

# Improved application development: Part 5, Test and verify with Rational

Skill Level: Introductory

[Martin C. Brown](#)

Author

Studio B

20 Sep 2004

This tutorial, which is the final of a five-part series, focuses on the role of software testing during application development. Testing is a vital part of any development process, and to perform adequate testing you need not only to identify faults but also to trace and track these faults, fixes, and the components they affect during each iteration of the development process. Only by tracking this information can you fix the faults and monitor the progress of your repairs. In this tutorial, we look at the integration between the Rational software testing products and other tools used in the development process, such as RequisitePro, ClearCase, and ClearQuest.

## Section 1. Before you start

### About this tutorial

When testing an application, either in its original form, during defect tracking or during the testing and verification phases of development, it is useful to be able to track those changes and modifications directly from the test environment into IBM(R) Rational(R) ClearQuest and to ultimately propagate this information through IBM WebSphere(R) Studio, IBM Rational ClearCase and the IBM Rational XDE environment.

This final tutorial in the series covers the integration of the testing applications in the Rational suite. You'll learn how you can use the information and reports that these applications generate during testing of the Auction application (the same application

used throughout the series) to track the progress of your application development and bug-fixing process. By tracking this information through ClearQuest and ultimately back to RequisitePro, you can more effectively monitor your progress and associate bugs with specific components and requirements. By using the techniques presented in this tutorial, you'll learn about the roles and capabilities of the Rational testing applications (including PurifyPlus and TestManager) and how to feed the results from these applications into the ClearQuest system.

In the previous tutorials in this series, you learned how to define requirements and develop the application. Testing is the final stage, and this tutorial provides the information you need to drive the next iteration: fixing the bugs and completing the development according to the original requirements.

In this tutorial you'll learn how to test an application using a suitable Rational testing tool (such as IBM Rational TestManager), while feeding test results and change requests back into the ClearCase and ClearQuest systems and how to integrate the testing process into Rational XDE.

Key parts of the tutorial include:

- Building the test environment
- Linking tests to application components
- Tracking tests and results to requirements and change requests
- Tracking code change submissions to fixes and tests

## Should I take this tutorial?

This tutorial series is primarily targeted for project managers, program managers, and developers. Others in the software development business may also find it useful. In each of the tutorials in this series, you will learn about different Rational tools and the part they play in the software development process. Different team members might be interested in some tutorials more than others, but it is worthwhile to take the complete series from beginning to end to see how all the products work together to form a cohesive process.

This final tutorial focuses on integrating different tools in the Rational software development environment, looking at the role of software testing using the Rational PurifyPlus Suite and the Rational TestManager system.

Ideally, you will already be familiar with the application development process, and be involved in at least one part of the process. Familiarization with Rational tools is helpful, but is not required.

## Prerequisites

In the first part of the series you concentrated on using two products in the Rational Suite: Rational RequisitePro and Rational XDE Developer (for Java™). Now you will use Rational PurifyPlus and Rational TestManager or Rational Team Unifying Platform. Links to download copies of these products are provided below.

- [Rational RequisitePro](#)
- [Rational PurifyPlus](#)
- [Rational TestManager](#) or [Rational Team Unifying Platform](#)

Note that you must have the same releases of the tools for the integration to work. You'll be using v2003.06.12 of all the various tools in this series.

A demonstration of developing the Online Auction system can be downloaded from the IBM® Web site using the following links:

- [Online Auction Demo Part 1](#)
- [Online Auction Demo Part 2](#)
- [Online Auction Demo Part 3](#)

You can also find the code itself supplied as the example with the WebSphere Studio Application Developer, a tool you'll actually be using later in the series. Check the Auction panel in the first section for more details of the auction application and where to obtain the code.

---

## Section 2. Test types and tools

### Test role in application development

All application software should be tested. No matter how good your developers or your software development processes, you are going to end up with some faults in your application. Some of these involve errors in logic, others are related to programming errors by developers. Some, of course, are due to the design and requirements of the original application, which won't always match the expectations of the original stakeholders in the project. Testing can also be used to verify the operation of an application against the original requirements.

Testing should account for at least 25% of the development effort and many people argue that 50% is a better proportion. Others say 80% of the development time is a better figure. Obviously, the methods and practices built into the Rational Unified Process are designed to help reduce the number of errors in your code through the use of requirements management and model based development, frequent testing, and the iterative approach that helps to incrementally improve the overall quality of your software.

What the Rational Unified Process doesn't aim to do is eliminate the need for the full coverage functional, performance, and analysis testing that should be part of every development process. The Rational toolset includes an extensive range of testing tools that you can use both individually, and integrated with other tools in the Rational toolset. This helps with the collation of the information about the tests, as well as executing the tests and providing communication and integration between the test assets (the tests and their results) and the application components.

Monitoring all of this information and tracking the effects of the dependencies between different components and the tests executed on those components can be tricky. This is where the Rational software excels.

This tutorial focuses on the testing process: how to test components and how to collate and report on the information using the Auction example as a guide. The role of requirements in the testing process is also examined, as well as how to integrate testing information between the testing applications and Rational ClearQuest, the defect management tool.

Start by reviewing how testing works in the overall development phase and how the original requirements and the defect tracking system provided by Rational ClearQuest work through the testing process.

## Integrating testing with development

Many teams divide the development of an application from the testing of an application, as if the two processes are completely separate. Testing is seen as the process that occurs at the end of development. It is designed to ensure that the application meets requirements and is used to identify bugs. Often this means that a large quantity of time is devoted to development, with a small component devoted to the testing part of the process.

Within the Rational Unified Process, testing is an integral part of the development iteration. Unlike the situation outlined above, testing occurs throughout the development process and as part of each development iteration. Problems identified during each iteration can then be fixed and resolved as part of the next iteration.

This incremental style combines the development and testing phases and enables

you to continually improve and monitor the quality of your software. In the short term this produces better quality releases. In the longer term, it spreads the testing phase over the entire period of development, effectively increasing the amount of time spent testing without pooling the entire sequence at the end of development.

This process also has the effect of improving your time to market as it helps to reduce the time between the initial 'safe' version and the final release.

## Feeding off requirements and change requests

One of the most important parts of the integration between the different Rational software components is that you can use the requirements for your Web auction system that you generated back in [Part 1](#) of this series to drive the tests that verify whether the application achieves its functional specification.

This link between the original requirements that you generated in RequisitePro and the tests that you create and use to verify the application is critical to ensuring that your application is of the highest quality.

Requirements and the tests you can create based on their definition fall into both of the main test categories: functional testing and performance testing. Functional tests match requirements with the application abilities. Performance tests match the required performance to the achieved performance. Outside of the requirements specification, there are also other tests that can be used to test your application. These can be conducted on a component by component basis and can be used to identify potential bugs like memory leaks, and component-specific performance testing.

Rational PurifyPlus handles these tests, so let's start by having a look at PurifyPlus and how to use it to test and report the results.

---

## Section 3. Using Rational PurifyPlus

### Rational PurifyPlus in software development

Rational PurifyPlus provides a suite of tools that can analyze and monitor your application development during startup. The suite consists of three tools:

- **Rational Purify** finds runtime errors and memory leaks in your

application.

- **Rational Quantify** provides performance analysis on your application, building a profile for your application's startup so that you can identify performance constrictions.
- **Rational PureCoverage** ensures that your entire code base is thoroughly tested.

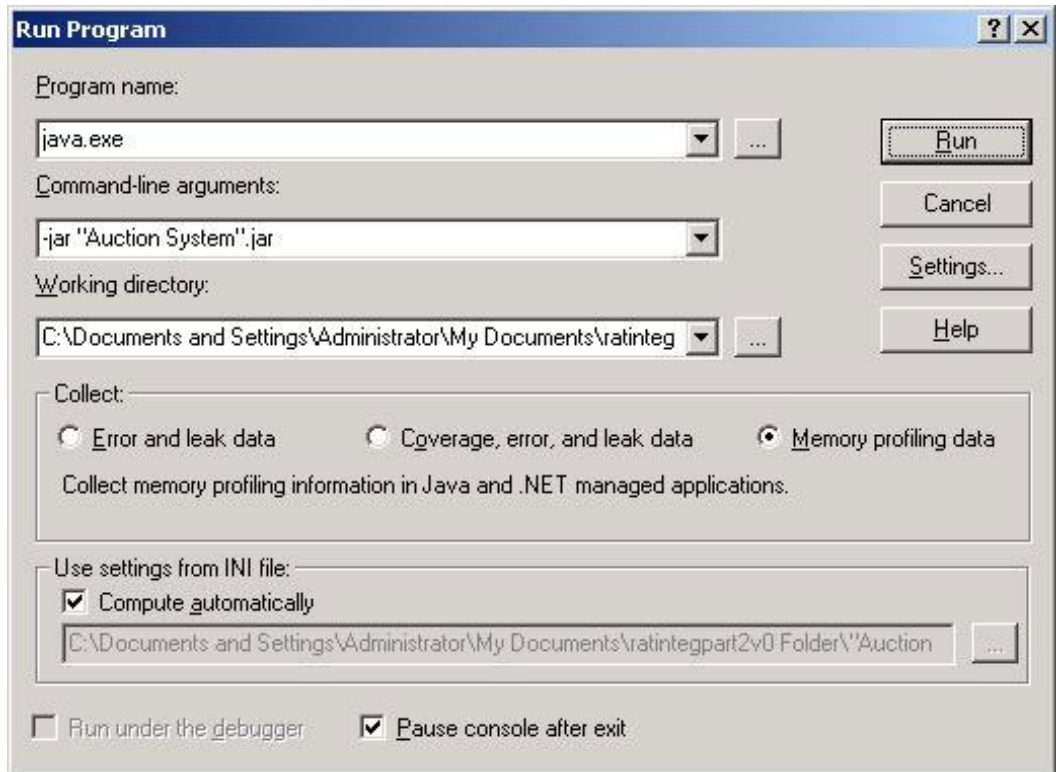
These tools work individually, and together, to ensure your application is tested for basic errors, performance, and code errors.

## Using Rational Purify

Using Rational Purify to monitor memory usage relies on taking an initial reading of the typical memory usage of your application to provide a baseline. You can then execute the code where you suspect a memory leak. By comparing the memory usage between the two you can get a fair report on which methods might be causing memory problems. Armed with the information about the methods, you can then examine the objects and class definitions that are the root of the problems. These details are beyond the scope of this tutorial, but we will look at the basic process.

