



WPI  **BARCLAYS**

Barclays Core Prime Brokerage QA Automated GUI Testing Migration Analysis

Project # KMS 1603

A Major Qualifying Project Report

Submitted to the faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements of the

Degree of Bachelor of Science

Submitted on 17th December 2016

In Cooperation with:

Barclays PLC

Submitted by:

Didi Dai, Industrial Engineering and Management Information System

Jonathan Ho Wu, Electrical and Computer Engineering

Submitted to:

On-site Liaison:

Michael Irving

Bharat Rao

Karlene Wright

Project Advisors:

Professor Michael J. Ciaraldi, Department of Computer Science

Professor Xinming Huang, Department of Electrical and Computer Engineering

Professor Renata Konrad, Foisie School of Business

Professor Kevin M. Sweeney, Foisie School of Business

Abstract

Sponsored by the Core Prime Brokerage (PB) Quality Assurance team within the Investment Bank division of Barclays PLC, this Major Qualifying Project (MQP) report centered on web applications. The goal of the project was to first identify free, open-source tools that would serve as alternatives to the current Hewlett-Packard's Quick Test Professional (QTP)/Unified Functional Testing (UFT) solution, while preserving the existing core functions written in VBScript scripting language. After performing thorough exploratory analysis and research, the MQP project team identified Selenium WebDriver as the best tool for Barclays Core PB's GUI automation testing purposes. Our recommendations not only aim to help eliminate the UFT license contention and execution issues, but also aim to save the annual licensing cost spent by the Barclays Core PB team and reduce the time spent troubleshooting any GUI automation testing issues.

Acknowledgments

The success of our Major Qualifying Project (MQP) would not have been possible without the help of many individuals inside and outside of Barclays. We would like to start off by thanking Barclays PLC for sponsoring us and enabling us to have this invaluable experience. In this regard, we would like to especially thank our direct boss and mentor, Karlene Wright, for her exhaustive support, guidance, and technological expertise throughout our tenure at Barclays. Her oversight and dedication were key to the completion and success of this project. Additionally, we would like to thank our other bosses, Michael Irving and Bharat Rao, for their support and for making this project a possibility. Furthermore, we would also like to express our gratitude towards our other team members in Kiev, Ukraine, who helped us understand the entire automated GUI testing process more thoroughly.

We would also like to thank our school, Worcester Polytechnic Institute (WPI) and the WPI Interdisciplinary and Global Studies Division (IGSD) for all their logistical support and planning. Finally, we would like to thank our advisors, Professor Kevin Sweeney, Professor Michael Ciaraldi, Professor Xinming Huang, and Professor Renata Konrad for all their continuous support and guidance throughout the project.

The project would not have reached this point if it was not for their countless recommendations and guidance in solidifying our project. We are very proud to have formed part of the Wall Street Project Center and to have worked in such with such an amazing and talented group of people within a renowned multinational corporation such as Barclays.

Authorship

Content	Primary Author
Title Page	Didi Dai
Abstract Acknowledgments Authorship	Jonathan Ho Wu
Executive Summary	Didi Dai
Chapter 1: Introduction	Didi Dai
Chapter 2: Background	Jonathan Ho Wu
Chapter 3: Methodology	Didi Dai
Chapter 4: Two Feasible Automation Tools	Jonathan Ho Wu
Chapter 5: Recommendations and Conclusions	Didi Dai, Jonathan Ho Wu
Chapter 6: Reflection	Didi Dai, Jonathan Ho Wu
Works Cited	Didi Dai, Jonathan Ho Wu
Appendix	Jonathan Ho Wu
All sections were edited as a team, with equal contributions made by every member.	

Executive Summary

Barclays, a pioneer in the Fin-Tech industry, drives technology implementations across the core prime brokerage in the United States and the rest of the world. This MQP project worked with the Core Prime Brokerage team in the IT department of Barclays America. Routinely, Barclays IT department releases new and updated web apps to their customers. Before releasing the apps, the quality assurance team plays an important role in testing and inspecting the apps. However, the current testing software presents a big financial investment for Barclays and faces licensing issues.

The main goal of this MQP project was to identify for a no-cost replacement for the current testing software, Hewlett-Packard's Quick Test Professional (QTP)/Unified Functional Testing (UFT). To make the new tool applicable in this complex environment, the replacement needed to satisfy three constraints: (i) open-source; (ii) VBScript scripting language support, and (iii) GUI Web Application Testing Support. The MQP team researched possible alternatives in the market. By identifying the strengths and weakness of each alternative, we conducted a comparison analysis to support our recommendations. Identifying Selenium WebDriver as the tool which best satisfies Barclays operating constraints, we researched how to embed the new tool into Barclays' automation GUI Web application testing framework and began the implementation. In doing so, Barclays addresses the existing QTP licenses issues and significantly reduces its investment in testing software. The MQP team identified a method to get the object from the website by using JAVA and linking JAVA with VBScript. In doing so, existing VBScript testing cases can be preserved and the automation testing migration from QTP to Selenium WebDriver can be realized. For easing the process of getting an object, we also researched on

the object repository and found three ways to store objects, in the condition of using Selenium WebDriver.

In addition, by learning the VBScript scripting language, we found the in-house development tool could also be a replacement of QTP. The in-house development tool is going to be developed by using Microsoft Internet Explorer Object. This tool can be accomplished by writing VBScript scripting language only. Tool implementation and object storage were also investigated.

We recommend in the short run, Barclays should use Internet Explore Object to create the in-house development tool, as it is simpler and more consistent since it uses a scripting language that the employees are already familiar with. In the long run, we recommend implementing Selenium WebDriver, as it is easier to debug and it is currently in draft form to become a W3C (World Wide Web Consortium) Web Standard.

Table of Contents

Abstract	i
Acknowledgments.....	ii
Authorship	iii
Executive Summary	iv
Table of Figures	viii
List of Tables	viii
Chapter 1: Introduction	1
Chapter 2: Background	2
2.1 Current GUI Web Application Automation Testing Framework.....	3
2.1.1 <i>Quality Center</i>	3
2.1.2 <i>QTP/UFT</i>	4
2.1.3 <i>VBScript Scripting Language</i>	5
2.1.4 <i>OR Translator</i>	5
2.1.5 <i>QTP Object Repository</i>	5
2.2 Problem Statement.....	7
Chapter 3: Methodology.....	9
3.1 Implementation Criteria.....	9
3.2 Comparing Existing Automation Tools.....	9
3.3 Rationale for Eliminating Alternatives	12
3.3.1 <i>Microsoft Coded UI Test</i>	12
3.3.2 <i>Test Complete</i>	13
3.3.3 <i>Rational Functional</i>	13
3.3.4 <i>iMacros</i>	13
3.3.5 <i>Oracle Application Testing Suite</i>	13
3.3.6 <i>Eggplant Functional</i>	13
3.3.7 <i>Sahi</i>	14
3.3.8 <i>Maveryx</i>	14
Chapter 4: Two Feasible Automation Tools.....	15
4.1 Alternative #1: Selenium WebDriver	15
4.1.1 <i>Background</i>	15
4.1.2 <i>Browser Compatibility</i>	16
4.1.3 <i>Calling Java by Using VBScript</i>	17
4.1.4 <i>Instantiating a Specific Browser</i>	18
4.1.5 <i>Other Ways to Get the Elements by Using Selenium</i>	18
4.1.6 <i>How to Implement an Object Repository in Selenium?</i>	19
4.2 Alternative #2: Using Internet Explorer Object	26

