands and notice how the commands correlate to the actions you took when you recorded the script.

Adding the Show Ship Date Script to a Shell Script

Recall that before you recorded the Show Ship Date script, you first set up the application by running the Tutorial – Shell Buy Beethoven And Bach script. During actual testing, you would have to replay Show Ship Date repeatedly. However, you would want to avoid going through manual steps to set up the application each time.

Instead, you can add the Show Ship Date script to an existing shell script. The shell script calls other scripts that set up and shut down the application.

Exercise Add the Show Ship Date script to the shell script.

1 From the Robot menu, click **File > Open > Script**.

2 In the Open Script dialog box, select **Tutorial – Shell Shop for CDs** and click **OK**.

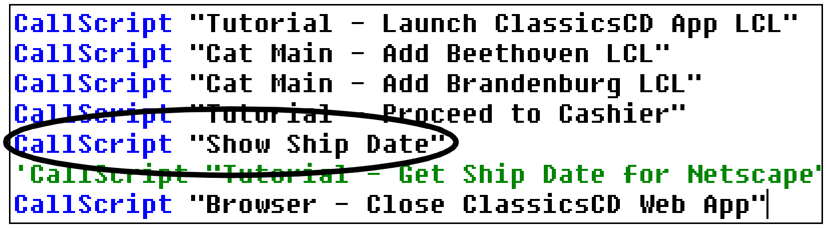
Robot displays the Tutorial – Shell Shop for CDs script. This script calls other scripts; it does not directly interact with the application.

3 Take one of the following steps, depending on whether Internet Explorer is installed on your system:

- Internet Explorer *is* installed. In the space after *CallScript "Tutorial - Proceed to Cashier"*, add the line:

```
CallScript "Show Ship Date"
```

Adding a Line to the Shell Script



```
CallScript "Tutorial - Launch ClassicsCD App LCL"
CallScript "Cat Main - Add Beethoven LCL"
CallScript "Cat Main - Add Brandenburg LCL"
CallScript "Tutorial - Proceed to Cashier"
CallScript "Show Ship Date"
'CallScript "Tutorial - Get Ship Date for Netscape"
CallScript "Browser - Close ClassicsCD Web App"
```

- Internet Explorer *is not* installed. A quote character precedes commented code. Remove the quote character from the beginning of the green line, *CallScript "Tutorial – Show Ship Date for Netscape"*. (We have recorded a script for you that performs the work in the Show Ship Date script.)

4 From the Robot menu, click **File > Save**.

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

Playing Back the Script On a New Build

While you were working, the developers delivered a new build of ClassicsCD.com.

Exercise Run the Test Suite on the new build.

1. From the Robot menu, click **File > Playback**.
2. Verify that the Playback dialog box selects the *Tutorial – Shell Shop for CDs* script. Click **OK**.
3. In the Specify Log Information dialog box, in the Build box, select **Build 2**. Click **OK**.
4. If Robot displays a confirmation dialog box asking if you want to overwrite the log, click **Yes**.

   Resist the temptation to interact with the application as Robot plays back the series of scripts. If you see a message box that starts “Do you want Windows to remember...”, wait. Robot will eventually continue.

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

Analyzing the Results

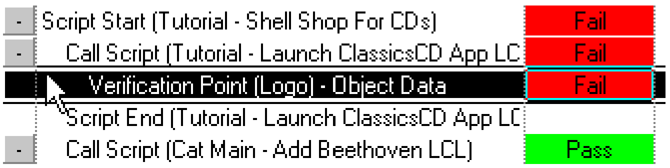
The LogViewer shows which verification points passed and which failed. The script you recorded, Show Ship Date, passes despite the UI changes in Build 2. However, the Cash Sale script has failures. Recall that when you played back Cash Sale on Build 1, it passed.

Handling an Intentional Change

Exercise Inspect the failure on the third line and decide how to handle it.

1 In the LogViewer, double-click the failure, **Verification Point (Logo)**.

Selecting the First Failure



-	Script Start (Tutorial - Shell Shop For CDs)	Fail
-	Call Script (Tutorial - Launch ClassicsCD App LC	Fail
	Verification Point (Logo) - Object Data	Fail
	Script End (Tutorial - Launch ClassicsCD App LC	
-	Call Script (Cat Main - Add Beethoven LCL)	Pass

The Object Properties Comparator for the logo appears, showing that the logo changed from `classicsCDsplash.jpg` to `classicsCDlogo.jpg`. It turns out that this was a planned change and results from the UI modifications mentioned earlier.