To use Purify on your application:

1. Run your Java program with Rational Purify. Rational Purify provides a simple interface for starting your applications, shown in Figure 1.  
**Figure 1. Rational Purify provides a simple interface for starting your applications**



2. Take a snapshot of the memory during normal use for a portion that you don't suspect of causing a problem. On each startup you'll get a code launch profile. You can see the progress of the application and obtain detailed information about the startup process through the Data Browser view shown in Figures 2-5 . There are four tabs: Memory that shows the memory usage; Call Graph that demonstrates the execution sequence; Function List View that shows the complete list of functions in the application, the number of calls and object memory; and the Object List View that shows a list of the objects in an application. Example output from the Auction application is shown in the four figures below.

**Figure 2. The Memory tab of the Data Browser view**

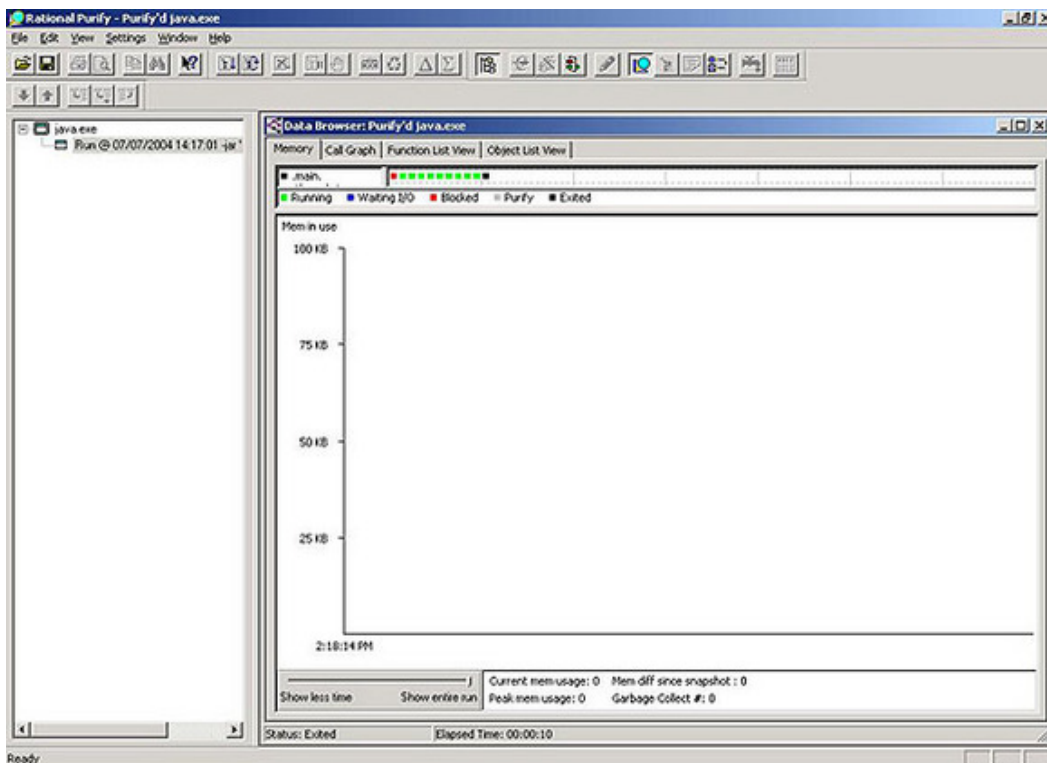


Figure 3. The Call Graph tab of the Data Browser view

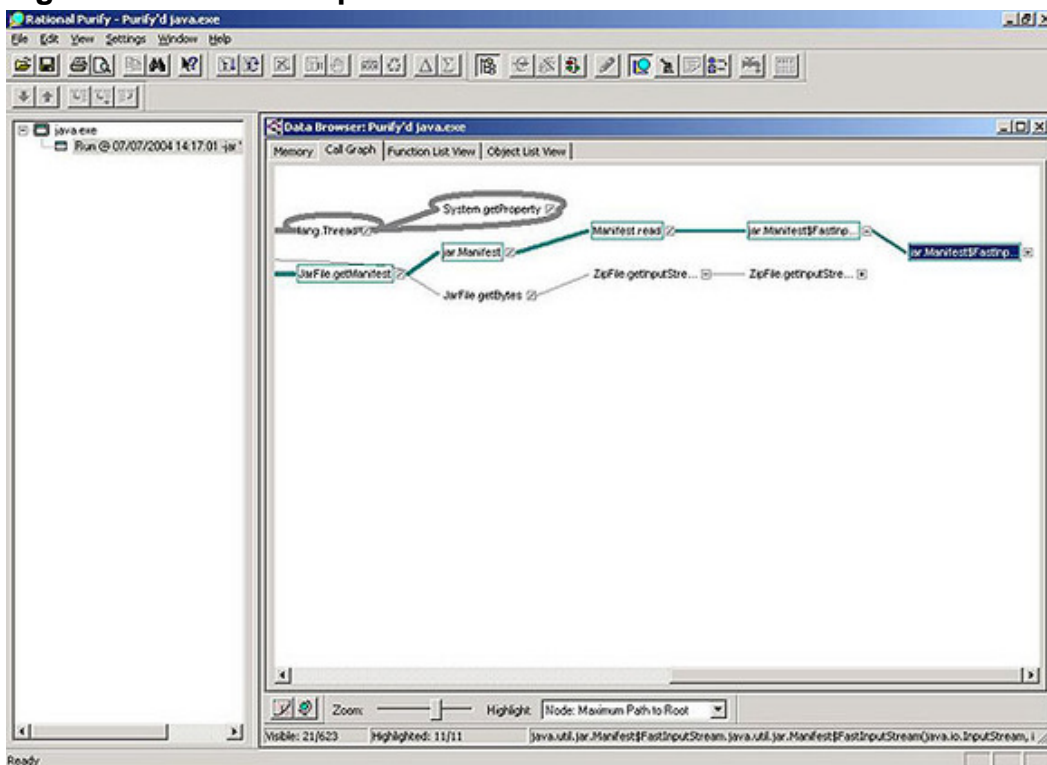


Figure 4. The Function List View tab of the Data Browser view

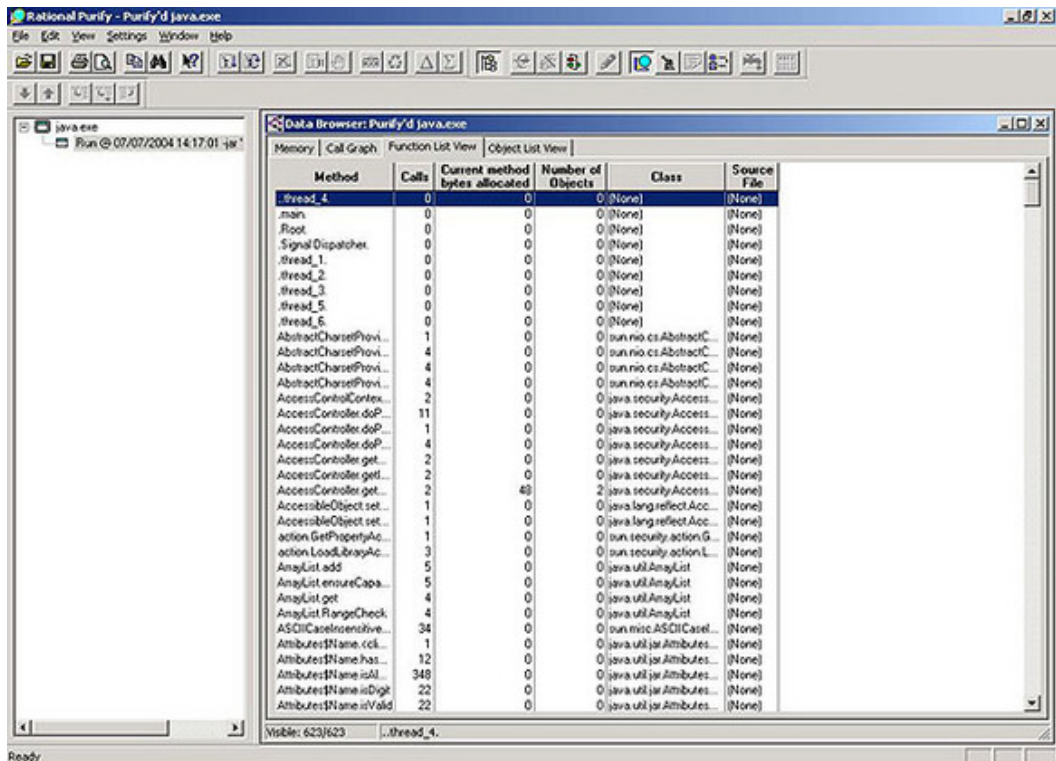
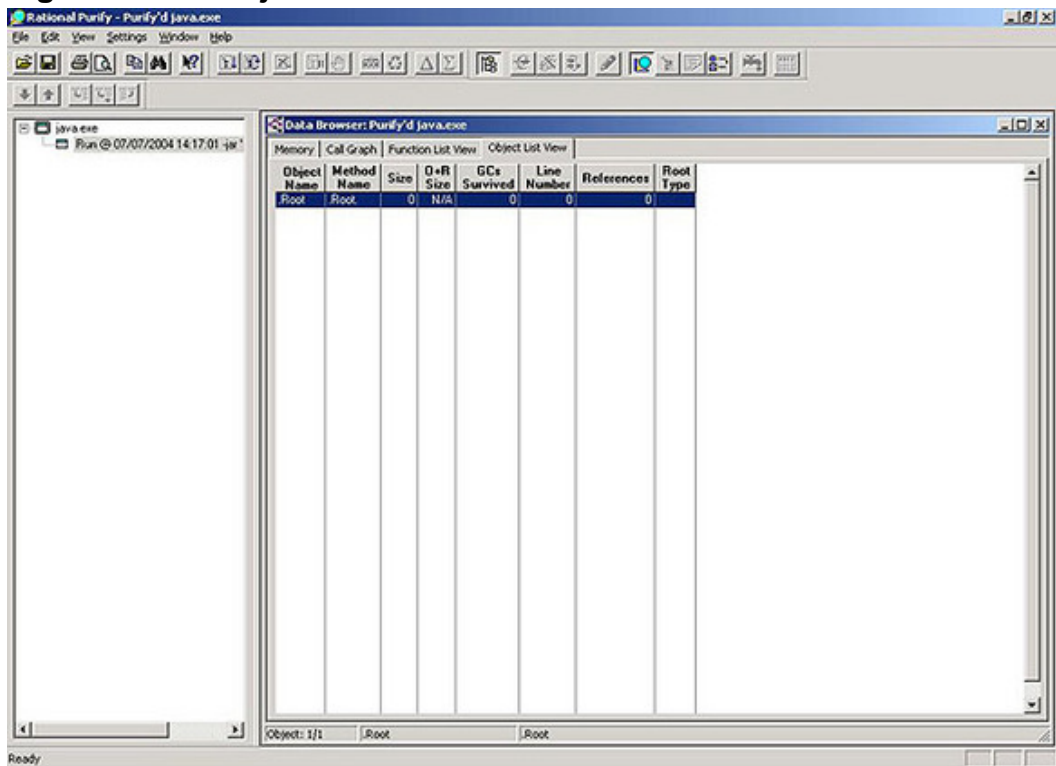


Figure 5. The Object List View tab of the Data Browser view



- Run the code that might be causing memory leaks.

