Comparative Study of Rational and Mercury Functional Testing Tools

Kavitha Karthik Subbiah
Division of Computing Studies
Arizona State University, Polytechnic
kkarthi1@asu.edu

Abstract

The purpose of this project is to take an existing Java web based application and perform functional test on it by the IBM Rational Functional Tester and the Mercury Quick Test Professional. The objective of the project is to conduct a comparative study of these two automated testing tools based on criteria such as the effort involved with generating test scripts, ease of modification of the test script and the ability to accommodate new versions of the web applications. The fundamental goal is to analyze the features supported by these two functional testing tools that aid in minimizing the resources in script maintenance and increasing efficiency for script reuse.

1. Problem Statement

There are many challenges for testers of web applications, particularly the creation and maintenance of the test scripts. It is a fact that an application undergoes modifications and improvements over time. It is therefore crucial that automated testing tools build robust scripts. If the application developer changes properties of the object or adds new objects or deletes old objects from the application, the scripts might break and they need to be rerecorded.

It would be helpful to the testers if the automated testing tools are capable of building scripts that are flexible to changes in the application. Additionally it should notify the tester about discrepancies found in the application. If the application has undergone changes, the script should continue without intervention from the user and it should be possible for the testers to reuse the scripts on the new build of the application.

This project uses the Rational Functional Tester and the Mercury Quick Test Professional. It analyzes the features supported by the two testing tools how well they support script creation, maintenance and reuse.

2. Background and Related Work

Testing is important in the development of complex business processes and software testing is a labor intensive task. The choice (Manual Testing tool or Automated Testing tool) depends on a number of factors such as cost, time etc.

2.1. Automation for Regression testing

Automated Testing tools are highly reliable because it eliminates human errors. They also drastically speed up the testing process because more tests can be repeated with different test cases with in a short period of time. This is time consuming when done with a manual testing tool.

When the application undergoes significant changes over time, the number of tests also increases. More tests have to be carried on the new build to find bugs. This is true for a GUI based application especially when the application developer changes the user interface screen. In such a case, automated testing tools are highly suitable to repeatedly test the same set of operations which is time consuming when done with the manual testing tool.

2.2. Manual Testing tool

Automation is not always the best option. They are not suitable for a short term perspective because the initial investment in training is tremendous. Automated testing tools are not suitable for an unstable application. Such applications depend on real time data and the tester cannot predict the expected behavior. Automated testing tools also need technical expertise. If this is not available, it may be time consuming to run when compared to a manual testing tool.

This project is oriented towards regression testing. The Rational Functional Tester is used to test Java applications, VB.Net applications and html applications that run on Windows 2003 server or Windows XP professional. The Rational Functional Tester is the newest version of Rational Robot [1].

The Mercury Quick Test Professional is used to test Java applets, Java applications, VB.Net applications that run on Windows XP professional or Windows 2003 server [2]. It is the newest version of Win Runner. It is GUI based and a novice tester finds it easy to work on.

3. Methodologies and Architecture

The testing process consists of recording and playing back the script. This records the action performed by the user for the application under test. The initial step is to configure the test environment. During recording, various commands can be inserted to verify if the application works as intended. Finally, the script can be played back to replay the user actions. After the application developer updates the application, the script can be reused on the new build. The steps followed in the testing process are shown in Figure 1.

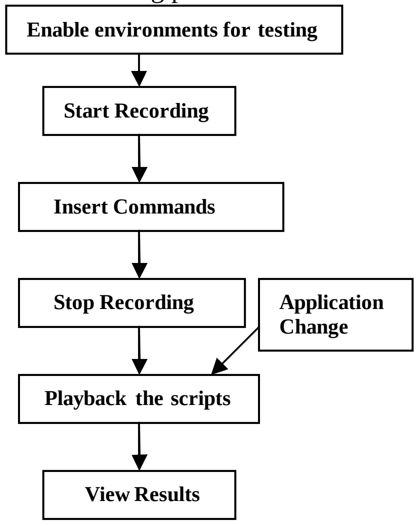


Figure 1: Steps for record/playback

3.1. Enable environments for testing

Initially the appropriate environment has to be configured to test the application. The browser should be enabled before recording to test HTML applications.

The Rational Functional Tester comes with Internet explorer as the default browser. However, there is an option for changing the default settings to add any other browser.

The application can be configured by loading the jar file for the Java application or the batch file for the windows application. JRE must be enabled for testing Java applications.

With the Mercury Quick test Professional, the browser is enabled before recording and a URL is provided for testing the html application. The tool supports Internet Explorer and Mozilla Firefox browsers.

3.2. Start Recording

The Recording monitor will generate statements for actions such as keystrokes and mouse clicks performed by the user.

The Rational Functional Tester has the recording monitor to record the user actions. The recording monitor contains buttons for inserting commands such as verification points and data driven commands.

The Mercury Quick Test Professional has a test pane. It generates statements for actions performed by the user.

3.3. Insert Commands

During recording, various commands can be inserted to verify if the application works as intended.

The verification points can be inserted to verify the data of the target objects and its properties (like maximum length of the text field, contents of the table, value of the input box etc). Different test cases can be generated.

The Rational Functional Tester has the data verification point to check the data of the target objects. The properties verification point can be used to verify the standard properties. During recording, data driven commands can be inserted to execute the application with different test cases [Appendix B].

The Mercury Quick professional has checkpoints to check if the application works as intended. The parameterizing test enables the application to perform the same operations with different test cases [Appendix B].

3.4. Stop Recording

After recording, the scripts are generated in the editor and the graphical objects in the application are stored in the object map/repository.

3.4.1. Scripts

The Rational Functional Tester generates scripts in the Java editor. Scripts are Java statements and they can be easily edited and executed using a standard Eclipse environment. [Appendix A].

The Mercury Quick Test Professional generates VB scripts. It has a test pane which contains two views. The Keyword View displays the graphical representation of the objects for the application under test [Appendix A]. The Expert view displays VB scripts.

3.4.2. Test Objects

The objects for the application under test have to be stored in the object map/object repository.

The Rational Functional Tester uses the test object map to represent graphical objects for the application under test [Appendix D].

The Mercury Quick Test Professional uses the object repository to represent graphical objects in the application [Appendix D].

3.5. Application Change

The application developer may introduce some significant changes in the application. If the same script is used on the new version, the script could break depending on the changes. However, automated testing tools have smart identification features which enable the script to accommodate many changes in the application.

The Rational Functional Tester has a smart recognition feature. It makes the scripts more flexible to changes. The Mercury Quick Test Professional has a smart identification mechanism which enables the script to be reused on the new build.

3.6. Playback the scripts

When the script is played back, it replays the user actions performed during recording. After playback, the results are generated in the test results window. It shows the pass/fail status for the application under test. It also displays the verification point results.

The Rational Functional Tester uses HTML/text log to display the results [Appendix C].

The Mercury Quick Test Professional has the Test result window which shows the pass/fail status [Appendix C].