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

2 From the menu, click **File > Replace Baseline with Actual**. On the confirmation dialog box, click **Yes** to direct Robot to perform the replacement.

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

3 Close the Object Properties Comparator.

Handling a Real Error

Exercise Inspect the second failure and determine how to handle it.

1 In the LogViewer, under *Call Script (Tutorial – Proceed to Cashier)*, double-click the next failure, **Verification Point (Cart Contents)**.

The Object Properties 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.

2 Close the Object Properties Comparator.

Reporting the Error

To report the error, use ClearQuest, which is integrated with Rational LogViewer.

Exercise Report the error.

1 From the LogViewer menu, click **Defect > Generate**.

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

2 If ClearQuest displays the Schema Repository – User GUI dialog box, choose **2000.02.10** and click **OK**. (This dialog box allows you to work with a database schema from the current ClearQuest release or from a previous release.)

3 If ClearQuest displays a Login dialog box, in both the User Name and Password boxes, type `pat`. If the Database list is enabled, choose **CLSIC: Rational Demo**. Click **OK**.

A Submit Defect dialog box opens. Red items are required: you cannot submit the defect until all required fields contain valid values. A tab with a red square indicates that the tab contains required items.

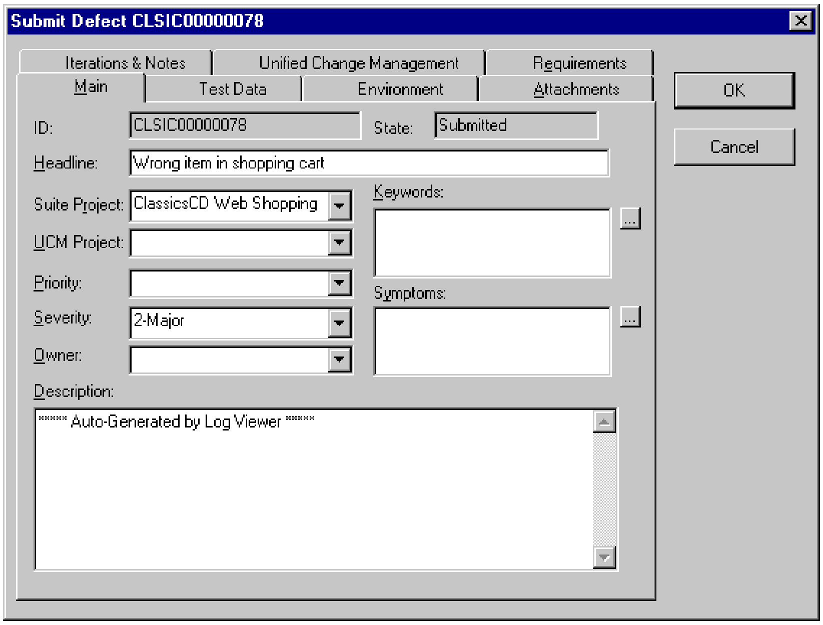
4 In the Headline box, type:

```
Wrong item in shopping cart
```

5 From the Severity list, select **2-Major**.

6 On the other tabs, for example, **Test Data**, Rational Test has already filled in fields related to the test script.

The Defect You Will Submit



ID: CLSIC00000078
State: Submitted
Headline: Wrong item in shopping cart
Suite Project: ClassicsCD Web Shopping
UCM Project:
Priority:
Severity: 2-Major
Owner:
Keywords:
Symptoms:
Description:
***** Auto-Generated by Log Viewer *****

7 Click **OK** to close the Submit Defect dialog box.

In the LogViewer, the defect ID appears next to the Cart Contents failure.

You have finished testing this iteration of ClassicsCD.com.

Summary

For More Information

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

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

To learn more about performance testing, see *Getting Started with Rational Suite PerformanceStudio*.

Cleaning Up

Quit any open Rational Test tools.

What You Learned in this Unit

In this unit, you learned:

- Functional testing helps you determine whether a system behaves as intended.
- Rational Test helps you plan, develop, run, and analyze functional tests.
- You develop test scripts by interacting with the application using Robot and including verification points in your scripts.
- You can develop modular scripts, then use shell scripts to call those scripts. You reuse scripts each time developers deliver a new software build.
- Robot makes it easy to address problems and updates that are discovered during testing.

What's Next

You are nearly finished with the tutorial! In the next unit, you plan the next iteration of ClassicsCD.com.