4. Compare the two snapshots -- standard and suspected memory leak -- to determine which parts of the application might be causing a problem.
5. Pinpoint the objects that are causing the problem and investigate them more fully to determine where the leaks are occurring.

When you find a fault, use the Error Defect submission system described in the next panel.

## Submitting an error defect to Rational ClearQuest

You can submit an error defect into Rational ClearQuest from any Rational PurifyPlus application.

1. From an Error view, select all the text from the error report that you want to submit as the description for the defect.
2. Right-click on the text and select **Submit ClearQuest Defect**. A defect form, shown in Figure 6, opens and is pre-filled with the selected text.  
**Figure 6. The defect form**

The screenshot shows a 'Submit Defect' dialog box with the following fields and controls:

- Tabbed interface: Iterations & Notes, Unified Change Management, Requirements, Main (selected), Test Data, Environment, Attachments.
- ID: auc00000014
- State: Submitted
- Headline: [Text Field]
- Suite Project: [Dropdown]
- UCM Project: [Dropdown]
- Owner: [Dropdown]
- Priority: [Dropdown]
- Severity: 1-Critical [Dropdown]
- Customer Priority: [Dropdown]
- Keywords: [Text Area]
- Symptoms: [Text Area]
- Description: [Large Text Area]
- Buttons: OK, Cancel, Values (dropdown)

3. Add any additional information, such as priority and severity, to the form. If you have generated Purify data and reports and saved that information in a file, attach that information to the defect entry.
4. Click **OK** to submit the defect.

Once the defect has been reported you can use the techniques that you used in Part 4 of this series to monitor and report the defects in the system and report on the progress.

## Performance testing with Rational Quantify

Rational Quantify is all about finding performance constrictions in your application. In a typical application there are numerous places where a constriction can cause a problem. In your Auction application, the performance is vital because users expect a faster response from a Web site than they do from a typical desktop application. Finding these constrictions can be difficult.

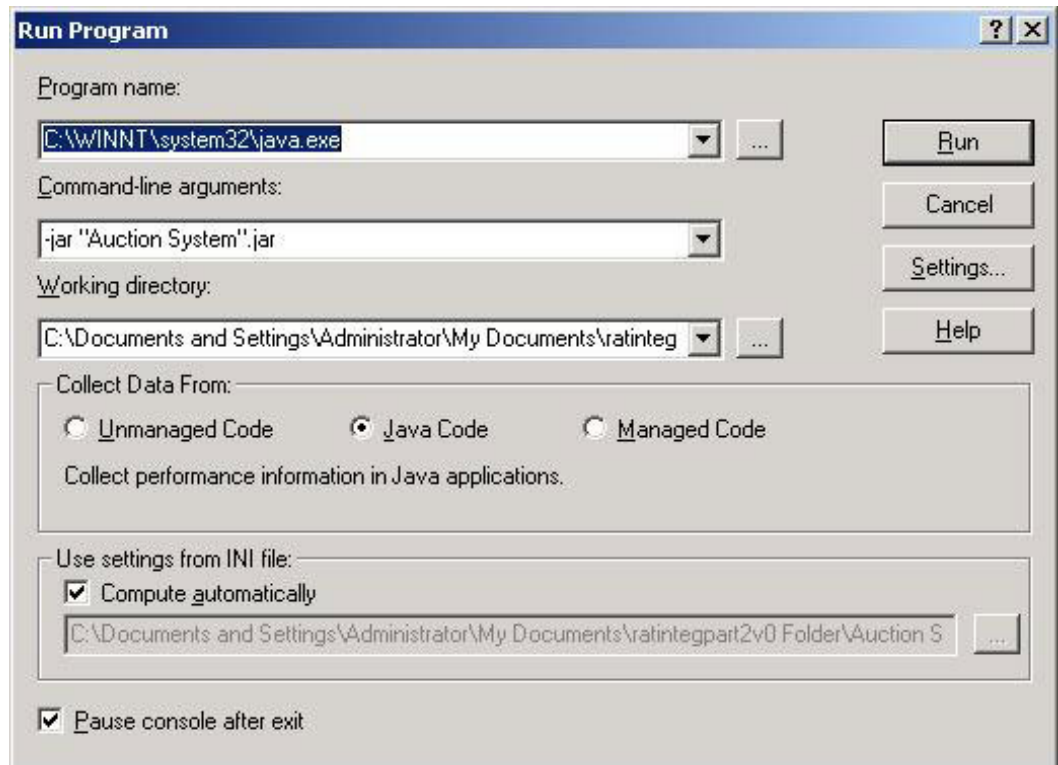
Finding constrictions is best achieved through profiling that monitors the time spent

different components of an application. By studying the timing and execution of these components you can identify which parts of the application are taking the most time.

To use Rational Quantify for profiling:

1. Run the program with Rational Quantify to collect the performance data (see Figure 7).

**Figure 7. Running the program with Rational Quantify**



2. Use the performance data to identify which components are spending the most time.
3. Modify the code to reduce the constrictions.
4. Rerun and compare your new results with the original tests.

Results can be reported directly into Rational ClearQuest from Rational Quantify in the same way as from Rational PurifyPlus.

## Coverage analysis

The purpose behind Rational PureCoverage is to ensure that you have fully tested

all of the components of your applications before it is released to your clients and customers. The problem is to analyze which components make up the bulk of your application and therefore to identify which components will benefit from the bulk of the testing process. Rational PureCoverage will also identify which components and even lines of code in the application have not been run and therefore not been subjected to the full range of tests.

Rational PureCoverage should be used throughout the development process so that you can monitor the tested components. Using the data generated by Rational PureCoverage as you go through each iteration you can determine which components are new in the system and which deserve more attention as you test and monitor the application.

When you execute Rational PureCoverage on your application you can identify:

- Each function defined in the application but not tested
- The number of lines of code that have been 'missed' (not executed)

With this detail you can identify the components of the application that should be tested more stringently. If you identify a coverage problem, you can submit the coverage defect directly into Rational ClearQuest from Rational PureCoverage.

## Submitting a coverage defect to Rational ClearQuest

Coverage defects can also be reported from Rational PureCoverage:

1. In the Module view of the File View tab of the Data Browser window, select the file or function with the coverage defect that you want to report.
2. Right-click the file or function and choose **Submit ClearQuest Defect**. As before, you'll get the Rational ClearQuest defect form.
3. Add any other information that you want to record for this defect.
4. Click **OK** to submit the defect.

Once the defect has been reported, use the techniques from Part 4 of this series to monitor and report on the defects in the system and report on the progress.

Rational PurifyPlus is only one part of the testing process. There are other tools and environments available for supporting functional and performance testing that are useful when making a comparison between our original requirements and the final application. The best environment for this is Rational TestManager, so let's have a look at how to use Rational TestManager for some in-depth testing and

investigation.

---

## Section 4. Rational TestManager

### Using Rational TestManager

The Auction application is Web based, but this doesn't mean you can rely on it working across a range of platforms and environments. Unfortunately the Internet and Web browsers are not the utopia of platform independence that we all expected. Whether you are using Internet Explorer, Netscape, Mozilla or even OmniWeb you have a range of different versions. On the server side there are a range of different deployment platforms from Sun, Apache, IBM and many others. Again there are a range of different versions and, worse, a range of different combinations.

Testing all these different combinations is time consuming by hand. Furthermore, testing all of the different components and possible input and output values for all the different components also by hand could take days. Some type of automation of the process is helpful.

Rational TestManager is a complete solution for testing applications; it makes up elements of Rational Functional Tester for Java and Web, Rational Robot and Rational Performance Tester. Designed to work in a team environment as a central testing solution, you can build suites of tests that can then be executed on an application at any stage during development, and during any iteration of the development process.

Each time you run a test the results are recorded and stored and you can use this information to build up an idea of the faults in your system and over the long term to record and monitor the statistics surrounding the bugs in the Auction application, or the application you are developing. For example, by monitoring and tracking the information you might find that one particular part of the application is generating the largest quantity of errors and therefore requires additional attention during the debugging process.

TestManager also provides a distributed testing facility, allowing you to spread the tests for different components of an application across a number of machines. You can use this either to execute a small number of tests and parameters across an application in a shorter time, or to provide a much wider range of parameters, and therefore more extensive tests, without the test process taking days or weeks. Distributed testing also enables you to test across a range of different platforms using the same test inputs, components and monitor the results from a central

location.

Once all of the information from the various components has been collated, Rational TestManager then provides a convenient way to report and summarize the information. This is useful to all the different groups involved with the project. Project analysts, architects, developers, quality assurance and project managers can all use the information provided by TestManager to gauge the status of the project.