4. Validation and Analysis

A Java web based application was used for testing. The application was hosted on the Tomcat web server. The web application consists of the following web pages:

a) Login.html: It prompts the user with the log in screen

b) Index.jsp: It allows the user to select the book and enter the quantity.

c) Purchase.jsp: It displays the user selected quantity and the book name in the table. It prompts the user for the credit card information.

d) Confirm.jsp: It confirms the purchase approval.

The browser (Internet Explorer) is enabled before testing. The URL for the application is given in the record settings. During recording, various commands are inserted to check the expected behavior of the application. Also commands are inserted to test the application with a variety of data.

4.1. Insert Commands

First the application is tested to see if it works as intended. The verification points can be inserted during/after recording to verify the state of the test objects. It can be used to confirm the state of the application across new builds.

4.1.1. Verification Points

The Rational functional tester supports two types of verification points [1]. Data verification point can be used to check the target object's data. Data verification can be used to check the data which can be in any form (table, list, tree, menu, state). Properties verification points can be used to verify properties like the contents in the table, value of the test object, name of the input text field, maximum length that a text box can accept etc. If the verification point fails, it opens the verification point comparator window showing the expected and the actual values. Failures are displayed in red. If the verification point passes, it opens the verification point editor. The following describes the results of my test. The Data Verification point was inserted in the "index.jsp" to verify the list by selecting multiple books. When the script was played back, the verification point failed because none of the books were selected. During playback, the books that were selected using the shift keys did not work. This can be eliminated by modifying the preference settings for the playback. So, the playback preference settings were modified to slow down the delays for the “Delay before key up” and “Delay before key down”. However, this was also throwing an exception. A radio button was added in the application to replace the multiple select capability and the user was allowed to select only one book. Finally, the verification point passed when one book was selected from the radio group. The Data verification point was inserted in the “purchase.jsp” to verify the table contents. The Verification point passed. The user selected quantity and book name were displayed in the verification point editor. The properties verification point was inserted in the “purchase.jsp” to test the maximum length of the data (card number) in the text box. The verification point passed. It accepted sixteen characters for the text field.

4.1.2. Checkpoints

The Mercury Quick Test Professional uses checkpoints to verify the expected behavior of the application. It supports different check points. Standard check points are used to verify the object’s standard properties such as value and name. Page check points are used to check number of links in the application and the time it takes to load the webpage. The text checkpoint is used to verify if the correct string is displayed correctly in the application. The table check point is used to verify the contents of the table [4].

The same test was performed with the Mercury Quick Test Professional tool. Multiple books were selected in the “index.jsp”. After recording, the standard checkpoint was inserted in the Book list to verify the selected value. During playback, the verification point passed. It displayed multiple books that were selected. Thus the books selected using the shift keys worked with the Mercury Quick Test Professional tool. The table check point was inserted to verify the table contents. It passed for the application. The text check point was inserted in the “purchase.jsp” to ascertain that it is for the correct customer. When the script was played back, the checkpoint passed for the application displaying the correct customer name.

Next, the application was executed with different test cases.

4.2. Execution with different test cases

The application was tested with multiple sets of input data. Instead of recording multiple tests to test multiple sets of input data, it is possible to make the script access the different sets of input data from the external source like data table, data pool. The verification point was inserted to verify if the application works properly with different sets of input data. With this feature, the data is not hard coded in the script. Since the data is separated from the script, it can be changed without affecting the script. New test cases can be added whenever it is needed.

4.2.1. Data Pool

The Rational Functional tester uses the “data driven test” for the generation of different test cases. The following steps were carried out to generate different test cases. During recording, the data driven commands were inserted in the “index.jsp” and “purchase.jsp”. Many records were added to enter a variety of input data for the books and quantity in the datapool. Each record represents one test case. The expected output values (Unit Cost and the Total Cost) were manually entered in the data pool. The verification point was inserted in the “purchase.jsp” to verify the table contents when different inputs were given for the quantity and the books. The verification point was created with a data pool reference instead of a literal value. So, when the script was played back, the script accessed one record from the test datapool. It supplied the input values to the variables in the script. Since the verification point referenced the data pool, it used the variable data as the baseline for comparison. Finally, the log displayed the verification point pass/fail status. The verification point passed for all the test cases.

4.2.2. Parameterizing the Tests

The Mercury Quick Test Professional uses the “Data table parameters” to generate different test cases. The test pane generates the script in the keyword view and the Expert view. The following steps were carried out to generate the different test cases. The value in the keyword view was configured to replace the constant value with the parameter. The value field accesses the input data from the data table. The values for the quantity and the book names were configured as the parameter values and different input values were entered in the datatable. In the Rational

Functional Tester, the output values for the Unit cost and the Total Cost were manually entered in the datapool. But with QTP there was an option for generating the output value. It was enabled in the data table. The output value for the Unit Cost and the Total Cost were generated automatically for the multiple books. The check point was inserted to check the contents of the table for the different books and the different quantities. The expected value of the test object "Book" was modified to match its value resulting from the parameter option. When the script was played back, the test result window generated the Runtime Data table. It showed all the input values given in the data table and the output values (Unit Cost and Total Cost) generated from the application .The check point passed for all the iterations of the application.

4.3. Scripts

After recording, the scripts were generated in the editor. Scripts are generated for the actions performed by the user for the application under test. Each step in the script represents a user action such as mouse clicks or keystrokes.

The script includes the method calls on the test objects, statements that perform the verification point and statements that create the data driven tests [Appendix A].

The Rational Functional Tester generated the scripts which are Java statements. All the graphical objects in the application are stored in the Test Object Map. Scripts contain references to those test objects.

The Mercury Quick Test Professional generated VB scripts. The keyword view in the test pane displays each step performed by the user in a table format [Appendix A]. It includes the items (test objects), operations (method calls on the test objects) and values for the application under test. It also generates the auto documentation for each step performed by the user. It was easy for novice testers to work with the keyword view. The expert view generates the VB scripts for the application under test. Once the testers gain technical expertise, they can write their own VB script to perform the test for the application [4].

4.4. Test Objects

The graphical objects in the application under test are stored in the test object map/Object repository.

The Rational Functional Tester uses the test object map to represent the graphical objects in the application. It stores the test objects in a hierarchical manner. The test objects contain the recognition and administrative properties. Each property has a weight ranging from 0-100 in its recognition properties. The Rational Functional Tester tool assigns the recognition score for each property it finds during the playback. The greater the recognition score, the less exact is the match between the recorded object and the object found during playback. If the score is within the threshold in the functional Tester preferences then it is not reported in the log. If the score is greater than the warning threshold then it is reported in the log.

The Mercury Quick Test Professional uses the Object repository to represent the graphical objects. Object repository stores all the test objects in a tree. When the target object was clicked in the tree, it shows the corresponding object's description properties as shown in Appendix D. It recognizes the objects using the objects description properties (name, html tag).If the QTP can not identify the objects using the descriptive properties objects then it uses the assistive properties.