4.2.1 Background	26
4.2.2 How Does It Work?	26
4.2.3 How to Implement the Object Repository in the In-house Development Tool?	27
4.3 Comparing the Two Feasible Solutions	29
4.3.1 Selenium WebDriver	29
4.3.2 Internet Explorer Object	30
Chapter 5: Recommendations and Conclusions	32
5.1 Conclusion	32
5.2 Recommendations	33
Chapter 6: Reflecting on our Experience at Barclays	35
6.1 Discussion of Design in the Context of the Project	35
6.2 Discussion of Constraints Considered in the Design	35
6.3 Discussion of the Need for Life-long Learning	36
Works Cited	37
Appendix 1	39
Appendix 2	40
Appendix 3	41
Appendix 4	43
Appendix 5	44
Appendix 6	45
Appendix 7	48
Appendix 8	50
Appendix 9	52

Table of Figures

FIGURE 1: CURRENT GUI TESTING FRAMEWORK WORKFLOW	3
FIGURE 2: QUALITY CENTER	4
FIGURE 3: UFT/QTP	5
FIGURE 4: WEIGHT BAR CHART FOR THE AVAILABLE AUTOMATION TESTING TOOLS	12
FIGURE 5: SELENIUM WEBDRIVER WORKING PRINCIPLE	16
FIGURE 6: SAMPLE PROPERTY FILE	22
FIGURE 7: OBJECT MAP	23
FIGURE 8: THE PRINCIPLE OF THE PAGE OBJECT MODEL	24
FIGURE 9: EXAMPLE OF A PAGE OBJECT MODEL IN ACTION	25
FIGURE 10: CHILD ELEMENTS IN THE PRODUCT	26
FIGURE 11: OBJECT REPOSITORY MANAGER	28
FIGURE 12: EXPORTING THE XML FILE	28

List of Tables

TABLE 1: POTENTIAL SOFTWARE COMPARISON TABLE	10
TABLE 2: BEST OPTION AMONG ALL SOFTWARE TOOLS	11

Chapter 1: Introduction

Our goal is to help develop an alternate solution that would allow the Core Prime Brokerage Quality Assurance team to run their existing GUI automated tests without using HP Enterprise's Unified Functional Testing (UFT) software too.

We first reviewed the current Quality Assurance framework and then identified two solutions capable of replacing the UFT's execution piece of code, while maintaining the existing core functions written in VBScript scripting language.

By the end of our project, we provided Barclays with a deliverable that detailed the recommendations on which tool is the most effective and how to start implementing the most suitable tool while considering the future testing of Barclays Core Prime Brokerage GUI Web applications.

Chapter 2: Background

Barclays Bank PLC, founded in 1690, is one of the largest multinational banking and financial services companies in the United Kingdom, and its headquarter is in London (Delaware, 2016). Barclays provides different financial services in retail, wholesale, investment banking, wealth management, mortgage lending and credit cards, which all operate in multiple countries across Europe, the Americas, Asia, and Africa. There are four core business sectors of Barclays: Personal & Corporate (Personal Banking, Corporate Banking, Wealth & Investment Management), Barclaycard, Investment Banking and Africa. Barclays has a primary listing on the London Stock Exchange and a secondary listing on the New York Stock Exchange. At the end of 2011, Barclays' assets achieved US\$2.42 trillion, and it ranked the seventh-largest bank worldwide (Group, 2014).

During the 2008 financial crisis, Barclays announced its agreement to purchase the investment banking and trading divisions of Lehman Brothers (White & Dash, 2008) that same year. In the end, Barclays PLC paid US\$1.35 billion to acquire the core business of Lehman Brothers, including Lehman's US\$960 million midtown Manhattan office skyscraper and the responsibility for 9,000 former employees (BZW - History - 21st Century - Lehman Brothers Acquisition, 2017).

Automated regression testing for any particular web application involves using a web browser to verify that the web app is working as expected. If done correctly, it can increase the effectiveness, efficiency, and coverage of the software testing. Time saved translates into cost savings. There are different types of software tests that can be automated, for example: unit testing, functional testing, integration testing, regression testing, in which functional testing and

regression testing are what related to this project.

2.1 Current GUI Web Application Automation Testing Framework

The framework consists of three components, quality center, QTP, and Excel. Quality Center stores the test cases; QTP accesses the web browser (it executes the tests); Excel stores the data during the execution; VBScript ties all the previous components together. All three components are tied together by VBScript. The framework is shown below in Figure 1.

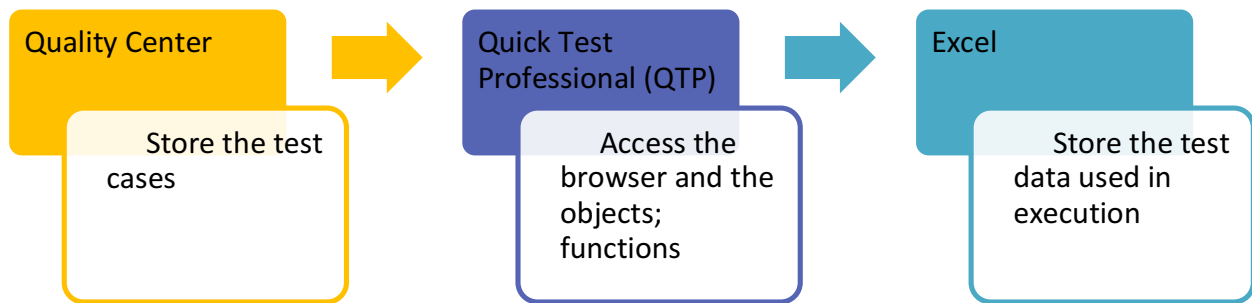


Figure 1: Current GUI Testing Framework Workflow

2.1.1 Quality Center

HP Quality Center is Hewlett-Packard's software quality management product, part of the company's application lifecycle management (ALM) software suite. It is a web-based tool that manages and drives efficient application testing processes. Quality Center stores all the test cases. Following by descriptions, each test case is written step-by-step and comes with a command function. The quality assurance team we are working with has all the command functions in VBScript scripting language. Quality Center clearly shows the objects being used, the object parents, and the value that is being applied to each function. By referring to the step

names, the test cases can be extracted from the core functions. Then, QTP can run the extractions to test the GUI Web application. Figure 2 is a sample screen shot of Quality Center.