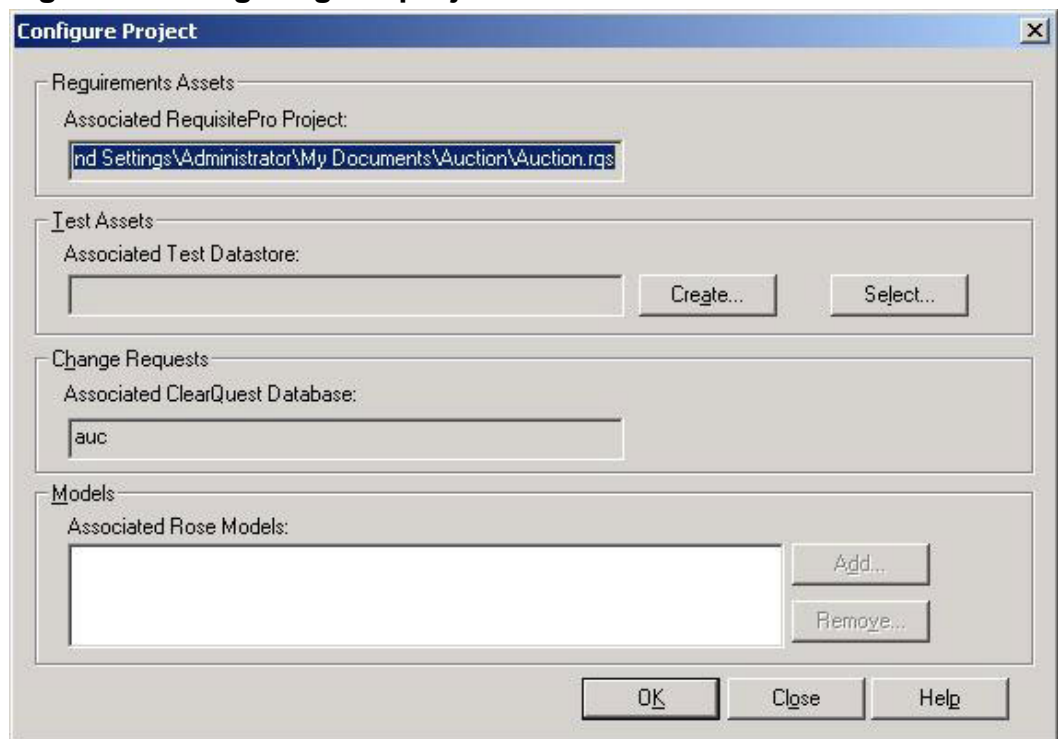
## Associating your project with Rational TestManager

The first stage to using Rational TestManager is to associate a Test datastore, which is used to store the results from your tests, with the rest of your project. Do this through the Rational Administrator application, which you've used in previous parts of this series.

To create a new datastore for your project:

1. Open Rational Administrator.
2. Right-click on the Project you want to configure and choose **Configure**. The Configure Project window opens.

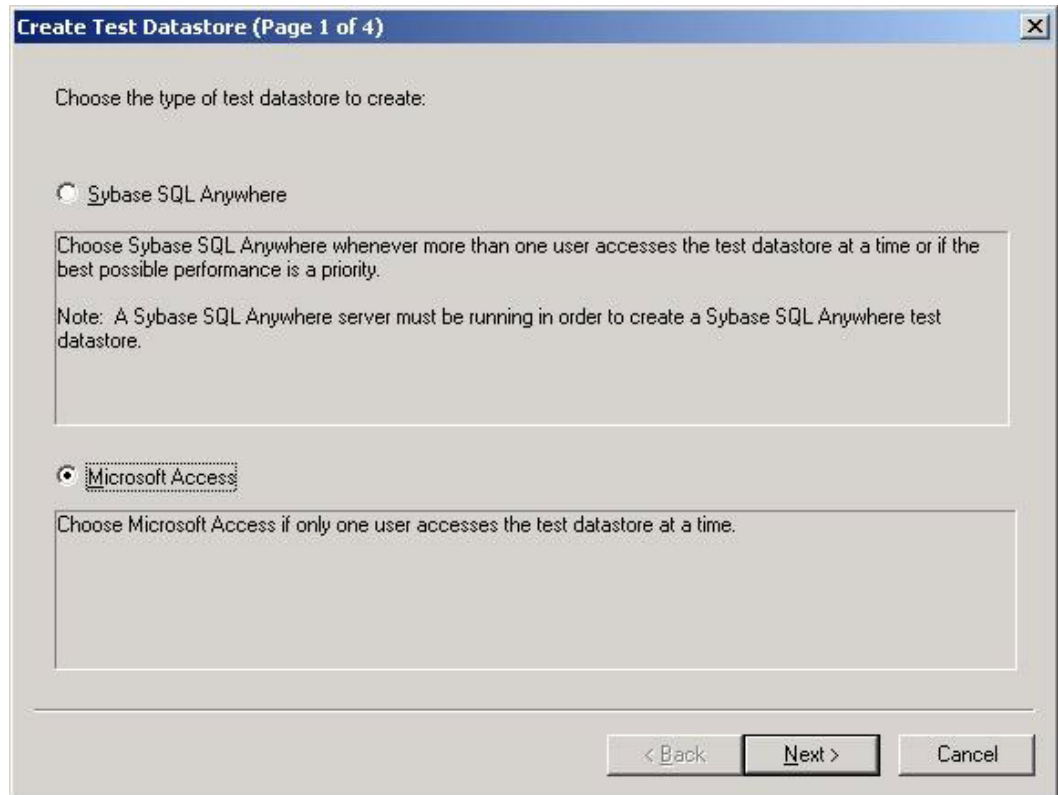
**Figure 8. Configuring the project**



3. In the Test Assets section of the window, click **Create** to create a new test

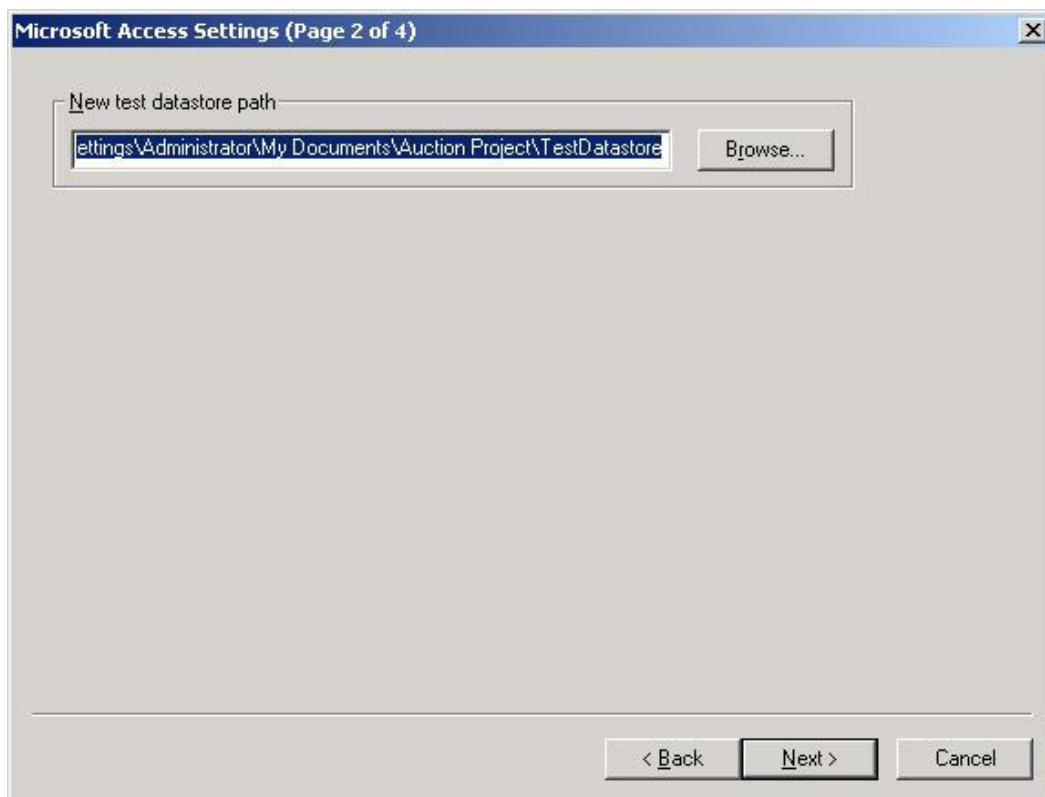
asset database. You can also use an existing test asset database if you have one. You'll be creating a new one for this tutorial. The database choice window opens (Figure 9). Choose **Microsoft Access** for the data store type. Click **Next**.

**Figure 9. Database choice window**



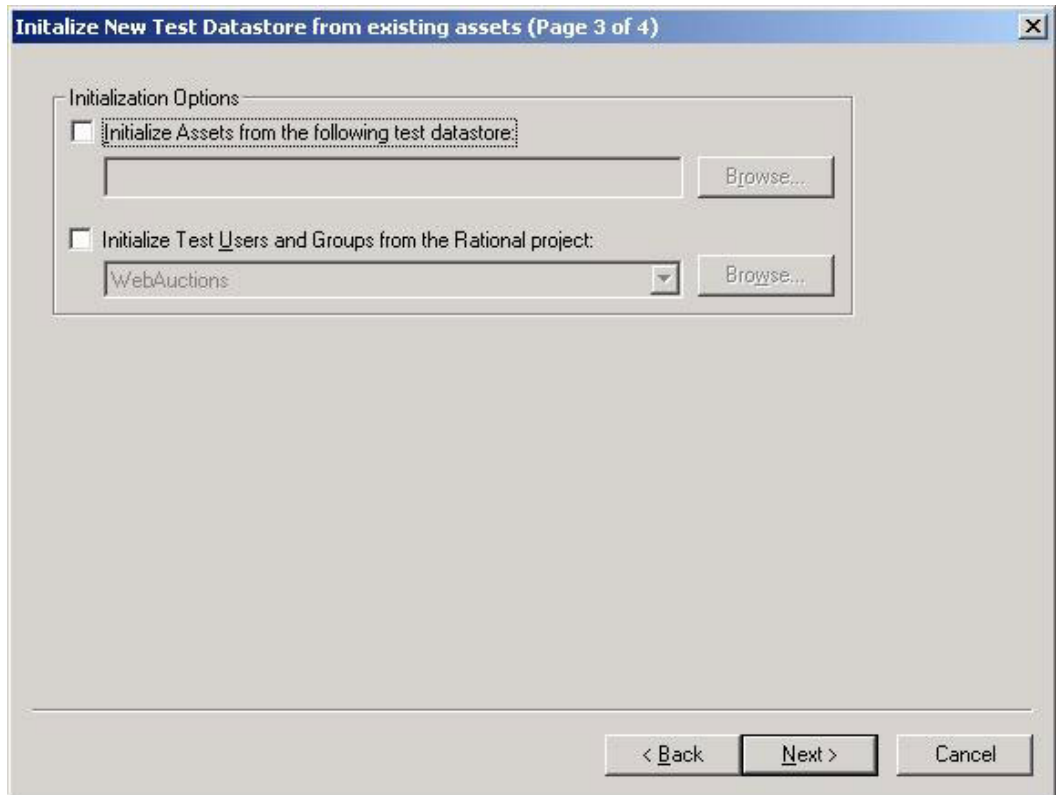
4. Choose a location for the Test Datastore, as seen in Figure 10. Click **Next**.