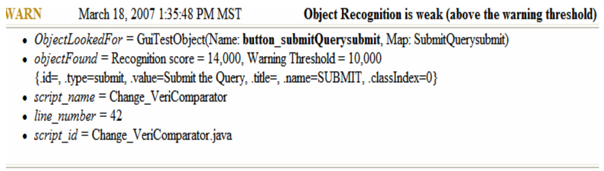
4.5. Script Reuse and Script Maintenance

The application was modified by changing some properties of the objects. New objects were added in the application and the same script was used on the new build.

4.5.1. Changing the Properties of the old objects

The dynamic content of the application was changed. The submit buttons in the "index.jsp" was changed from "Submit query" to "Submit the query" and purchase button in the "Purchase.jsp" was changed from "Purchase" to "purchase". The same script was used to test the application after these changes.

With the Rational Functional Tester, the script slowed near the "Submit the Query" and the "purchase" button. The script was looking for the old objects, but the objects had been changed. When the log was generated, it displayed the warning message as "Object Recognition is weak" as shown in Figure 3.



```
WARN    March 18, 2007 1:35:48 PM MST    Object Recognition is weak (above the warning threshold)
• ObjectLookedFor = GuiTestObject(Name: button_submitQuerysubmit, Map: SubmitQuerysubmit)
• objectFound = Recognition score = 14,000, Warning Threshold = 10,000
  {.id=, .type=submit, .value=Submit the Query, .title=, .name=SUBMIT, .classIndex=0}
• script_name = Change_VeriComparator
• line_number = 42
• script_id = Change_VeriComparator.java
```

Figure 3: Warning Message

The Rational Functional Tester assigned the recognition score for each property of the test objects it finds during playback. The value field in the recognition properties of the submit buttons were changed. Since one property had been changed for the submit buttons (Submit Query and the Purchase), it generated a warning message in the log. The properties verification point was inserted in the submit button and the value field was selected. The verification point failed for the application. There were three options to fix this:

4.5.1.1. Using the Verification Point comparator

The verification point failed and the verification point comparator was opened. It displayed the actual value and the expected value. The verification point comparator was updated to load the difference by editing the actual value. When the script was played back, it passed with no warnings.

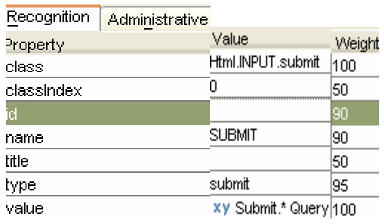
4.5.1.2. Using the Test object Map

The application containing the new object was selected to run. The new object (Submit the Query) was chosen in the "index.jsp". The new object was inserted in the test object map. The old object (button_submitQuery submit) was selected and it was dragged into the new object (button_submittheQuery) using the "Unify Test Object Wizard". Both the old and the new object properties were listed. When the script was played back, it passed with no warnings.

4.5.1.3. Using Regular Expressions

Regular expressions can be used to allow the script to run on more than one version of the application.

The test objects (button_submittheQuery) and the (button_Purchsesubmit) was selected .The value field in the recognition property was changed from "Submit the Query" to "Submit.* Query" as shown in Figure: 4 and "Purchase" to "[pP] urchase", so that it can be run on several versions of the application.



Property	Value	Weight
class	Html.INPUT.submit	100
classIndex	0	50
id		90
name	SUBMIT	90
title		50
type	submit	95
value	xy Submit.* Query	100

Figure 4: Regular Expressions

The same test was carried with the Mercury Quick Test Professional. The submit button was changed from "Submit Query" to "Submit the Query" and the purchase button was changed from "Purchase" to "purchase". When the script was played back the test result window generated the warning message for the "Submit the Query" button .But the change "Purchase" to "purchase" did not generate the warning message as with the Rational Functional Tester. The test objects are case insensitive in the Mercury Quick Test Professional tool.

In the object Repository, the "submit Query" object was selected and its name was changed in its description properties to "Submit the Query". When the script was played back, the test result passed with no warnings.

4.5.2. Adding a new Object

A reset button was added in the "index.jsp". A new object must be added to the test object map/Object Repository and to the script if a decision to add has been made. With Rational Functional Tester, the new object must be added to the test object map. A "Reset" button was selected from the "index.jsp" in the new application and it was added to the test object map. To add the new object to the script, the new object and the corresponding method calls were inserted in the script. After adding the new object in the script and in the test object map, the same script was reused.

With Mercury Quick Test Professional, a new object must be added to the Object Repository and to the script. Initially, a reset button was added in the object repository and its description properties were listed. The method calls for the new object were inserted in the keyword view of the test pane and the same script was reused.

4.5.3. Changing the User Interface Screen

The user interface screen was changed in the "index.jsp" by changing the location of the "Name", "Quantity" and the books were changed and tested. The same script was used and the script did not break .The two tools does not depend on the screen coordinates to find the test objects, it uses the recognition and the mandatory properties to ascertain the test objects.

Therefore the script can be reused even when the user interface screen changed.

4.5.4. Calling another script

Tests can be divided into multiple actions. Sometimes identical activities are to be repeated, instead of recording multiple times, the script can call another script to avoid duplication of tests.

The Rational Functional Tester has script support commands to call one script from another script.

The Mercury Quick Test Professional permits to divide the tests into multiple actions. Call to an existing action may be performed with in a script."Login.html" page was introduced in the application and tested with both the tools. The old script was called to repeat the identical activities. It requires technical expertise to work with Mercury Quick Test Professional to call the existing action in the same script.

As a novice tester, I performed the testing on a Java web based application with the tools for a period of three months and observed the features supported by these tools in the script creation, maintenance and reuse.

The major strengths and weaknesses of the tools are highlighted as follows. Mercury Quick Test Professional is easy for a novice tester to work with. It is GUI based. With the Rational Functional Tester, the multiple select feature using shift keys did not work. The output values have to be manually entered for the data pool feature of the Rational Functional Tester. With Mercury Quick Test Professional the output values are automatically generated at runtime. Rational Functional Tester is cheaper than Mercury Quick Test Professional. The results of all the tests executed with both the tools are rated as shown below and given in the table.