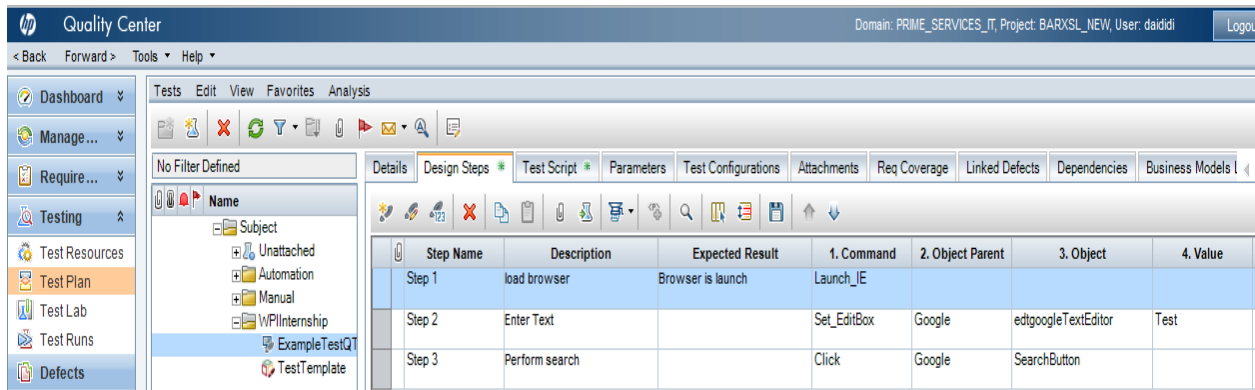


Figure 2: Quality Center

2.1.2 QTP/UFT

HP Unified Functional Testing, also known as UFT, and previously known as QuickTest Professional, or QTP, is an automation testing tool. Barclays relies on this software to do regression and functional test for its GUI Web applications. QTP executes one-line commands to get the objects. After getting the object, it then runs the test cases extracted from the core functions, which in this case are all written in VBScript scripting language. All the objects are saved in the Object Repository, which is written in QTP scripting language. To get the object, testers can simply use the function below:

```
Browser("Google").Page("Google").WebEdit("q")
```

This QTP function will then browse the object on the GUI Web App. Finally, by doing record and playback, QTP inspects the quality of GUI Web application. Figure 3 is a sample screen shot of QTP:

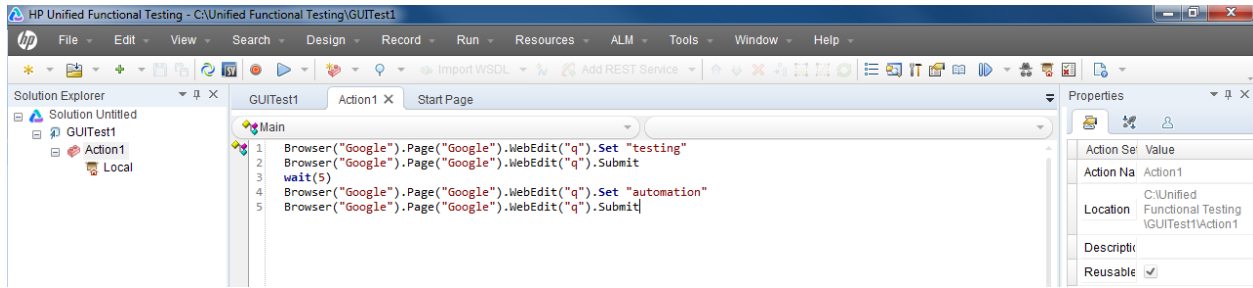


Figure 3: UFT/QTP

2.1.3 VBScript Scripting Language

VBScript scripting language is used to tie the entire framework together. Barclays has the core functions written in VBScript to extract the design steps of the test cases. After the extraction, the command set, which is also written in VBScript, then executes the functions. Finally, the function “Get_Object” is used in VBScript to call the OR Translator in QTP.

2.1.4 OR Translator

The object repository is a centralized location that stores all test objects and elements available from the application that are being tested along with its information or properties (objects’ information). These are later used by the test script to identify specific items during the execution of the test. Stored objects within an object repository can be any of the elements that are present in the GUI Web applications. These objects can be anything that is visible on the GUI (i.e. text fields, images, list items, etc.).

2.1.5 QTP Object Repository

Object Repositories are composed of different components (Reilly & Tupelo-Schneck, 2010):

- Object Identifiers: Every new object that is added to the object repository will get an individual identifier. The identifier helps the test scripts locate a specific object more

easily. After locating them, the test scripts proceed to work on them.

- **Object Class:** The object class value is assigned based on the object's type. Every software has its respective set of object class values. For example, QTP/UFT has its own built-in list of classes for different categories (i.e. list box, edit box, button, images, links, etc.).
- **Object Location:** The object location value defines where the object is going to be stored, regardless of whether it is part of a Local Object Repository or a Shared Object Repository.
- **Object Properties:** Object properties are the properties and characteristics that are used to identify the objects in a specific application.
- **Visual Identifiers:** Visual identifiers are sets of definitions that help identify the objects by using the object's relative location to neighboring objects. Visual identifiers help enhance the process of identifying objects, which then helps reduce the probability of falsely recognizing an object.
- **Ordinal Identifiers:** Ordinal identifiers are additional properties that help differentiate the objects from each other. There are three kinds of ordinal identifiers:
 1. **Index:** This indicates the order in which the objects show up in the application relative to other objects with an identical description. It starts with an index of 0.
 2. **Location:** This indicates the order of the object in the parent window or frame.
 3. **Creation time:** This indicates the sequence in which the browser is opened.

After knowing what an object repository consists of, we need to acknowledge that there are two different types of object repositories. First, there is a type called a Local Object Repository, which is just available to a particular action. This kind of OR employs a .mtr file extension. The objects added to a Local Object Repository cannot be used in any other action,

even if they are the part of the same test. In this type of object repository, the objects are stored for a particular action and only that particular action can access its respective stored objects. This kind of repository is best suited for backing up test objects and for learning new objects. This kind of repository is preferable to test non-dynamic applications with respect to time. It is important to remember that all repositories are local by default. The following list is a list of Local Object Repository operations:

- Local Object Repositories highlight an object stored in the repository;
- Local Object Repositories check whether a particular object in the Application Under Test is stored in the Object Repository;
- Local Object Repositories cut, copy, paste, modify, or delete objects;
- Local Object Repositories update a property's description from the application using the update function, in the rare case where a property's value was changed by accident.

The second type of object repository is called a Shared Object repository, which has a global scope. This kind of OR uses a .tsr file extension and can be used by multiple actions in different tests. This flexibility makes this the ideal repository type for storing and preserving test objects. In order to create a Shared Object Repository, one needs to first export the local objects. Before being able to use the .tsr file, it is necessary to associate the repository to the test by clicking on "Resources" and then "Associate Repository".

2.2 Problem Statement

Employees in the Barclays Core Prime Brokerage have QTP license contention issues since not all of them can access QTP at the same time. This inhibits and affects their work efficiency. Moreover, as the cost of QTP license is expensive, Barclays has stopped investing in new licenses

for this software. Therefore, the current license count is not enough. Due to these issues, many Barclays employees have complained and reported issues with the software. However, not much has been done since they are still working on testing projects from GUI Web applications that solely depend on QTP. Meanwhile, over 1000 GUI automated test cases were created using QTP, so all these test cases need to be preserved. Thus, the main purpose of this project was to look for a replicable free software, which can preserve the legacy test cases while also solving the QTP license issues by replacing it.