**Figure 10. Location for the Test Datastore**



5. Optionally, choose to initialize the database with assets from another data store (useful if you are migrating or creating a new major release of a project), or to initialize the user and group information from the Rational Project (to use the same authentication information you are already employing). Click **Next**.

**Figure 11. Initialize the database with assets from another data store**



6. A summary of the work to be completed is displayed. Click **Finish** to create the test datastore.

## Test types

Rational TestManager supports two main types of tests: functional tests and performance tests.

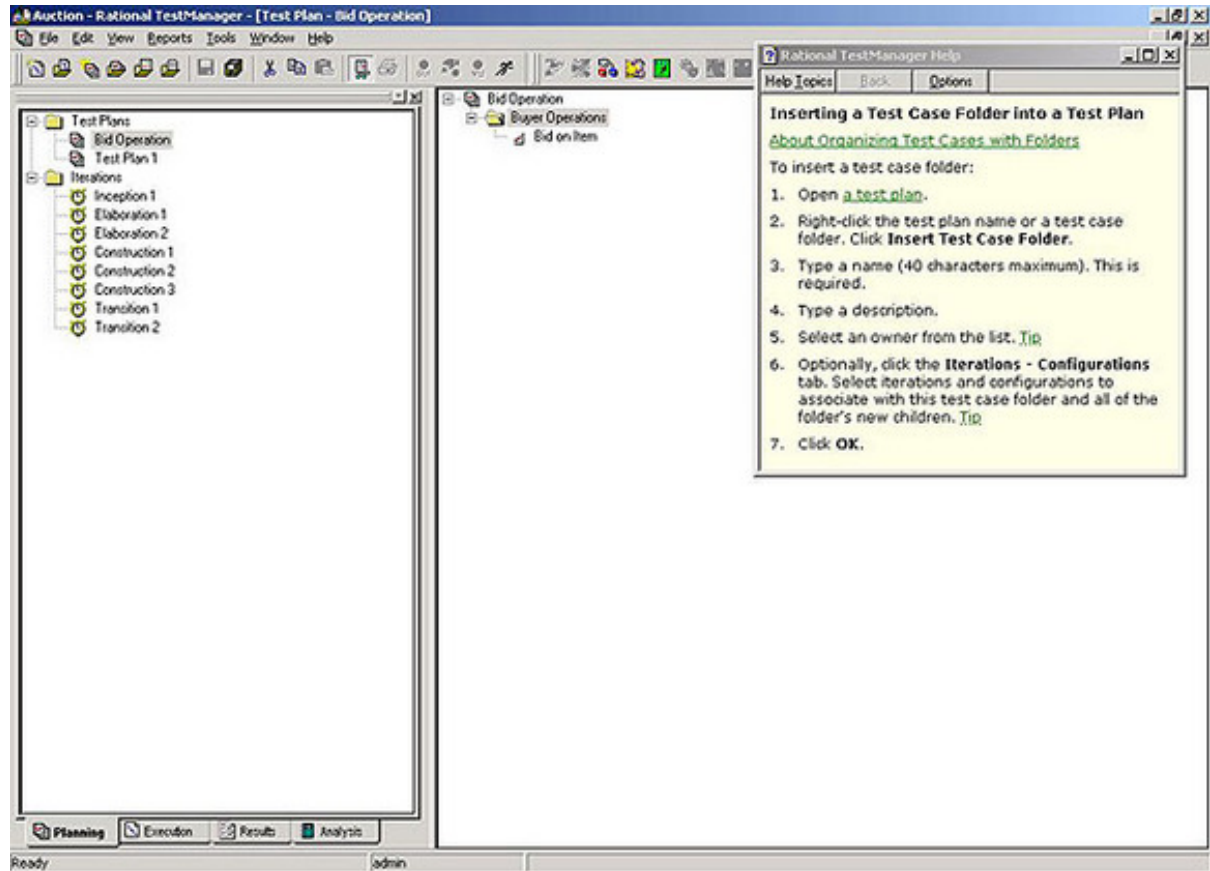
With functional testing you need to determine the features that you want to test and how to identify whether the features pass or fail -- i.e., whether the actual operation matches the functional definition. Functional tests can be used to achieve different target results depending on what you test. For example, you might want to determine whether the current release of the application matches the current requirement specification. A failure indicates that the application has not yet been completed.

Performance tests can also be used to indicate a variety of situations. Ultimately, they are all based on performance, but you can use the output to identify potential constrictions or simply identify components that do not match the required performance level.

## Inside Rational TestManager

Following is a sample window from Rational TestManager.

**Figure 12. Sample window from Rational TestManager**



The main window is split into two sections. The panel on the left provides an interface to the various parts of the Test project. The tabs at the bottom switch between the different sections: Planning, Execution, Results and Analysis. These panels are used to drive the test process:

- Planning provides an interface to the test plan structure, and the iterations (including any parameters) used to test those components.
- Execution is the interface for controlling the execution of the tests.
- Results provides the results for each test execution.
- Analysis is the interface into analyzing and collating the raw results into meaningful reports on the status of the tests.

The larger panel on the right displays details about individual components, for example the properties of an individual test, or the results from a test. The panel also displays other windows used during the testing and management process, for example the list of test inputs and reports and results.

## Testing process

Tests within Rational TestManager are centered around and controlled by test plans. Creating a test plan enables you to define the parameters of the test in terms of determining the objectives for the test. A test plan is not a static entity. Use a test plan throughout the life of the project to define and build new tests and new situations as the project progresses.

A test plan consists of a list of test cases. Individual test cases can be organized within a test plan in a hierarchical fashion to make it easy to organize and identify the different tests in the system.

There are in fact a number of steps to building the test plan. Therefore, to create a series of tests in TestManager, follow this sequence:

1. Define test inputs. Test inputs define the entities that are tested by the test cases in a test plan.
2. Create test plan. Test plans hold the information about input, test cases and other components.
3. Create test cases. Test cases define the individual tests that will be performed. You can organize these into test case folders for easier identification.
4. Define the configurations. Test configurations are the OS environments that you want to use when testing the system. In a Web environment, such as our WebAuction application, configurations are used to define the server computers that will be accessed and the client computer configurations that will access the server.
5. Define the iterations. Iterations define when the tests will be executed.

Let's take a closer look at the individual components and how to configure them.

## Test inputs

Test inputs are used by Rational TestManager to define what is to be tested. Rational TestManager supports a number of built-in test types that can be used to

test and report on specific elements of a project. Two of these types are directly related to earlier stages of the development process, requirements and application models.

Requirements from Rational RequisitePro are automatically available. You can see a list of the requirements from your own requirements model.

Model elements from Rational Rose are also automatically available as test inputs. Any of the model elements can be used as a potential test input source providing you have associated your Rose models with the rest of your project through Rational Administrator.

You can also define any number of custom inputs through the Test Input Adaptor (TIA) system. You can use a custom plug-in type using the Rational TestManager extensibility components provided with TestManager to build your own test inputs for your application.

## Creating test cases

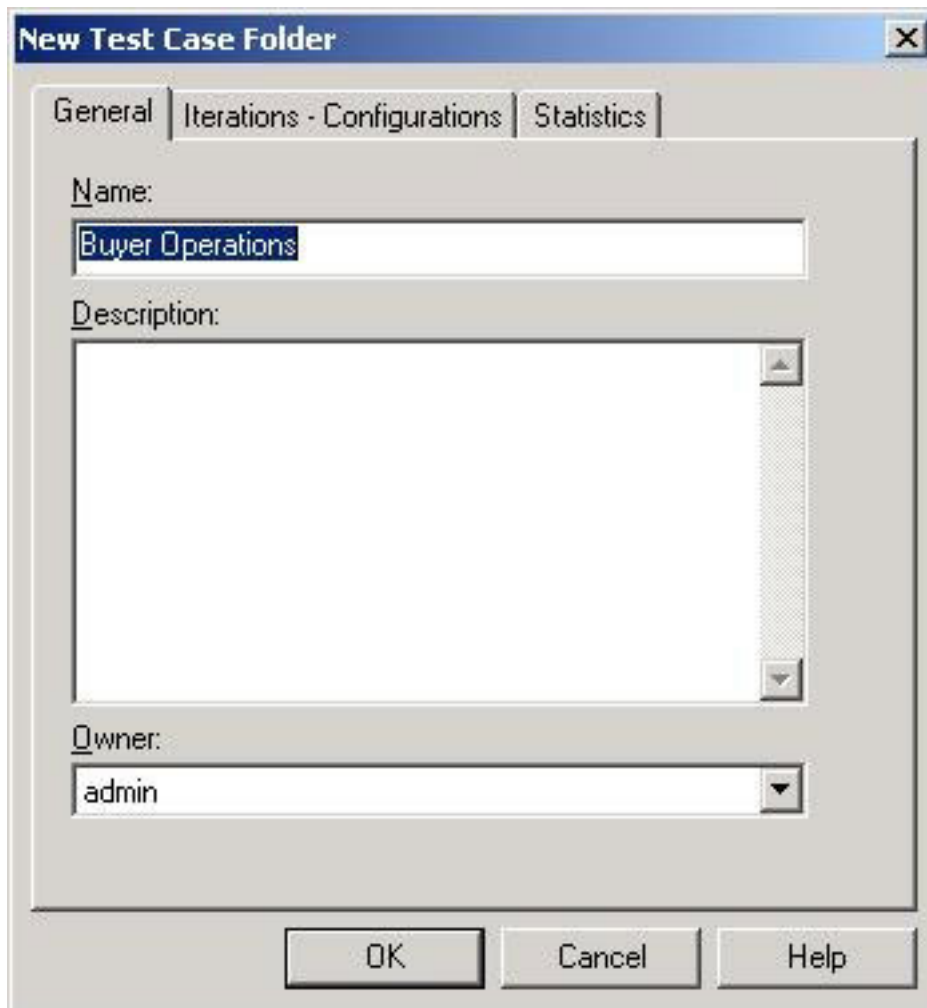
Test cases define the relationship between your test inputs and what you want to test. For example, we may want to create a test cases that maps between the requirement for an auction image an the implementation of that component in the system.