10 Planning the Next Iteration

The ClassicsCD.com enhancement is now complete. You have finished work on this iteration. This unit describes the next steps.

Audience

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

Getting Your Bearings

In this unit, you use Rational ClearQuest. Refer to the tool chart you filled out (Table 1, Rational Suite Tools, on page 25) to determine whether ClearQuest is installed on your system.

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

If ClearQuest *is* installed, start it now:

1. Click the **Start** button and choose **Programs > Rational Suite > Rational ClearQuest**.
2. If ClearQuest displays the Schema Repository – User GUI dialog box, choose **2000.02.10** and click **OK**. (This dialog box allows you to work with a database schema from the current ClearQuest release or from a previous release.)
3. In the ClearQuest Login dialog box:
   - In both the **User Name** and **Password** boxes, type `pat`.
   - From the **Database** list, choose **CLSIC: Rational Demo**.
   - 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 Unit 9, "Functional Testing," you used ClearQuest to report a defect in the software. In this unit, you use ClearQuest charts and queries to assess the state of your project.

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

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

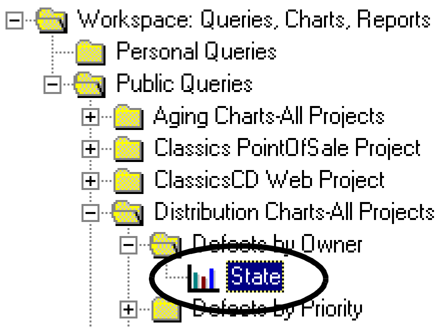
Showing the Workload

At the end of an iteration, you probably want to review each project member's workload so that you can most effectively allocate work for the next iteration.

Exercise Display a chart showing workload.

1 In the left pane of ClearQuest, navigate to: **Public Queries > Distribution Charts – All Projects > Defects by Owner > State**. Double-click this entry.

Choosing the Chart, Defects by Owner and State


Workspace: Queries, Charts, Reports
Personal Queries
Public Queries
Aging Charts-All Projects
Classics PointOfSale Project
ClassicsCD Web Project
Distribution Charts-All Projects
Defects by Owner
State
Defects by Priority

ClearQuest displays the workload chart. Maximize the chart to see the details more clearly. Notice that Devon has just one defect assigned. The turquoise bar on the left represents unassigned defects.

2 Click a bar to see the owner, state, and number of defects.

3 Double-click the turquoise bar (the leftmost bar) to list the defects summarized in that bar.

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

4 Click **OK**.

ClearQuest lists the defects and displays details about the selected defect. You can edit the detailed defect.

ClearQuest can help you plan a new iteration. Now that you have listed the unassigned defects, you can assign them (on the **Main** tab) and link them to requirements (on the **Requirements** tab).

So from general information, you can easily navigate to details.

Assigning an Enhancement Request

Recall that you started the tutorial by looking at an enhancement request that you then implemented. Now find another enhancement request to implement in the next iteration.

Exercise Start by examining the enhancement requests:

1 In the left pane of ClearQuest, navigate to **Public Queries > Unassigned ERs**. Double-click this entry.

2 In the list at the top of the right pane, click *CLSIC00000077, Email current member base with monthly specials* to display the enhancement request's details.

3 Optional: Click the request's tabs to learn more about the request and its history.

4 Click **Actions**, then select **Assign**.

Notice that some fields turn white, indicating that you can change them. Also, on tabs where you must fill in a field, a red square appears.

5 Click the Analysis tab.

The Owner field is red, indicating that a value is mandatory. Recall that Devon does not have a lot of assigned work right now.

6 From the Owner list, select **devon**.

7 Click **Apply**.

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 RequisitePro or ClearQuest to identify defects to implement in the next iteration.

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

What Will Happen in the Next Iteration?

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

- An analyst discusses planned enhancements with stakeholders. Using RequisitePro and Rose, the analyst creates one or more use cases and supplies step-by-step details, including basic flow and alternative flows.
- A test engineer uses Rational TestManager to plan the tests for this iteration. The engineer writes a test plan document, develops test requirements, and designs the tests.
- A developer uses Rose and visual modeling techniques to describe how planned enhancements fit within the architecture of the system.
- Anyone on the team can use SoDA to create a use case report. This report is useful during discussions with stakeholders and design sessions.
- Developers use Rose to initiate implementation of the enhancement.
- Test engineers use TestFactory, Purify, Quantify, and PureCoverage to verify the iteration's reliability.