Chapter 3: Methodology

3.1 Implementation Criteria

After interviewing our manager and gathering more information about the problems the team has from using QTP, four criteria were identified, which must be met for a new free GUI Web Application automation testing tool to be implemented within the Barclays Core Prime Brokerage Quality Assurance team: zero-cost, VBScript scripting language support, open-source software, and GUI-web application testing support. Any feasible solutions have to be free, open source, while supporting both the VBScript scripting language and GUI web application testing. It is important to acknowledge that all criteria are equal. Any possible automation tools that may have failed to meet the standard of either one of the four criteria were eliminated.

3.2 Comparing Existing Automation Tools

As discussed in Section 2.1.2, the QTP tool is costly and is not open source. To start replacing QTP, we identified nine alternative tools which could be used to test GUI Web applications. These include Selenium WebDriver, Microsoft Coded UI Test, Test Complete, Rational, iMacros, Oracle Application Testing Suite, Eggplant Functional, Sahi, and Maveryx. Our findings are summarized in Table 1.

Software	Cost	VBScripting Language	Open source	GUI support
Unified Functional Testing (QTP)	No	Yes	No	Yes
Selenium WebDriver	Yes	Yes	Yes	Yes
Microsoft Coded UI Test	No	Yes	Yes	Yes
Test Complete	No	Yes	No	Yes
Rational Functional	No	No	No	Yes
iMacros	No	Yes	No	Yes
Oracle App Testing Suite	No	Yes	No	Yes
Egplant Functional	No	No	No	Yes
Sahi	No	Yes	Yes	Yes
Maveryx	No	Yes	Yes	Yes

Table 1: Potential Software Comparison Table

To determine the tool which best satisfies Barclays criteria, we created an indicator as seen below. For each criterion, a zero indicates that an alternative does not meet the criterion, while a one indicates that it does. For example, a “1” in iMacros in the VBScript scripting language criterion indicates that it supports VBScript. This means that by using this software, all legacy test cases can be preserved. Each row, which is the score on each criterion, is summed at the end. The greater the score, the better the tool is in terms of being able to meet Barclays’ requirements. The total weight is then added to the last column on the right. By comparing the total weight, we boxed the highest, which in this case is Selenium WebDriver. Selenium is the best tool since it has a zero cost and is an open source tool. It also supports GUI. And by using Java, the VBScript testing scripts can also be accessed by Selenium.

Software	Cost	VBScripting Language	Open source	GUI support	Total Weight
Unified Functional Testing (QTP)	0	1	0	1	2
Selenium WebDriver	1	1	1	1	4
Microsoft Coded UI Test	0	1	1	1	3
Test Complete	0	1	0	1	2
Rational Functional	0	0	0	1	1
iMacros	0	1	0	1	2
Oracle App Testing Suite	0	1	0	1	2
Eggplant Functional	0	0	0	1	1
Sahi	0*	1	1	1	3
Maveryx	0*	1	1	1	3

Table 2: Best Option Among All Software Tools

To better illustrate the comparison process, see Figure 4. From this figure, the blue bar represents the cost, the orange bar represents the VBScript scripting language, the gray bar denotes that it is open source, and the yellow bar denotes GUI Web app testing support. The bar chart shows the relationship of the four criteria with each tool. With the highest bar, Selenium stands out. Since Selenium is the only tool that fulfills all four criteria, it is, therefore, the only feasible solution. For the rest of the tools, they did not qualify for at least one criterion. The worst options are Rational Functional and Eggplant Functional. The ones close to becoming feasible solutions are Microsoft Coded UI Test, Sahi, and Maveryx. A more detailed reason for the elimination processes will be written in the next section.

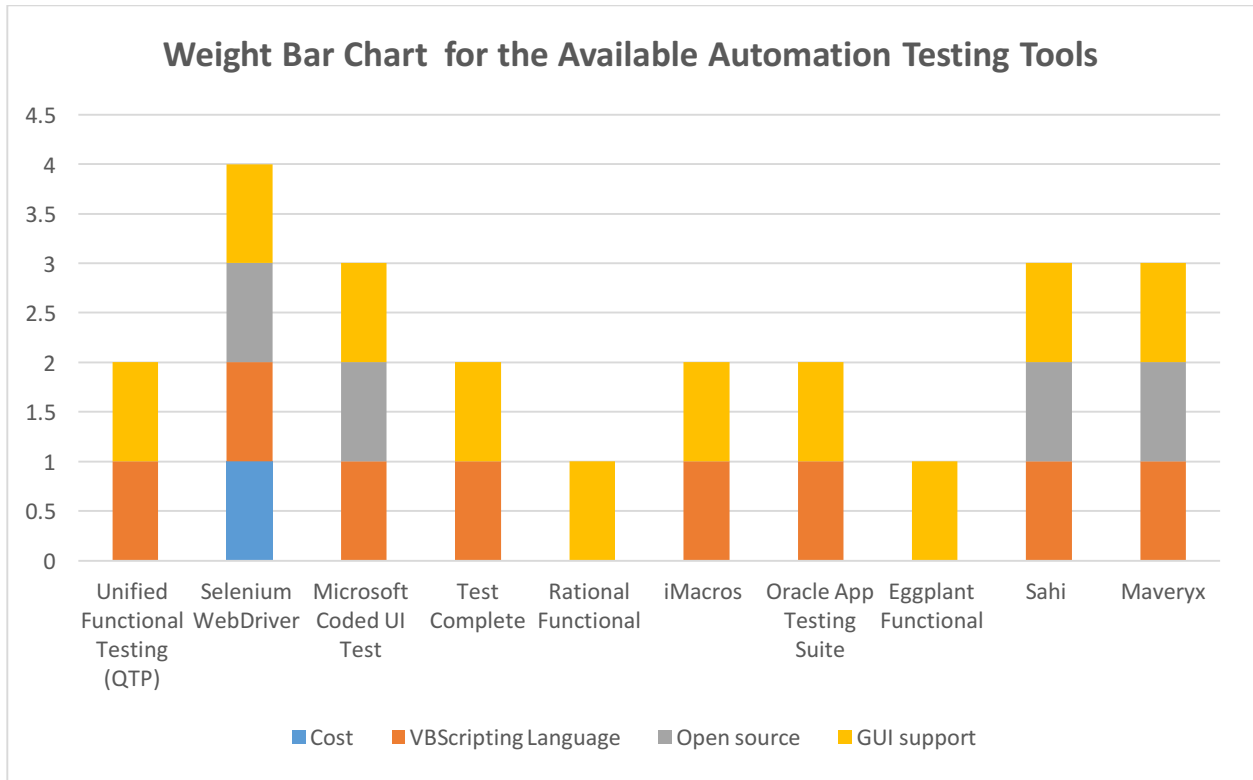


Figure 4: Weight Bar Chart for the Available Automation Testing Tools

3.3 Rationale for Eliminating Alternatives

3.3.1 Microsoft Coded UI Test

Coded UI uses MSAA and UIA technologies to identify objects in an object repository (or what most call UIMap), while UFT uses a coded-injection approach.