Test cases can be organized through a series of folders that can be used to divide up the test cases according to the components you are testing, the testers involved in the test process or any other structure that makes sense within the confines of your test plan.

To organize your test case folders:

1. Right-click on **Test plan** and choose **Insert Test Case Folder**.
2. Fill in the details for the folder as shown in the Test Case Folder properties window (Figure 13).

### **Figure 13. Test case folder properties window**

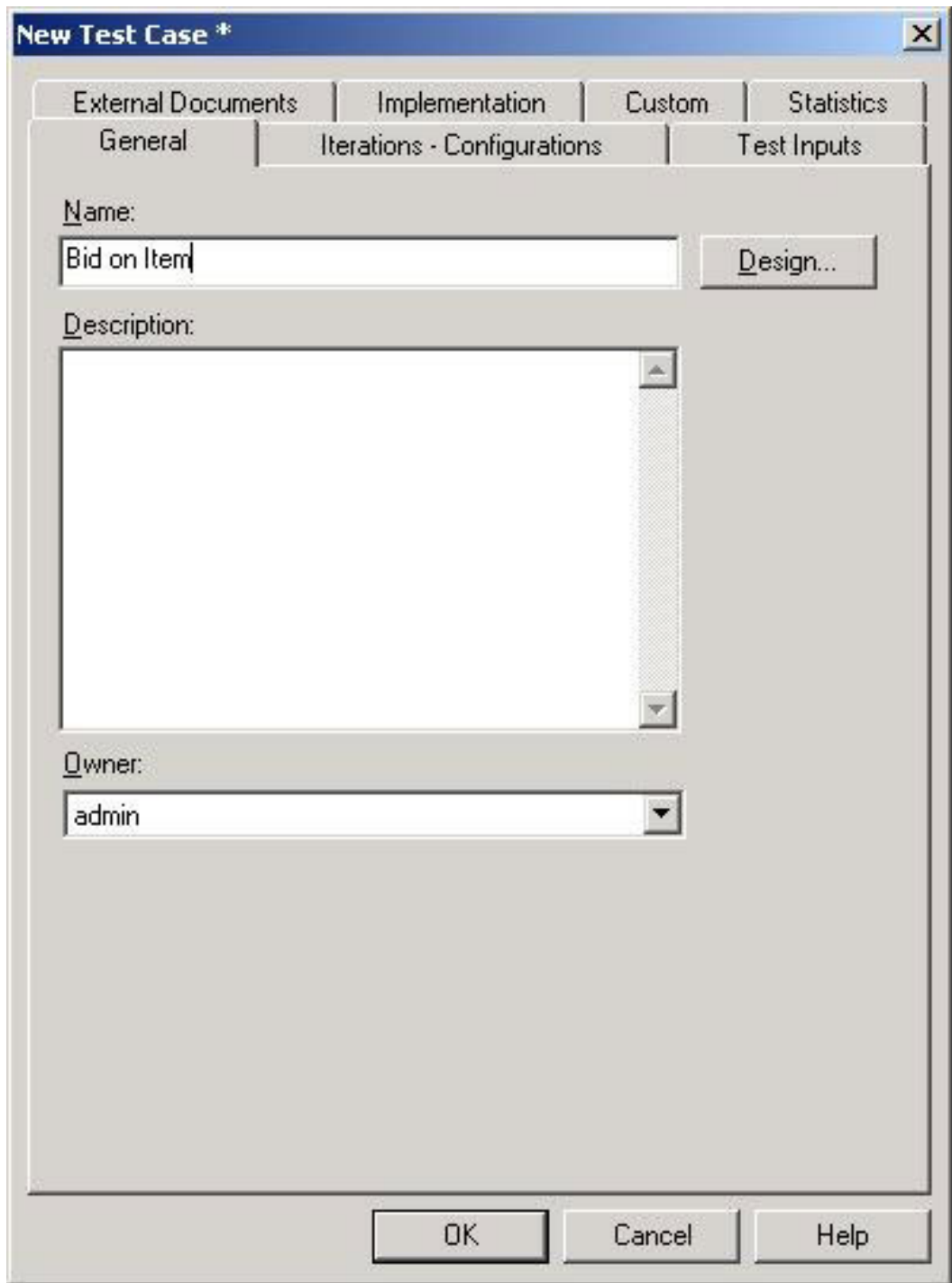


The image shows a dialog box titled "New Test Case Folder". It has three tabs: "General", "Iterations - Configurations", and "Statistics". The "General" tab is selected. Inside the dialog, there are three main sections:

- Name:** A text box containing "Buyer Operations".
- Description:** A large empty text area with a vertical scrollbar on the right.
- Owner:** A dropdown menu showing "admin".

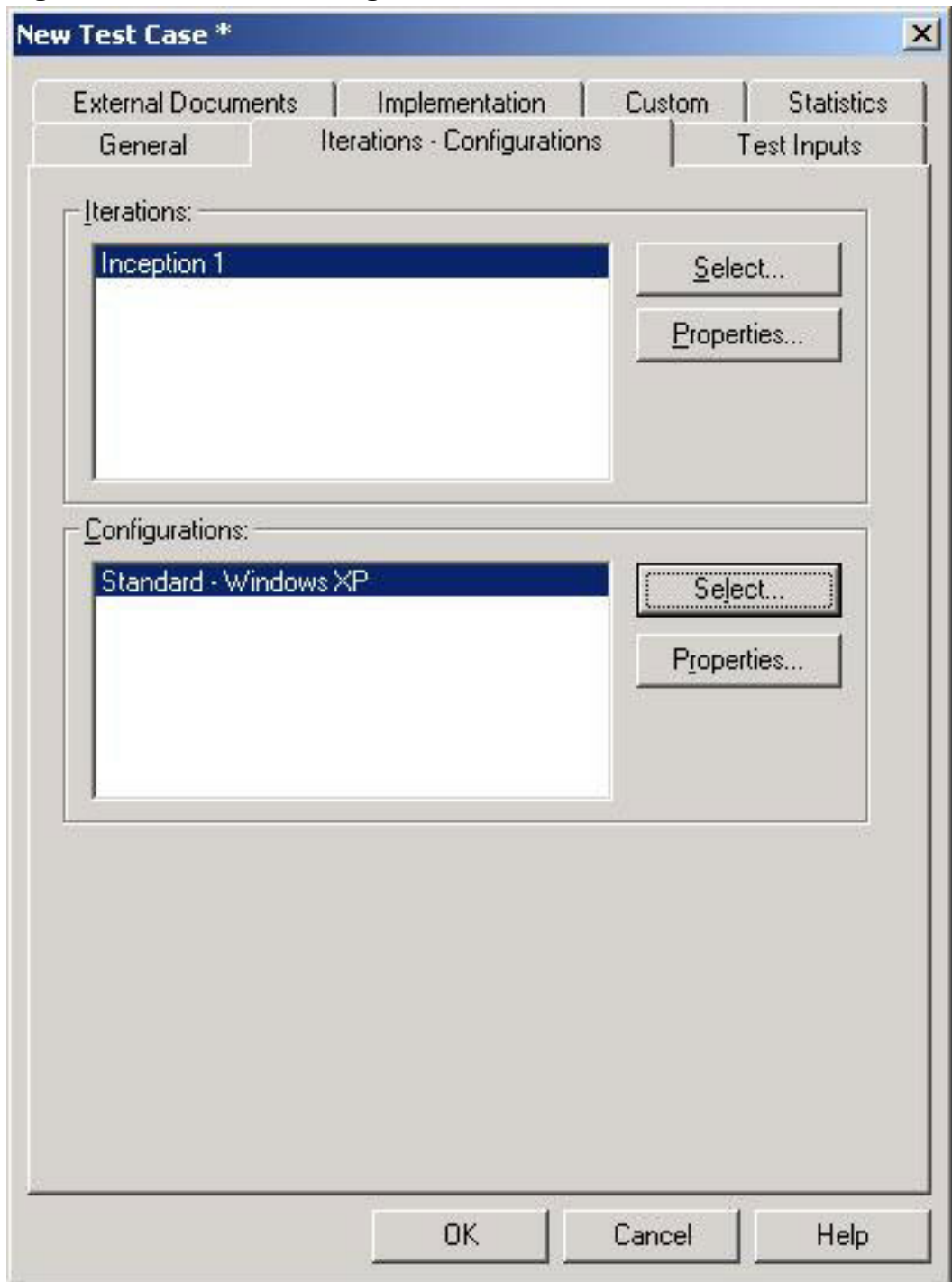
At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

3. Define the name of the folder, the description, the owner (within your testing team) responsible for the test cases in the folder and the number of iterations for the tests in this folder. Iterations are covered in a later panel in this section.
4. To create a new test case, right-click a test case folder and choose **Insert Test Case**.
5. Fill in the form as shown in Figure 14.  
**Figure 14. New test case form**



6. Give the test case a name, a description, an owner.
7. Switch to the Iterations-Configurations tab. Iterations define when the tests are run. Configurations define on which platforms and machines the tests should be run. Both will be covered in later panels.

**Figure 15. Iterations-Configurations tab**



Further panels in the test case allow you to specify the test inputs, additional documents and the implementations of the test case.

## Configuration management

Tests are designed to compare the expected results with the results generated by the test scripts. Different configurations are used to enable you to test your applications using these test scripts under different operating systems, browsers and other configurations.