★★★★★	Excellent
★★★★	Very Good
★★★	Good
★★	Satisfied
★	Unsatisfied

Results

S.NO	Criteria	Rational Functional Tester	Mercury Quick Test Professional	Reason
1.	Generation of Scripts	★★★★★	★★★	The Rational Functional tester is capable of generating VB scripts and Java scripts (Java statements). It is Eclipse based. The Mercury Quick Test Professional generates only VB scripts.
2.	Scripts	★★	★★★★★	Mercury Quick Test Professional is GUI based. Auto documentation is created for each step performed by the user (in the table) in the keyword view and a novice tester finds the tool easy to work with. The Rational Functional Tester requires some programming experience.
3.	Playback of the scripts	★	★★★★★	User actions performed during recording are replayed during playback. Multiple values selected using the shift keys did not work with the Rational Functional Tester. However, multiple select capabilities worked with Mercury Quick Test Professional.
4.	Feature to generate different test cases	★★★	★★★★★	The Rational Functional Tester has data driven commands to generate different test cases. The Mercury Quick Test Professional uses “parameterizing the tests” to generate test cases. However, the output values have to be manually entered with the Rational Functional Tester. With Mercury Quick Test Professional the output values are generated automatically.
5.	Cost	★★★★★	★★	Rational Functional Tester is cheaper than Mercury Quick Test Professional.
6.	Accommodation of new versions of applications	★★★★★	★★★	The two tools have features that allow one script to call another script and identical activities are not repeated. This process is easily accomplished with the Rational Functional Tester compared to the Mercury Quick Test Professional which requires technical expertise.
7.	Script Reuse	★★★★★	★★★★★	The tools have smart recognition features which permit reuse of the script on a new build.
8.	Test Results	★★★	★★★★★	The test results are displayed in the html/text log for the Rational Functional Tester. But the Mercury Quick Test Professional displayed the results in the form of a tree in the test result window. When the target object was selected, the tool gives a visual representation of the snapshot (captured during recording) in the screen recorder.

5. Conclusion:

The project successfully evaluated the two tools and verified the expected behavior of the application. The verification point failed with the Rational Functional Tester when multiple books were selected. Hence, the application had to be modified and the radio button was added to replace the multiple select capabilities. However, the application worked with the Mercury Quick Test Professional. The application was tested with different test cases using the data table feature of Mercury Quick Test Professional and the datapool feature of the Rational Functional Tester. The changes were introduced in the application and the same script was reused to run on the new build and successfully tested.

Appendix A shows the Rational Functional Tester IDE and the Mercury Quick Test Professional IDE.

Appendix B shows different test cases generated by datapool feature of Rational Functional Tester and different test cases generated by Data table feature of Mercury Quick Test Professional.

Appendix C shows the html log generated by Rational Functional Tester and the test results window generated by Mercury Quick Test Professional.

Appendix D shows the Test object Map (represents the test object) of Rational Functional Tester and the Object Repository of Mercury Quick Test Professional.

6. Acknowledgements

I would like to thank my chair Dr.Harry Koehnemann for his continuous support and guidance. This was instrumental to the success of this project. I would like to also thank my committee members Dr.Kevin Gary and Dr.Bruce Millard for their valuable feedback.

7. References

[1] IBM Rational Software, “Essentials of IBM Rational Functional Tester, Java Scripting, v.6.1”, IBM Corporation, Jan 2005.

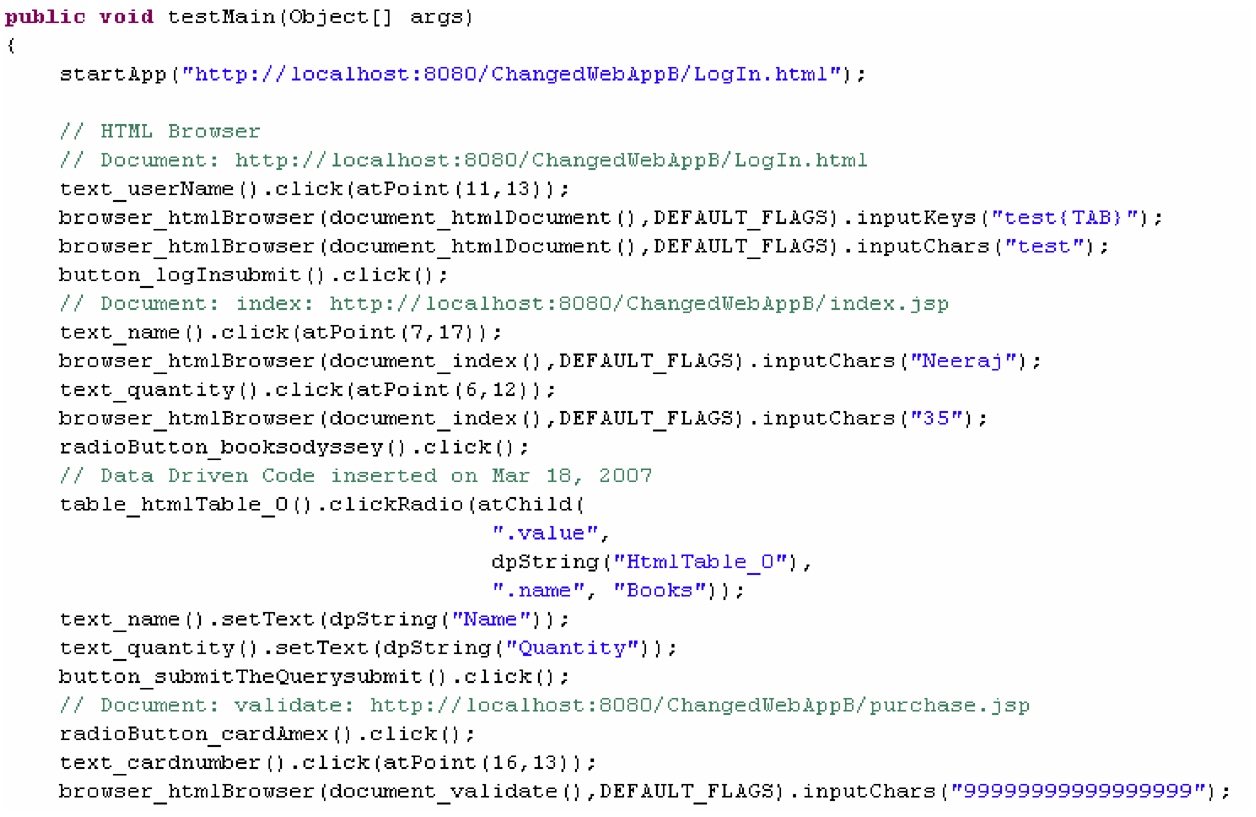
[2]Hewlett Packard Development Company, L.P. “Mercury QuickTestProfessional”.(2007).http://www.mercury.com/us/products/qualitycenter/functional-testing/quicktest-professional/ (11 Mar.2007).