Coded UI conducts a test in two ways: either by record and playback or by hand coding. It mostly supports C# language and not VBScript. We looked into converting VBScript to C# (Microsoft, 2015). However, going from VBScript to CodedUI's object oriented C# is not that easy as just adding a plug-in. While QTP is a functional testing tool, CodedUI is used more for unit testing; thus, there are many architectural differences. Due to these reasons, this option was eliminated.

3.3.2 Test Complete

Test Complete is one of the few existing automation testing software that supports VBScript language. However, it is a proprietary functional automated testing platform owned by SmartBear Software. The yearly cost is \$7000 for one license (SmartBear, 2017). We discarded this option as our focus is to find tools that are free and open-source.

3.3.3 Rational Functional

Rational Functional tester is from the Rational Software division of IBM. It does provide automated testing capabilities for functional, regression, GUI, but as it only satisfied one criterion, it was then eliminated. This software provides no support for VBScript.

3.3.4 iMacros

iMacros can record and replay repetitious work and is the only web automation software that mostly works with every website. However, it is not an open source software tool, and therefore, not a feasible solution that we can consider.

3.3.5 Oracle Application Testing Suite

The Oracle Application Testing Suite CD-ROM includes the Microsoft® Visual Basic® Scripting Edition (VBScript) Language Reference documentation. It is a rare existing software that supports VBScript (Oracle, 2008). Although it supports both VBScript language and GUI Web app testing, it is neither free or an open source tool.

3.3.6 Eggplant Functional

Eggplant Function was eliminated as it only satisfied the criterion of GUI Web app testing. The scripting language it uses is called SenseTalk (TestPlant.com, 2017); therefore, it would not

be possible to use the VBScript testing cases currently in existence.

3.3.7 Sahi

Sahi is an open source tool for testing GUI Web applications that works on every browser. It also supports the VBScript scripting language. However, the only deficiency is that it is only free to access the Sahi OS. However, even though it is free, Sahi OS only includes limited features. To get the access to all of Sahi's features, it is mandatory to purchase Sahi Pro, which costs \$695 for a license (Pro, 2017). Due to its imminent cost, this option was then eliminated.

3.3.8 Maveryx

Maveryx is an automated functional and regression test tool for Java and Android applications. We researched this tool because almost all the GUI Web apps that we have in Barclays are written in Java. The software only free when running on a free-trial mode, meaning that it offers limited functions for a limited time. For a large firm like Barclays, this cannot be a long-term solution.

After researching each of these tools, there is only one remaining alternative, which is Selenium WebDriver. It is the only feasible solution that satisfies all four criteria. The following sections will discuss more about how to replace HP's QTP/UFT with Selenium WebDriver.

Chapter 4: Two Feasible Automation Tools

This chapter will specifically explain how Selenium WebDriver works and unveil another feasible solution apart from using the existing software tool. It will cover the background, the working principle of each feasible tool, and numerous ways to replace the Object Repository that is found in QTP. The reason why it is important to focus on QTP Object Repository is because it is the core process that needs to be replaced. To be specific, it is the only part that is not written in VBScript and that directly connects to QTP. Additionally, the other parts of the testing framework are easier to fix if the object repository implementation is solved.

4.1 Alternative #1: Selenium WebDriver

4.1.1 Background

Selenium is an open-source tool that is commonly used for test automation particularly in the Financial Services Industry. Selenium WebDriver was first introduced in Selenium 2.0 and it directly calls the browsers by using the browser's native support for automation and other browser-specific features. Its unique platform and language neutral interface allows for a more seamless control of a Web browser. How Selenium WebDriver works is shown in Figure 5 below:

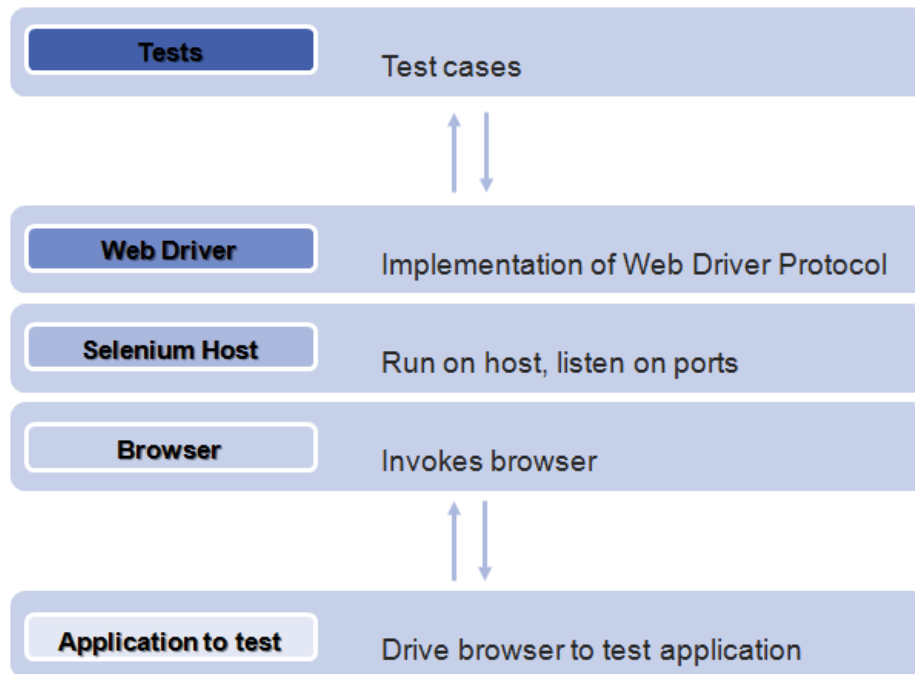


Figure 5: Selenium WebDriver Working Principle

4.1.2 Browser Compatibility

Browsers have different compatibilities. For example, Mozilla Firefox supports WebDriver by default. However, Chrome and IE (32-bits and 64-bits) require a specific executable driver (SeleniumHQ, 2014) . To solve this problem, the first step is to download the executable driver from Google’s Chromium website

(<https://sites.google.com/a/chromium.org/chromedriver/downloads>)

and include the following line of code into Selenium:

```
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
```

Even though Selenium WebDriver supports multiple browsers, we will be focusing more on IE and Google Chrome as Barclays only supports these two. Having said this, it is important to

consider the possibility of adopting a headless browser testing model or a browser simulator (i.e. HTMLUnit) since Selenium WebDriver also supports this. Significant processing resources can be saved by using a headless browser. And since all the functions would run in the background, this would then create an even more seamless and “real” automated testing environment. Additionally, one could start automating on a headless server (i.e. continuous integration servers such as Jenkins).