- Test engineers use Rational TestManager and Robot to verify that the enhancements meet requirements that defects are fixed correctly, and that no regression failures have occurred.
- Meanwhile, project and group managers use ClearQuest, SoDA, and RequisitePro to assess that state of the project. Later, they use these tools to plan subsequent iterations.

Configuration Management

This tutorial does not discuss configuration management, a crucial component of any software development environment. Configuration management is the process of controlling changes to, and maintaining the integrity of, your project's code and other artifacts.

Rational ClearCase, the premier configuration management solution, is the perfect complement to Rational Suite because it is integrated with Rational Suite tools and with Microsoft Visual Studio. ClearCase provides a unified approach to managing change throughout the software development lifecycle. As part of a total change management solution that spans configuration management, change request management, and workflow automation, ClearCase can help you streamline and simplify the process of change.

Summary

For More Information

For more information about ClearCase, please see the following Web page:

`http://www.rational.com`

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

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

Cleaning Up

When you are ready, quit ClearQuest.

What You Learned in this Unit

In this unit and tutorial, you learned:

- ClearQuest is a powerful tool that helps you manage and monitor change requests on your project.
- ClearCase is an important tool that assists with configuration management. It is the perfect complement to Rational Suite.

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.
- Rational Suite simplifies adoption by providing a comprehensive set of integrated tools that have simple installation, licensing, and user support.
- Rational Suite supports the entire development life cycle and the primary participants on a development team – analysts, developers, testers, and managers.

What's Next

Congratulations! You have finished the Rational Suite tutorial. Your next job is to find out more about the tools you will use on your next project and to get to work! We hope that by using Rational Suite, you will ensure your own continued success.

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, and documents.

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, and accurate process.

class

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

component

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

component-based architecture

A design technique in which a software system is decomposed into individual components.

configuration management

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

developer

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

Extended Help

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

forward engineering

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

iterative development

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

method

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

metrics

The measurements of project activity.

object

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

Rational Administrator

Tool that manages Rational repositories. For more information, see *Using the Rational Administrator*.

Rational ClearCase

Provides comprehensive configuration management, including version control, workspace management, build management, and process control.

Rational ClearQuest

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

Rational PureCoverage

Automatically pinpoints areas of code that have not been tested.

Rational Purify

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

Rational Quantify

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

Rational RequisitePro

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

Rational Robot

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

Rational Rose

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

Rational SoDA for Word

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

Rational Suite

An easy-to-adopt-and-support solution that optimizes the productivity of analysts, developers, and testers – and unifies them, creating highly effective software development teams.

Rational Suite AnalystStudio

Edition of Rational Suite optimized for analysts. Contains the team unifying tools – Rational Unified Process, RequisitePro, ClearQuest, and SoDA – and Rational Rose (Modeler Edition).

Rational Suite DevelopmentStudio

Edition of Rational Suite optimized for system developers and designers. Contains the team-unifying tools – Rational Unified Process, RequisitePro, ClearQuest, and SoDA – 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 tools – Rational Unified Process, RequisitePro, ClearQuest, and SoDA – plus Rational Rose RealTime, Rational Purify, Rational Quantify, and Rational PureCoverage.

Rational Suite Enterprise

Edition of Rational Suite containing all Rational Suite tools except Rational LoadTest.

Rational Suite PerformanceStudio

Edition of Rational Suite optimized for test engineers who develop and run performance tests. Contains the team-unifying tools – Rational Unified Process, RequisitePro, ClearQuest, and SoDA – plus Rational Test tools, Rational Rose (Enterprise Edition), and Rational LoadTest.

Rational Suite TestStudio

Edition of Rational Suite optimized for test engineers. Contains the team unifying tools – Rational Unified Process, RequisitePro, ClearQuest, and SoDA – and Rational Test tools.

Rational Synchronizer

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

Rational TestFactory

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

Rational Unified Process

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

real-time application

An application or system with stringent requirements for latency, throughput, reliability, and availability.

requirement

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

requirements management

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

reverse engineering

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

risk

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

risk management

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

round-trip engineering

The ability to do both forward and reverse engineering as often as needed.

test engineer

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

Tool Mentor

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

traceability

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

Unified Modeling Language (UML)

The industry-standard language for specifying, visualizing, constructing, and documenting software systems. It simplifies software design, and communication about the design.

use case

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

version control

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

vision document

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

visual model

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

worker

The role played by an individual team member.

workflow

The sequence of activities performed in a business that produces a valuable result to an individual actor in the business.

Index