[3]Marty hall and Mary Brown, “Core Servlets and Java Server pages”, Sun Microsystems Press/Prentice Hall PTR

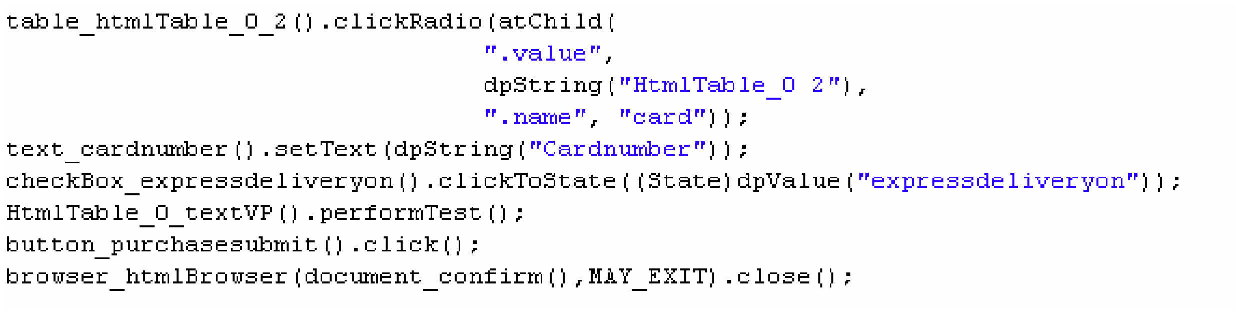
[4]IBM Corporation. “Rational Functional Tester”. http://www-306.ibm.com/software/awdtools/tester/functional/index.html (25 Jan.2007).
.

APPENDIX A:

Rational Functional Tester IDE (Scripts are Java statements)

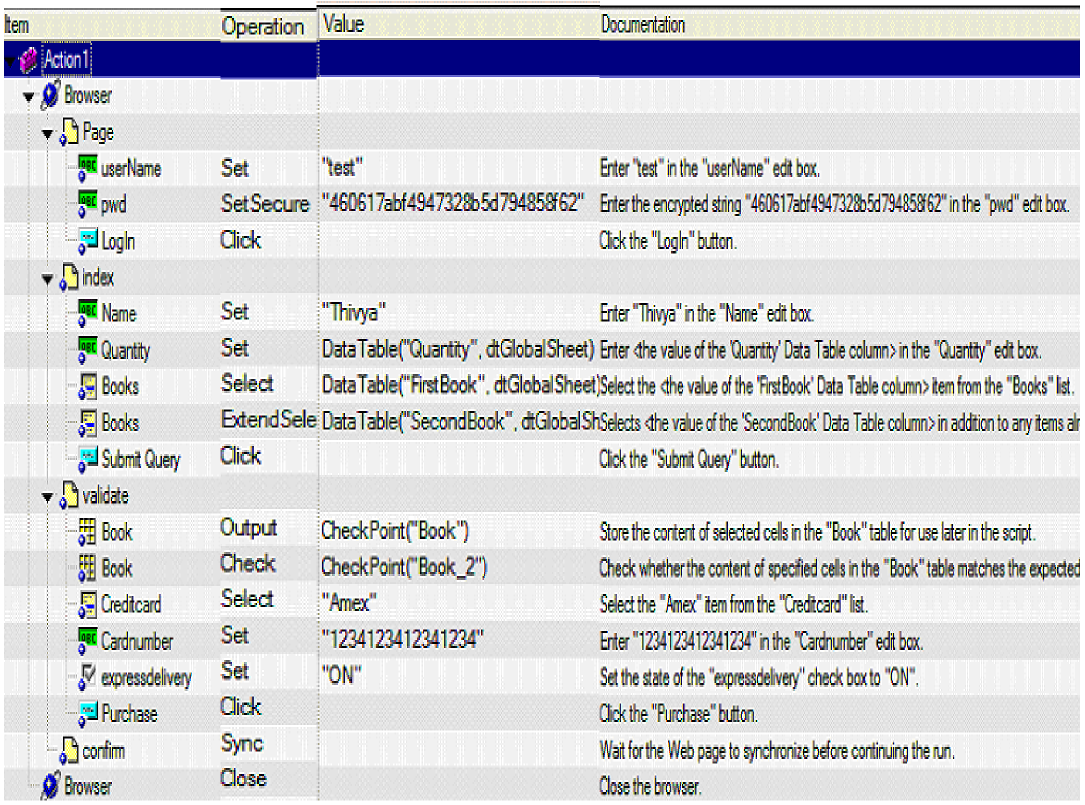


```java
public void testMain(Object[] args)
{
    startApp("http://localhost:8080/ChangedWebAppB/LogIn.html");

    // HTML Browser
    // Document: http://localhost:8080/ChangedWebAppB/LogIn.html
    text_userName().click(atPoint(11,13));
    browser_htmlBrowser(document_htmlDocument(),DEFAULT_FLAGS).inputKeys("test{TAB}");
    browser_htmlBrowser(document_htmlDocument(),DEFAULT_FLAGS).inputChars("test");
    button_logInsubmit().click();
    // Document: index: http://localhost:8080/ChangedWebAppB/index.jsp
    text_name().click(atPoint(7,17));
    browser_htmlBrowser(document_index(),DEFAULT_FLAGS).inputChars("Neeraj");
    text_quantity().click(atPoint(6,12));
    browser_htmlBrowser(document_index(),DEFAULT_FLAGS).inputChars("35");
    radioButton_booksodyssey().click();
    // Data Driven Code inserted on Mar 18, 2007
    table_htmlTable_0().clickRadio(atChild(
                                ".value",
                                dpString("HtmlTable_0"),
                                ".name", "Books"));
    text_name().setText(dpString("Name"));
    text_quantity().setText(dpString("Quantity"));
    button_submitTheQuerysubmit().click();
    // Document: validate: http://localhost:8080/ChangedWebAppB/purchase.jsp
    radioButton_cardAmex().click();
    text_cardnumber().click(atPoint(16,13));
    browser_htmlBrowser(document_validate(),DEFAULT_FLAGS).inputChars("99999999999999999");
```



```java
    table_htmlTable_0_2().clickRadio(atChild(
                                ".value",
                                dpString("HtmlTable_0 2"),
                                ".name", "card"));
    text_cardnumber().setText(dpString("Cardnumber"));
    checkBox_expressdeliveryon().clickToState((State)dpValue("expressdeliveryon"));
    HtmlTable_0_textVP().performTest();
    button_purchasesubmit().click();
    browser_htmlBrowser(document_confirm(),MAY_EXIT).close();

}
```

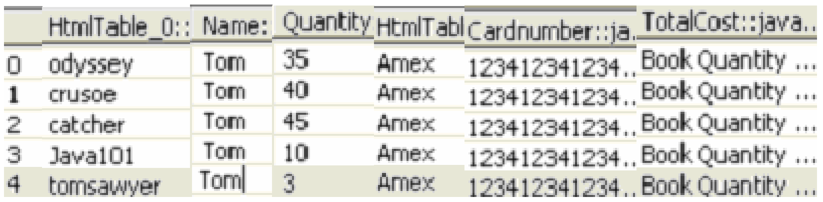
Mercury Quick Test Professional IDE (Keyword View)