4.1.3 Calling Java by Using VBScript

To preserve the existing testing code that is written in VBScript, we had to find a way to link JAVA and VBScript together for the following reasons:

1. Selenium cannot use VBScript directly, but Selenium can use JAVA;
2. JAVA is one of the most commonly used scripting languages nowadays;
3. It is possible to connect JAVA with VBScript.

Before proceeding, we have to identify how Selenium would preserve VBScript. After researching, we found that we can use JAVA to get the object, and then use VBScript to call JAVA.

To get the object, the code we can apply in JAVA is shown below:

```
WebDriver driver = new ChromeDriver()

WebElement element = driver.findElement(By.name("q"));

element.sendKeys("test")
```

This specific code is shown in Appendix 1. Appendix 1 is the extension of the code seen above. Whereas the code above just contains the main functions that perform the executions.

After getting the object, we use the code in VBScript to call the JAVA code in order to test the objects found in the test cases that are written in VBScript. The code to invoke JAVA by using VBScript is shown in Appendix 2. All the code found in the Appendices are written with the help of our manager, Karlene Wright.

4.1.4 Instantiating a Specific Browser

To install a specific browser, we need to create a browser name by adding a new driver. This piece of code is written in Java. The web browser names can be: Firefox, Chrome, Safari, or InternetExplorer. The code is shown below:

```
WebDriver driver = new "BrowserName"Driver();
```

To access a webpage in Selenium, we need to use WebDriver and HtmlUnit Driver classes:

```
WebDriver driver = new HtmlUnitDriver();
```

```
WebElement element;
```

More sample code to open a browser, which checks to see which browser is currently being used is shown in Appendix 3. The sample code to run test cases in Google Chrome using Selenium WebDriver and the sample Java code implementing a Google search in Selenium are also included in Appendix 3 for reference.

4.1.5 Other Ways to Get the Elements by Using Selenium

There are multiple ways to get the elements in Selenium. Selenium uses JAVA to get the object, so the most commonly used function is the ".FindElement" function found in JAVA. The

".FindElement" function can be used in different ways to access the objects. In most cases, programmers find the element using the object's name and ID. Sometimes, the elements can also be found by using ClassName, Tagname, LinkText, partialLinkText, or XPath. The code to find an element through the numerous aforementioned ways can be found in Appendix 4.

4.1.6 How to Implement an Object Repository in Selenium?

It is hard to look for an object every time when we need to test it. However, it's easy for QTP, because it has an integrated object repository that stores all the objects. If we are going to use the Selenium WebDriver to replace QTP, we had to find a way to store the objects in order to make it easier to find the objects. After conducting our research, we found three methods to apply an object repository within Selenium:

1. We can store the OR in an Excel file (.xml), just like QTP does.
2. We can use properties files.
 - This method uses key-value pair format, which tends to be more readable by the software.
3. We can apply the Page Object Model or POM.
 - This method will ensure the OR's structure is similar to QTP's OR structure. It also means that the final command will look exactly like a QTP command.

To further explain how these three possible ways, how they work, and how to use them to execute the object repository's work, we detailed everything below step by step:

Excel file (.xml)

An example of an XML being used as an object repository, including code retrieval and

execution, is shown in Appendix 5.

Property Files

Background about Property Files

We can create an object repository by relying on the properties file feature found in Java. The .properties file in Java is just a basic collection of key-value pairs. Therefore, the OR could then be implemented as a collection of key-value pairs, with the key being a logical name identifying the object and the value containing unique objects properties used to identify the object on a screen. This can be achieved by comparing the file's properties to a specific format that is recognized by the class `java.util.Properties`. The rules of the Java Properties are as follows (ATG, 2016):

1. Entries are generally expected to be a single line of the form, either one of the following two:
 - o `propertyName=propertyValue`
 - o `propertyName:propertyValue`
2. White space that appears between the property name and the property value is ignored, as exemplified below:
 - o `name=Stephen`
 - o Whitespace at the beginning of the line is also ignored.
3. Lines that start with the comment characters `!` or `#` are ignored. Blank lines are also ignored.

4. The property value is generally terminated by the end of the line. White space following the property value is not ignored and is treated as part of the property value.
5. A property value can span several lines if each line is terminated by a backslash ('\') character. This can be seen below:

```
targetCities=\n    Detroit,\n    Chicago,\n    Los Angeles
```

This is equivalent to `targetCities=Detroit,Chicago,Los Angeles`. In this case, we have to remember that the white space at the beginning of the lines has to be ignored.

6. The character's newline, carriage return, and tab can be inserted with characters `\n`, `\r`, and `\t`, respectively.
7. The backslash character must be escaped with a double backslash. This is exemplified below:

```
path=c:\\docs\\doc1
```

One important thing to note is the fact that it is not necessary to escape backslashes in property values when using the ATG Control Center Components window; the ATG Control Center will handle the escape characters automatically.

8. UNICODE characters can be entered as they are in a Java program, using the `\u` prefix. For example, `"\u002c"`.

We can use the Java interface in Selenium to read the object properties into variables and then have the scripts reference the variables present in the interface. To do so, we must first decide the format of the properties file. After that, we can store all the element locators in a

.XML file as an object repository. However, one drawback of this method is that it might take a long time to manually create all the property files. A sample properties file format is the following one (Chavan, 2016):

```
[logical_name]=[locator_type]~[locator_value]
```

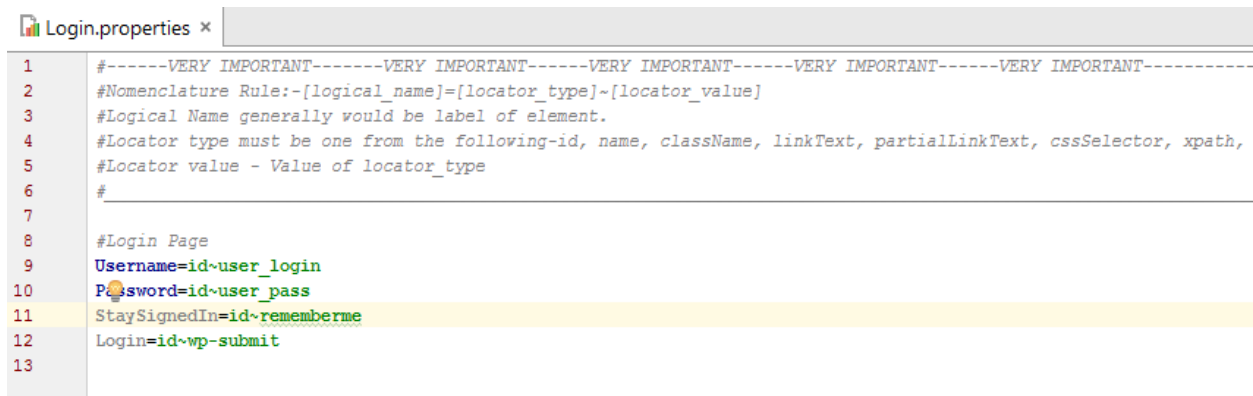
The definition of each term is shown below:

[logical_name]: The logical name generally would be label of element. [As displayed on the UI]

[locator_type]: The locator type must be one from the following types: id, name, className, linkText, partialLinkText, cssSelector, xpath, tagName.