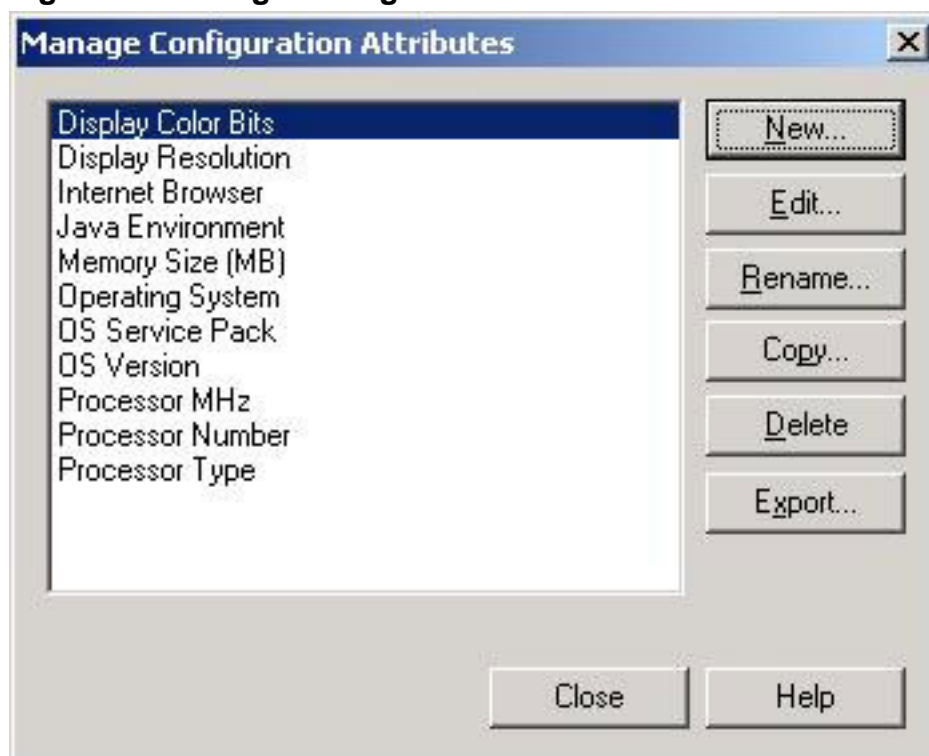
By using distributed testing you can test the functionality and/or performance of your application across a range of different configurations simultaneously.

To create a configuration, define the attributes for a given configuration. Attributes specify the operating system and browser and any other attributes you require. Once you have generated a list of valid configurations, define the configurations that you want to test in your test plan, and then associate specific test cases with the configurations that you want to test.

To define configuration attributes:

1. Click **Tools > Manage > Configuration Attributes**.
2. Click **New**, as shown in Figure 16.

**Figure 16. Manage Configuration Attributes window**



3. Fill in the form as shown in Figure 17. Specify the attribute and its possible values. The figure shows browsers but it can be any definable attribute that is required by your application. These attributes are used to define configurations. Repeat this operation for each required attribute in a configuration.

**Figure 17. Configuration Attributes Properties window**

**Configuration Attribute Properties**

General | Statistics

Name:  
Internet Browser

Description:  
[Empty text area]

Owner:  
admin

Source of values:  
 Dynamic  List

List values:  
[Empty text input] [Add] [Remove]

- Internet Explorer 4.x
- Internet Explorer 5.x
- Internet Explorer 6.x
- Mozilla 1.2
- Mozilla 1.7
- Netscape 4.0
- Netscape 7.0
- OmniWeb 4.x
- OmniWeb 5.x

OK Cancel Help

To define the configurations to test:

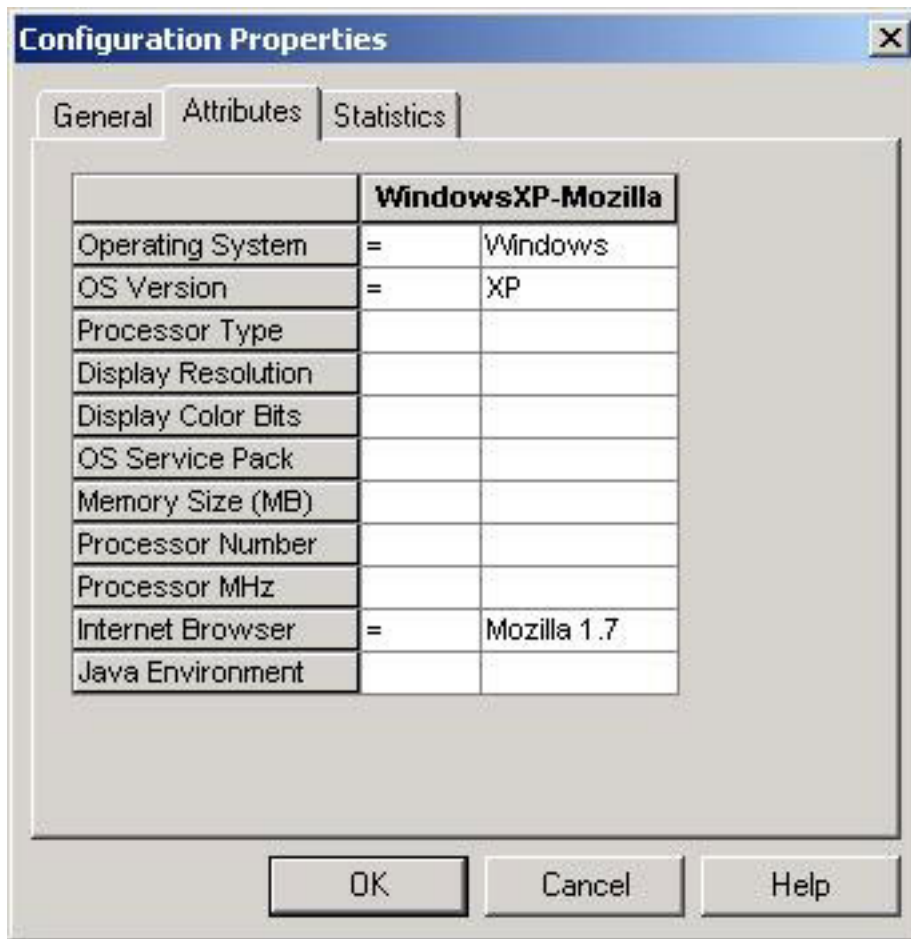
1. Click **Tools > Manage > Configurations**.
2. Click **New**.
3. Fill in the form as shown here in Figure 10. Give the configuration a name that can easily identify the configuration. For example, this one is named WindowsXP-Mozilla.

**Figure 18. New Configuration window**



4. Change to the attribute tab and fill in the attributes information that you defined in the previous sequence.

**Figure 19. Configuration Properties dialog**



Once you have defined the configurations, associate the configuration with the test case that you defined in the previous panel.

## Iteration specification

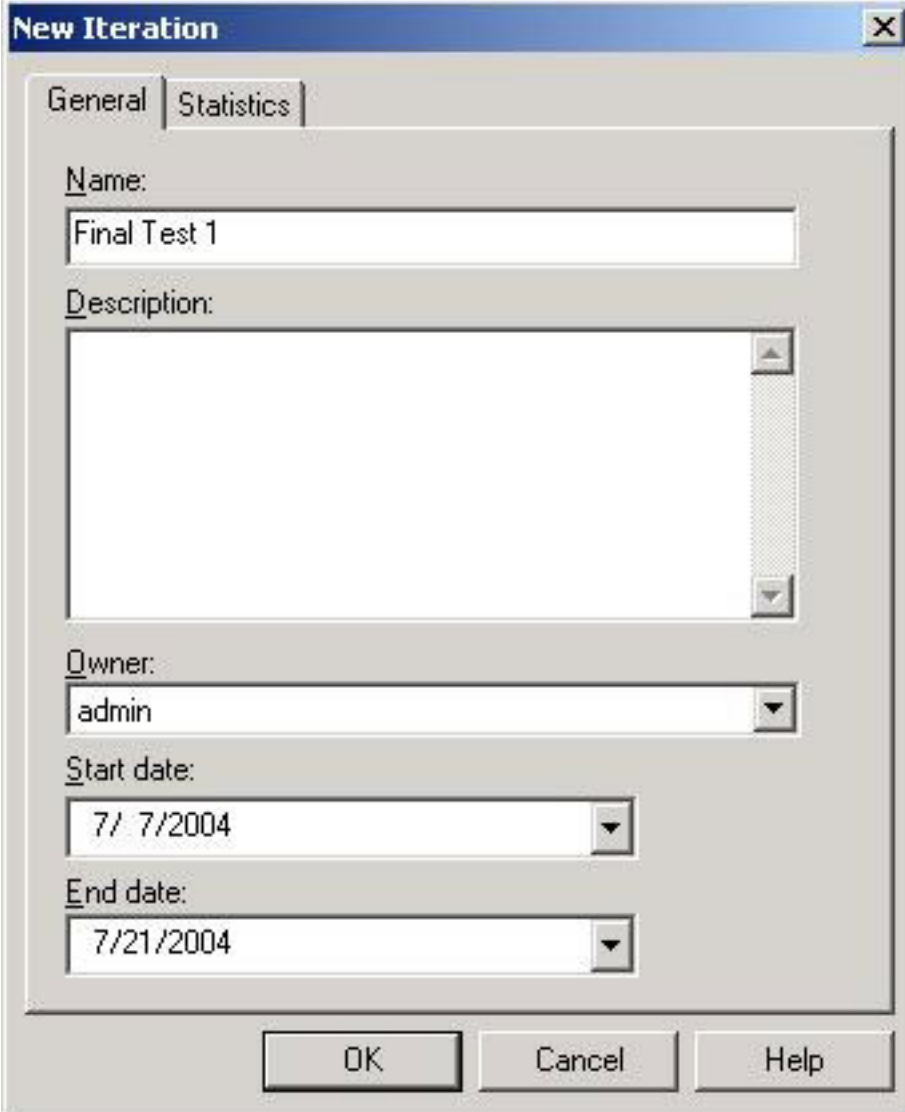
Iterations specify when different test cases should be executed during the development of your application. For example, imagine that your analyst has defined that during the first iteration of your development process, you need to test certain core components. Create the test cases to test these components and then create an iteration that includes these tests.

In later iterations you might not need to test these items so don't include these test cases in the test plan. You can mix and match and create as many different iterations to work in combination with your test cases as you require. The definition can be quite complex and it's likely that you will create a new iteration specification for each new iteration of your development.

Rational TestManager includes a standard range of iterations based on the Rational Unified Process, but you can also create your own iterations. To create an iteration:

1. Click **Tools > Manage > Iterations**.
2. Click **New** to create a new iteration. The New Iteration window opens (Figure 20).

**Figure 20. New Iteration window**



The screenshot shows a dialog box titled "New Iteration" with a close button (X) in the top right corner. The dialog has two tabs: "General" and "Statistics". The "General" tab is selected. The "Name" field contains "Final Test 1". The "Description" field is a large empty text area. The "Owner" dropdown menu is set to "admin". The "Start date" dropdown menu is set to "7/ 7/2004". The "End date" dropdown menu is set to "7/21/2004". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

3. Give the iteration a name, a description, and an owner. Iterations are applied according to the dates valid for the iteration of the project so you need to specify the start and end date for the iteration.
4. To actually associate an iteration with a test case, use the Test Case

properties and the Iterations-Configurations tab.

## Designing tests

Up to now you've defined only the features that you want to test -- based on the test inputs -- combined with the configurations that you want to test and the dates that the tests should be conducted (through the iteration specification).