Item	Operation	Value	Documentation
Action1			
Browser			
Page			
userName	Set	"test"	Enter "test" in the "userName" edit box.
pwd	SetSecure	"460617abf4947328b5d794858f62"	Enter the encrypted string "460617abf4947328b5d794858f62" in the "pwd" edit box.
LogIn	Click		Click the "LogIn" button.
index			
Name	Set	"Thivya"	Enter "Thivya" in the "Name" edit box.
Quantity	Set	DataTable("Quantity", dtGlobalSheet)	Enter <the value of the 'Quantity' Data Table column> in the "Quantity" edit box.
Books	Select	DataTable("FirstBook", dtGlobalSheet)	Select the <the value of the 'FirstBook' Data Table column> item from the "Books" list.
Books	ExtendSele	DataTable("SecondBook", dtGlobalSh	Selects <the value of the 'SecondBook' Data Table column> in addition to any items alr
Submit Query	Click		Click the "Submit Query" button.
validate			
Book	Output	CheckPoint("Book")	Store the content of selected cells in the "Book" table for use later in the script.
Book	Check	CheckPoint("Book_2")	Check whether the content of specified cells in the "Book" table matches the expected
Creditcard	Select	"Amex"	Select the "Amex" item from the "Creditcard" list.
Cardnumber	Set	"1234123412341234"	Enter "1234123412341234" in the "Cardnumber" edit box.
expressdelivery	Set	"ON"	Set the state of the "expressdelivery" check box to "ON".
Purchase	Click		Click the "Purchase" button.
confirm	Sync		Wait for the Web page to synchronize before continuing the run.
Browser	Close		Close the browser.

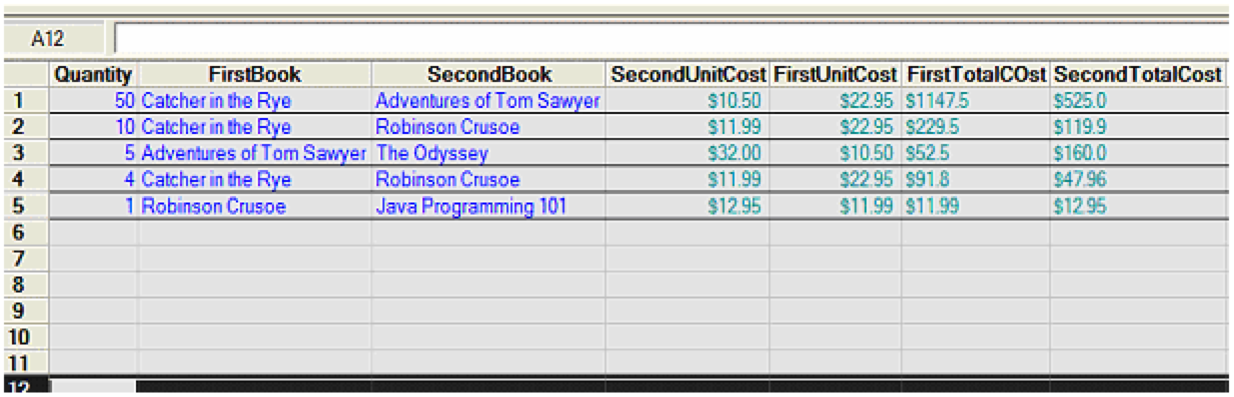
APPENDIX B:

Rational Functional Tester (Data Pool)



	HtmlTable_0::	Name:	Quantity	HtmlTabl	Cardnumber::ja.	TotalCost::java..
0	odyssey	Tom	35	Amex	123412341234..	Book Quantity ...
1	crusoe	Tom	40	Amex	123412341234..	Book Quantity ...
2	catcher	Tom	45	Amex	123412341234..	Book Quantity ...
3	Java101	Tom	10	Amex	123412341234..	Book Quantity ...
4	tomsawyer	Tom	3	Amex	123412341234..	Book Quantity ...

Mercury Quick Test Professional (Parameterizing the tests)