[locator_value]: This is the locator value, or the value of locator_type

This is how the property file would end up looking like, as seen in Figure 6:



```
1 #-----VERY IMPORTANT-----VERY IMPORTANT-----VERY IMPORTANT-----VERY IMPORTANT-----VERY IMPORTANT-----
2 #Nomenclature Rule:-[logical_name]=[locator_type]~[locator_value]
3 #Logical Name generally would be label of element.
4 #Locator type must be one from the following-id, name, className, linkText, partialLinkText, cssSelector, xpath,
5 #Locator value - Value of locator_type
6 #
7
8 #Login Page
9 Username=id~user_login
10 Password=id~user_pass
11 StaySignedIn=id~rememberme
12 Login=id~wp-submit
13
```

Figure 6: Sample Property File

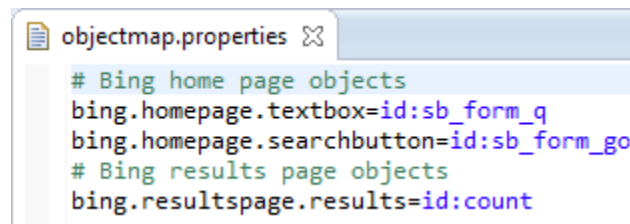
Creating the Object Repository in Selenium (Code for Running a Simple Bing Search)

To create an object repository in Selenium by using the properties file, we need to write the code for doing the Bing search. In this particular case, a Bing search script would need to manipulate three screen objects:

- The textbox where the search string is typed;
- The search button that is clicked to submit the search query;

- The text field that displays the number of search results

The object map will be a simple `.properties` text file that we can add to our Selenium project, as seen in Figure 7:



```
objectmap.properties
# Bing home page objects
bing.homepage.textbox=id:sb_form_q
bing.homepage.searchbutton=id:sb_form_go
# Bing results page objects
bing.resultspage.results=id:count
```

Figure 7: Object Map

The key for each object, i.e. `bing.homepage.textbox`, is a logical name for the object that we will use in our script. The corresponding value consists of two parts: the attribute type used for uniquely identifying the object on screen and the corresponding attribute value. For example, the aforementioned text box is uniquely identified by its `id` attribute, which has the value `sb_form_q`.

Retrieving and Fetching Objects from the Object Repository

To retrieve objects from a newly created object map, we would need to define a class, `ObjectMap`, with a constructor that takes a single argument, which is the path to the `.properties` file. Details on how to write an object map can be referred to Appendix 5. Objects can be identified using a number of different properties (i.e. object IDs, CSS selectors and XPath expressions).

How Can We Use the Objects from an Object Map?

After being able to retrieve objects from the class object map, we can start using these

objects in the script as part of the functions. The code to execute the method of accessing an object requires a very easy and simple step: This step involves referring to the key in the object map. This is also called as logical name and this code is shown in Appendix 6.

Page Object Model

Background

A better approach to script maintenance is to create a separate class file, which can then find web elements, fill them, or verify them. This class can be reused in all the scripts that use the same element. In the future, if there is any change in the web element, we would then need to make change in just one class file and not in all ten different scripts. This approach is called Page Object Model (POM). This approach helps make code more readable, maintainable, and reusable. The principle of POM is showed below, in Figure 8. If implemented properly, POM can help organize the code, reduce maintenance efforts, and reduce script creation time.

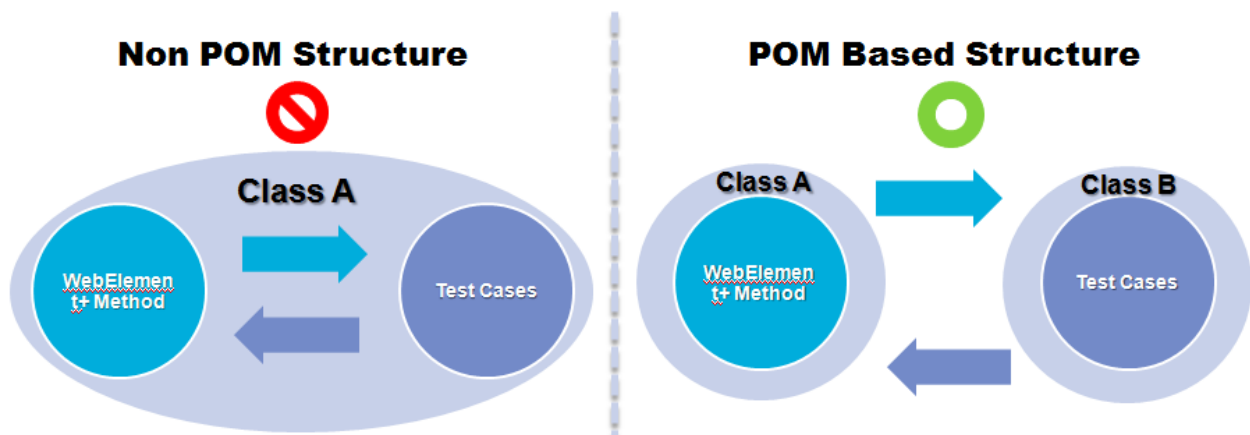


Figure 8: The Principle of the Page Object Model

- In a POM model, each web page in the application must have a corresponding page class.
- This page class will find the web elements of that web page and will also contain a function

that will perform operations on the web elements.

Advantages of the Page Object Model

The code is cleaner and easier to understand because the structure requires that all the UI operations and flows be separated from verification. In addition, the Object Repository is independent of the test cases. The same OR can be used for numerous purposes and numerous tools. Last but not the least, given the fact that the OR can be reused, the amount of code is thereby reduced and optimization can thereby be attained.

How to Set POM Up?

Follow the steps below to set the POM up and the code can be found in Appendix 7:

1. Right click on 'x' package then select **New > Class** and name it as "**ProductListing_Page**".
2. Create another public static class inside the above class 'x' and name it as **Product_1**.
3. Now create different Static Methods for each child element of Product_1. These methods will need an Argument (WebDriver) and a Return value (WebElement).

To test the code: Type ProductListing_Page in the test script and press dot. This will display all the products we have specified in the class, as seen in Figure 9.