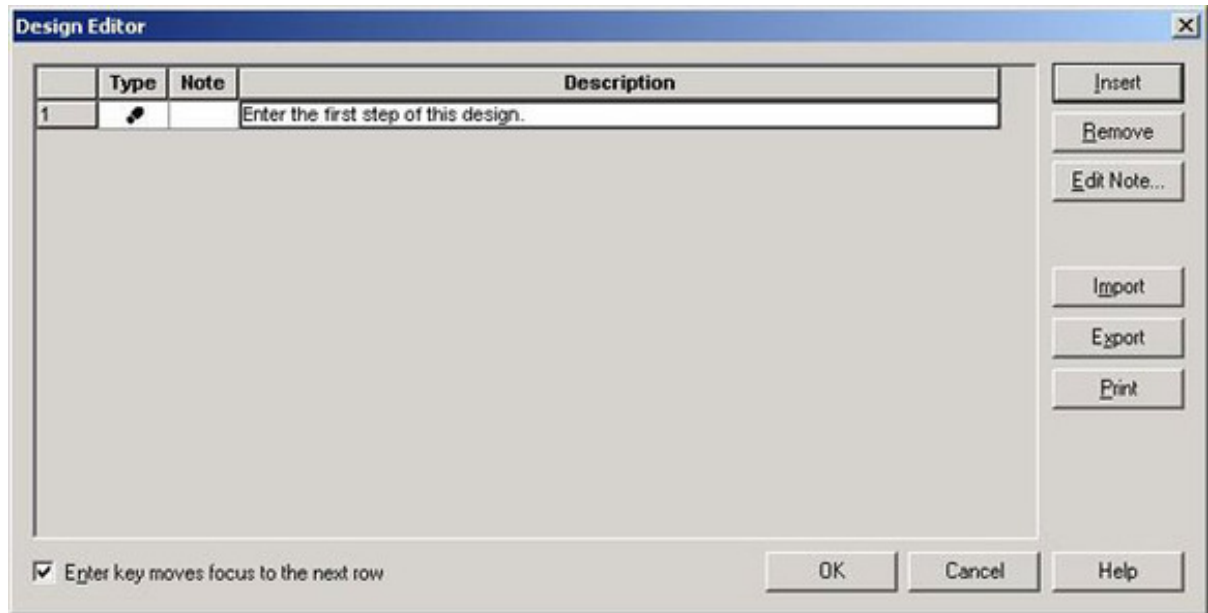
The actual tests are based on a test design. The design defines:

- The steps required to interact with the application to perform the test. For a Web application like your Auction system it will be the URL and any form data required to reach the test component.
- The validation specification. These are the expected results against which you should test the application. For example, if the component should output a value between 50 and 100 then this is the validation specification of the test.
- Any pre-conditions required to set up the application for testing.
- Any post-conditions required to clean up after the test has been run.
- The acceptance criteria, which will be used to determine whether the test case passed or failed.

The full test design is beyond the scope of this tutorial and is application and component specific. Functional tests, for example, combine the test inputs and components with comparison information that allows you to compare real outputs with the expected outputs.

You can see in Figure 21, however, how the Design Editor in the Test Case properties can be used to specify and populate the test case information.

### Figure 21. Design Editor



---

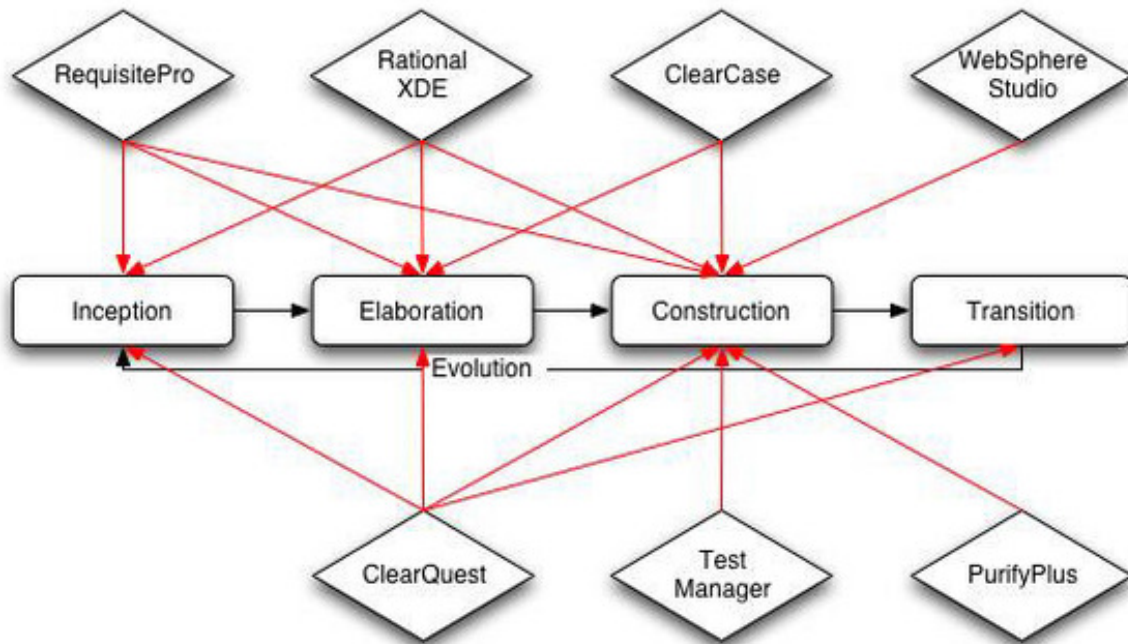
## Section 5. End of the cycle

### Reviewing the development process

Throughout this series you've been looking at the development process of an application and how you can use the various Rational tools individually, and together, to produce an application of the highest quality.

The different tools in the Rational suite have helped through every step of the process. It's worth reviewing the tools, their sequence, and the role during the development process. Figure 22 from Part 1 demonstrates the development sequence.

#### **Figure 22. Sequence of tools in the development process**



Having run through the tool set and the development process throughout this series, you can begin to see the sequence and the integration available that allows you to share information between components and more easily develop top quality applications.

## Preparing the next iteration

You've followed the process from the original requirements, to developing the model and writing the code, to monitoring the changes, defects management, and testing. Now that you've achieved this first iteration of the development, repeat the process.

The next iteration of development improves on this release. The exact sequence depends on the application, but typically it involves some or all of the following steps:

1. The defects and change requests from Rational ClearQuest are combined with the original requirements to generate the next revision of the requirements specification in Rational RequisitePro.
2. The new requirements are used to adjust the models in Rational XDE.
3. The models generate new code that is merged with the other code in the system to generate the application components.

4. The code is updated to fix any defects identified during the testing in the previous iteration.
5. The tests from the first iteration are updated and expanded, if necessary, and re-executed on the build of the application.

Of course, once this happens, go over the same process again with each new iteration expanding the application, improving on the previous build and fixing the faults and performance problems identified in the previous revision.

## Requirements driven model

What should have been clear, right from the very first part of this series, is that the requirements for the WebAuction application as defined by the various stakeholders in the project is the main driving force behind the development. The requirements define everything about the development, from a specification for the application to the performance requirements. All this is managed through Rational RequisitePro.

The requirements also form a vital part of the ongoing development model. The requirements become the linchpin of the process. You can link everything, from the original client requirements and specification to the models and code components to the requirements. Once you have this link, it's easy to connect other components such as test results to the components and ultimately back to the requirements. Through this same system you can also connect defect requests and changes and enhancements back to the original requirements.

Armed with this information you can refine and update the requirements to help specify the application and provide a guide and description of the application you are trying build. Ultimately, this leads to better application development. By having a centralized location for controlling the application and recording defects, enhancements, and other components, you can concentrate on the components of the application that need to be developed, updated and fixed.

---

## Section 6. Wrap up

### Summary

In this tutorial, the last of a five part series on developing software with the Rational Suite, the role of software testing and the Rational software that provides testing

facilities has been covered. The Rational PurifyPlus Suite provides a complete solution for determining internal faults such as memory leaks, performance issues and other programming issues.

For functional testing, Rational TestManager provides a comprehensive interface to a number of Rational tools, including Functional Tester. Through the Rational TestManager system you can define tests and distribute the tests through a variety of different platforms and environments to get an all-around test on as many environments as possible. Through the test cases you can also develop and define tests that cover the entire range of functionality in your application.

Of course, you can integrate the test applications with other components in the Rational suite. For example, you can automatically filter results into the Rational ClearQuest system so you can monitor and report on their progress as the tests are conducted and recorded.

This five part series has examined the crucial role Rational software plays in aiding application development. Rational software is a management platform for monitoring the entire development effort, from the inception of the requirements through to testing the components and ensuring that the developed application matches the requirements of the client.

# Resources

## Learn

- The previous tutorials of this series:
  - [Part 1, Translating requirements into an application model](#)
  - [Part 2, Integrating IBM Rational XDE Professional into WebSphere.](#)
  - [Part 3, Merging original requirements with changes](#)
  - [Part 4, Tracking changes during the application lifecycle](#)
- [Distributed functional testing: Testing across multiple platform architectures with Rational TestManager](#)
- [Releasing Better Software Faster with IBM Rational Purify and IBM Rational Quantify](#)
- [Getting development and test teams to use PurifyPlus together](#)
- [Using TestManager to report test coverage and progress](#)
- Find more information on the entire suite of [Rational products](#).
- The [developerWorks Rational site](#) is a portal for more information, tutorials and articles on the Rational environment.
- [New to Rational](#) can help you get started with Rational products.

## Get products and technologies

- Download a trial version of [RequisitePro](#).

## Discuss

- [Participate in the discussion forum for this content.](#)

# About the author

## Martin C. Brown

Martin C. Brown, a [Studio B](#) author, is a former IT Director with experience in cross-platform integration. A keen developer, he has produced dynamic sites for blue-chip customers, including HP and Oracle and is the Technical Director of Foodware.net. Now a freelance writer and consultant, MC, as he is better known, works closely with Microsoft as an SME, is the LAMP Technologies Editor for LinuxWorld magazine, a core member of the AnswerSquad.com team and has

written a number of books on topics as diverse as Microsoft Certification, iMacs and open source programming. Despite his best attempts, he remains a regular and voracious programmer on many platforms and numerous environments. MC can be contacted at [questions@mcsip.com](mailto:questions@mcsip.com), or through this [Web site](#).