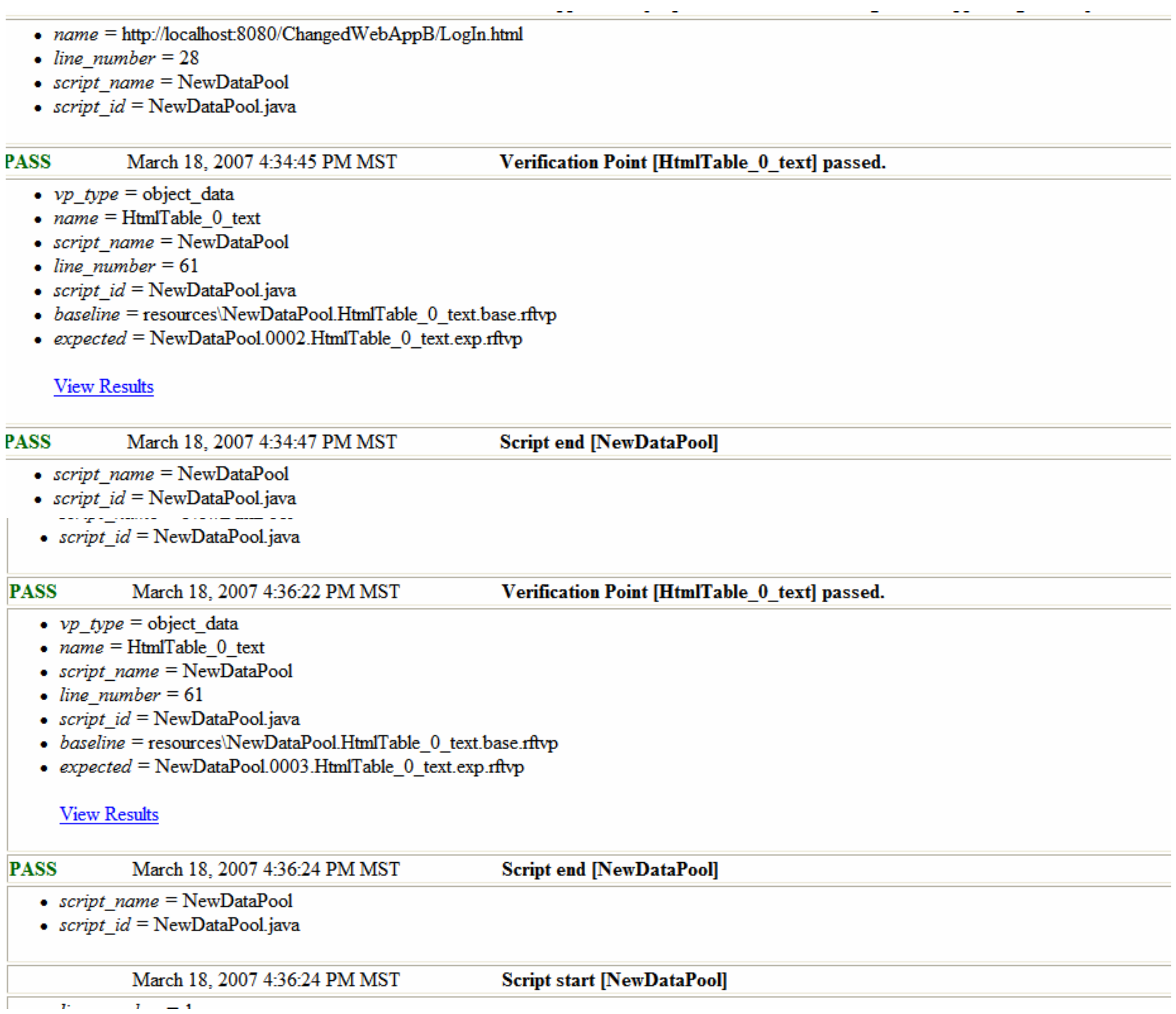


A12

	Quantity	FirstBook	SecondBook	SecondUnitCost	FirstUnitCost	FirstTotalCOst	SecondTotalCost
1	50	Catcher in the Rye	Adventures of Tom Sawyer	$10.50	$22.95	$1147.5	$525.0
2	10	Catcher in the Rye	Robinson Crusoe	$11.99	$22.95	$229.5	$119.9
3	5	Adventures of Tom Sawyer	The Odyssey	$32.00	$10.50	$52.5	$160.0
4	4	Catcher in the Rye	Robinson Crusoe	$11.99	$22.95	$91.8	$47.96
5	1	Robinson Crusoe	Java Programming 101	$12.95	$11.99	$11.99	$12.95
6							
7							
8							
9							
10							
11							

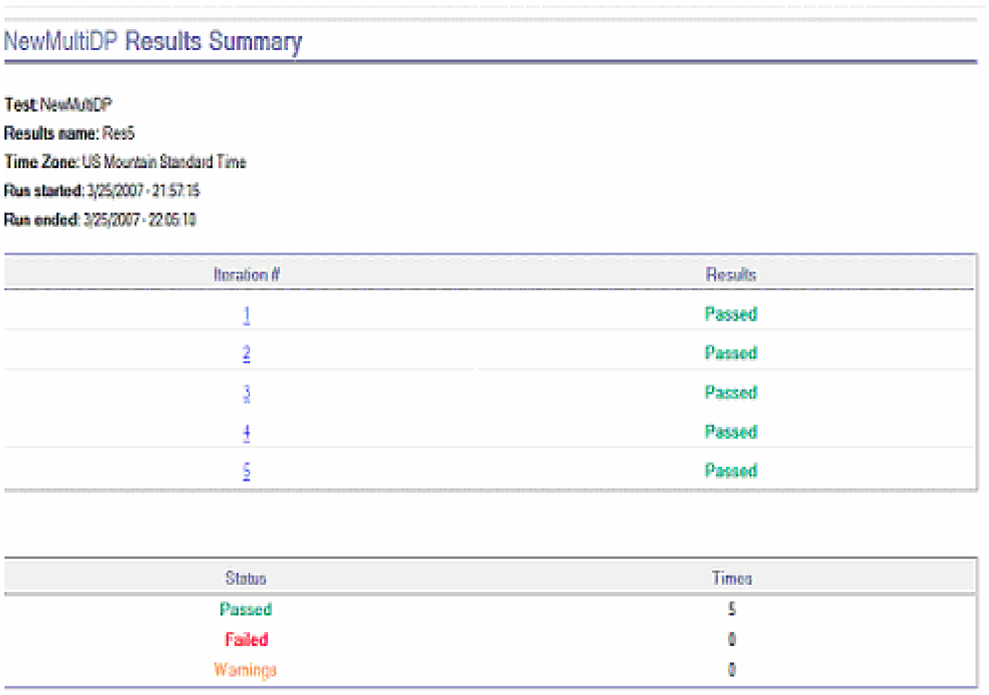
APPENDIX C:

Test Results for the Rational Functional Tester (HTML Log)



```
• name = http://localhost:8080/ChangedWebAppB/LogIn.html
• line_number = 28
• script_name = NewDataPool
• script_id = NewDataPool.java

PASS        March 18, 2007 4:34:45 PM MST        Verification Point [HtmlTable_0_text] passed.
• vp_type = object_data
• name = HtmlTable_0_text
• script_name = NewDataPool
• line_number = 61
• script_id = NewDataPool.java
• baseline = resources\NewDataPool.HtmlTable_0_text.base.rftvp
• expected = NewDataPool.0002.HtmlTable_0_text.exp.rftvp

  View Results

PASS        March 18, 2007 4:34:47 PM MST        Script end [NewDataPool]
• script_name = NewDataPool
• script_id = NewDataPool.java
• script_id = NewDataPool.java

PASS        March 18, 2007 4:36:22 PM MST        Verification Point [HtmlTable_0_text] passed.
• vp_type = object_data
• name = HtmlTable_0_text
• script_name = NewDataPool
• line_number = 61
• script_id = NewDataPool.java
• baseline = resources\NewDataPool.HtmlTable_0_text.base.rftvp
• expected = NewDataPool.0003.HtmlTable_0_text.exp.rftvp

  View Results

PASS        March 18, 2007 4:36:24 PM MST        Script end [NewDataPool]
• script_name = NewDataPool
• script_id = NewDataPool.java

            March 18, 2007 4:36:24 PM MST        Script start [NewDataPool]
```

Test Result Window for Quick Test Professional



NewMultiDP Results Summary

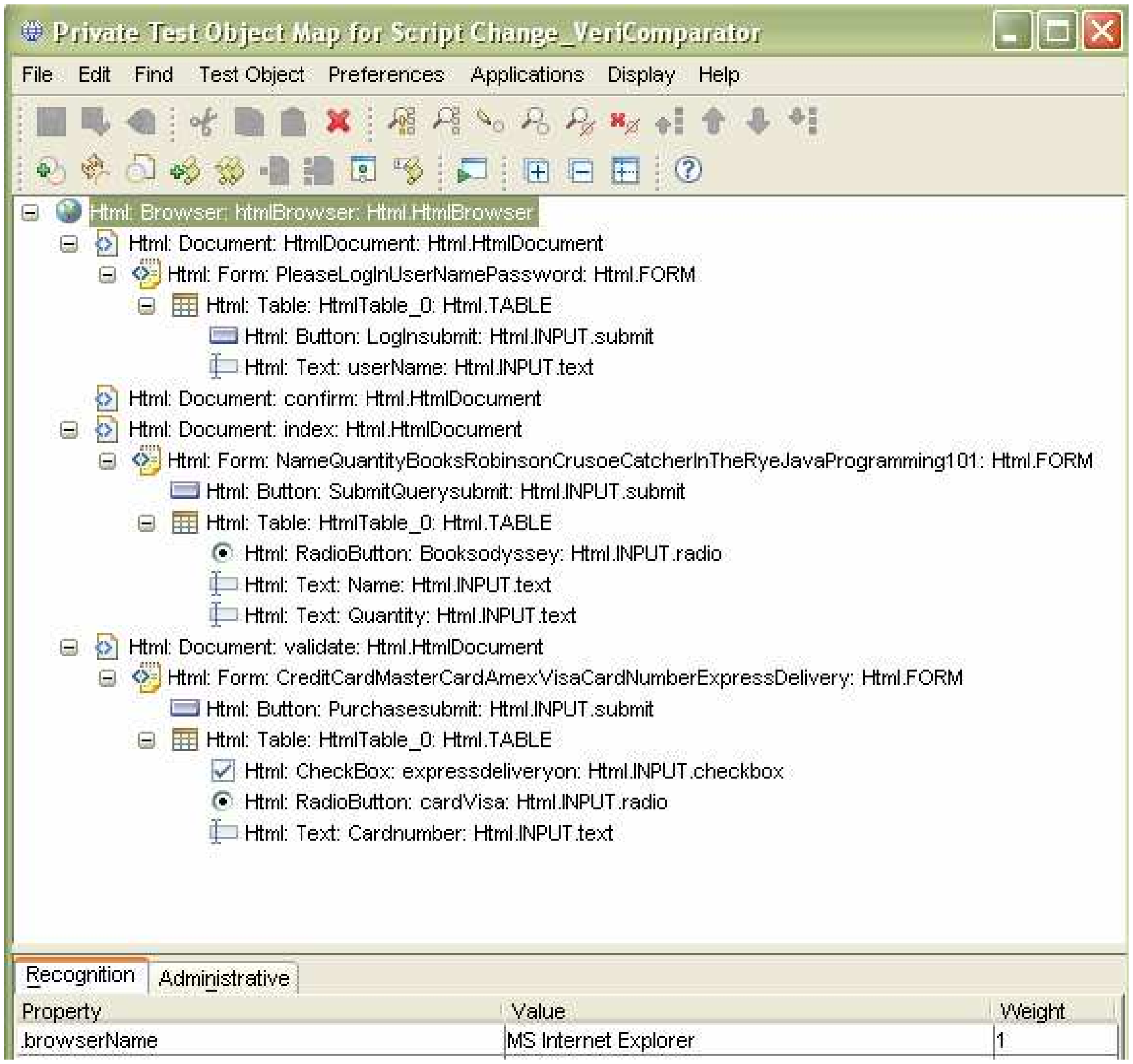
Test: NewMultiDP
Results name: Res5
Time Zone: US Mountain Standard Time
Run started: 3/25/2007 - 21:57:15
Run ended: 3/25/2007 - 22:05:10

Iteration #	Results
1	Passed
2	Passed
3	Passed
4	Passed
5	Passed

Status	Times
Passed	5
Failed	0
Warnings	0

Appendix D:

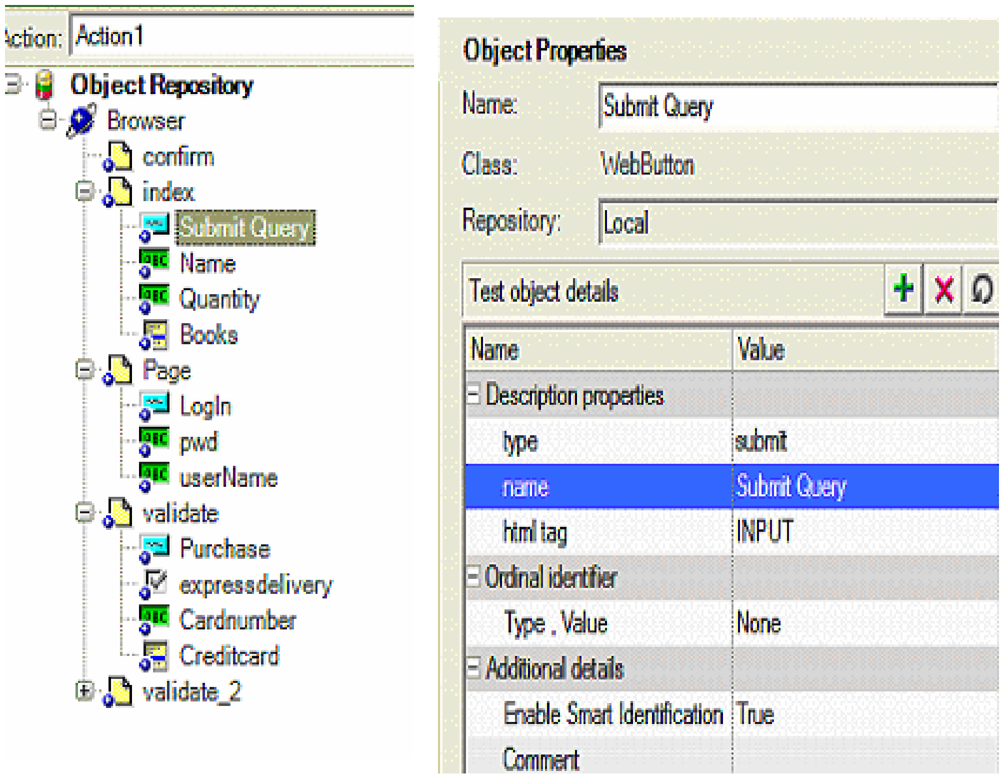
Rational Functional Tester (Test Object Map to represent the graphical objects in the application)



```
Html: Browser: htmlBrowser: Html.HtmlBrowser
  Html: Document: HtmlDocument: Html.HtmlDocument
    Html: Form: PleaseLogInUserNamePassword: Html.FORM
      Html: Table: HtmlTable_0: Html.TABLE
        Html: Button: LogInsubmit: Html.INPUT.submit
        Html: Text: userName: Html.INPUT.text
  Html: Document: confirm: Html.HtmlDocument
  Html: Document: index: Html.HtmlDocument
    Html: Form: NameQuantityBooksRobinsonCrusoeCatcherInTheRyeJavaProgramming101: Html.FORM
      Html: Button: SubmitQuerysubmit: Html.INPUT.submit
      Html: Table: HtmlTable_0: Html.TABLE
        Html: RadioButton: Booksodyssey: Html.INPUT.radio
        Html: Text: Name: Html.INPUT.text
        Html: Text: Quantity: Html.INPUT.text
  Html: Document: validate: Html.HtmlDocument
    Html: Form: CreditCardMasterCardAmexVisaCardNumberExpressDelivery: Html.FORM
      Html: Button: Purchasesubmit: Html.INPUT.submit
      Html: Table: HtmlTable_0: Html.TABLE
        Html: CheckBox: expressdeliveryon: Html.INPUT.checkbox
        Html: RadioButton: cardVisa: Html.INPUT.radio
        Html: Text: Cardnumber: Html.INPUT.text
```

Property	Value	Weight
.browserName	MS Internet Explorer	1

Mercury Quick Test Professional
(Object Repository to represent the graphical objects for the application)



Action: Action1

Name: Submit Query

Class: WebButton

Repository: Local

Name	Value
Description properties	
type	submit
name	Submit Query
html tag	INPUT
Ordinal identifier	
Type , Value	None
Additional details	
Enable Smart Identification	True
Comment	