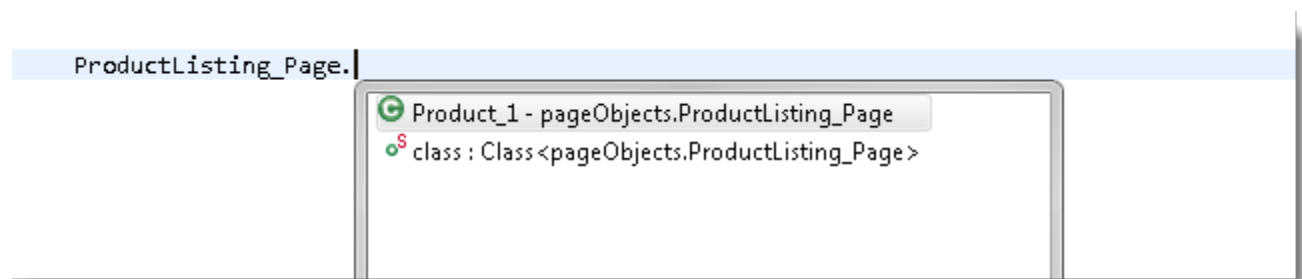


Figure 9: Example of a Page Object Model in Action

Select Product_1 and press dot again. This will now display all the child elements

associated with the parent Product_1, as seen below in Figure 10:

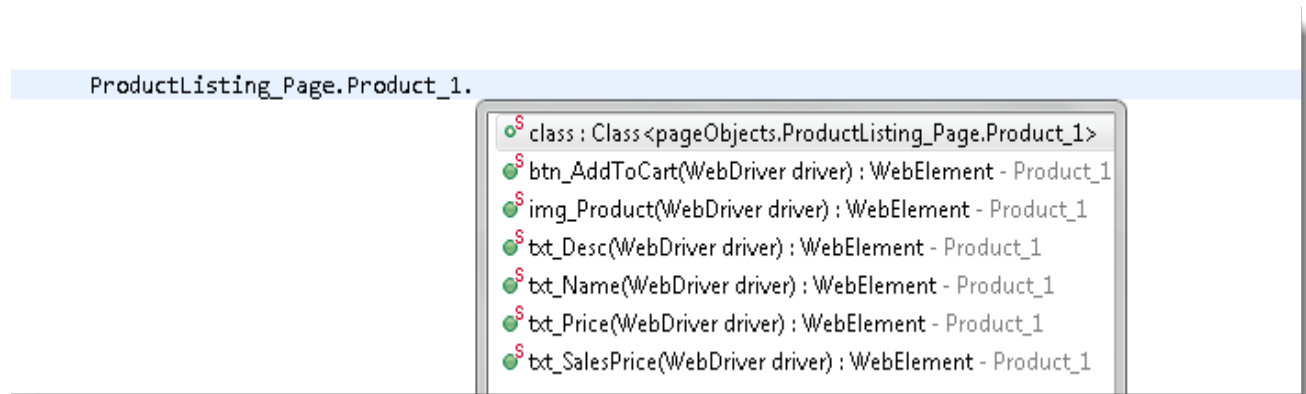


Figure 10: Child Elements in the Product

The resulting command line is QTP-like:

```
ductListing_Page.Product_1.btn_AddToCart(driver).click();
```

4.2 Alternative #2: Using Internet Explorer Object

4.2.1 Background

Internet Explorer Objects contains three types of members: Events, Methods and Properties. Visual Basic can access this functionality since Internet Explorer supports automation. For example, the function “CreateObject” can be used in Visual Basic to launch an instance of Internet Explorer.

4.2.2 How Does It Work?

For example, let’s create a VBScript that can be used to test a Google webpage. To access the Google website URL, we have two pieces of codes in VBScript that are attached in Appendix 8. Specifically, we use the Internet Explorer Object to control an instance of Windows Internet

Explorer through automation. Then .application gets the automation object for the application that is hosting the web browser control. Finally, we use the “Navigate” function to navigate the resource identified by a URL. In addition to grabbing objects through the URL, we also researched the way to get objects by ID. These codes are shown in Appendix 8. Additionally, we can also get the web button and the web table by using a similar code in VBScript, which is also shown in Appendix 8.

4.2.3 How to Implement the Object Repository in the In-house Development Tool?

There are two methods to store the objects: Using either Excel or the reference dialog box method.

Excel (.XML)

Similar to the object repository, which is originally introduced in QTP, we can create our own object repository by creating an XML file. To create our own object repository, there are two methods that can be used: Either using "ObjectRepositoryUtil Object" or using the object repository manager. Then, we can work with object repository files (Local/Shared) from outside of UFT.

For the first method, to convert an object repository into .XML format, we either enter "ObjectRepositoryUtil" in the search box underneath the Index tab or use the code found in Appendix 8.

For the second method, we can simply export the .XML file, by using the object repository manager. We do so by:

1. To open the object repository manager, click on Resources

and then clicking on Object Repository Manager

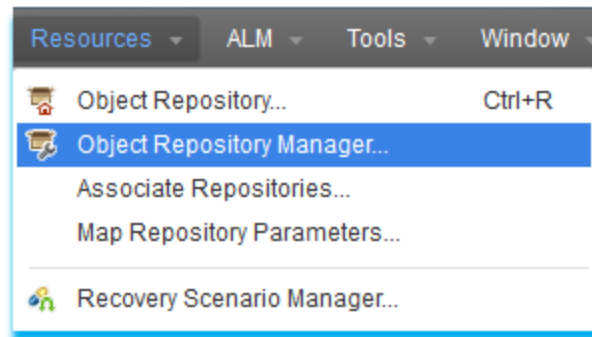


Figure 11: Object Repository Manager

2. Opening the Object Repository, then clicking on File, and then open
3. To export to .XML format, click on File and then on Export to XML

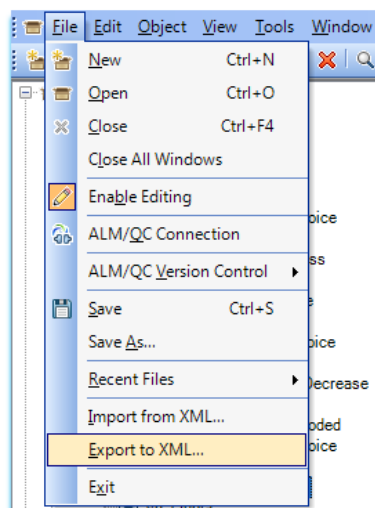


Figure 12: Exporting the XML File

To add the object by using VBScript, we load an existing object repository and use the AddObject function to add an object to it. This is an example of how we can add the object by using VBScript. So reversely, it is also possible to get the object by using VBScript. The detailed code is attached to Appendix 9.

Additionally, there is another way that can be used to get the object from XML file. The code for this way is also attached to Appendix 8.

Reference Dialog Box

We can make the objects in the Internet Explorer available to Visual Basic 4.0 through the reference dialog box. The reference dialog box lists all the objects available to Visual Basic. The dialog box can be accessed by selecting Tools/References from the menu bar. The reference dialog box has a reference set to the Internet Explorer object library. The proper reference is described in the dialog box as "Microsoft Internet Controls."

Once a reference is set in Visual Basic, we can start exploiting the objects in the code. For example, to get an instance of the Internet Explorer browser, which can be accessed from Visual Basic, we could use the following code:

```
Dim MyBrowser As SHDocVw.InternetExplorer  
Set MyBrowser = New SHDocVw.InternetExplorer
```

The reference dialog box allows us to select another application's objects that we want available in our code by setting a reference to that application's object library.

4.3 Comparing the Two Feasible Solutions

4.3.1 Selenium WebDriver

Using Selenium's WebDriver would be beneficial in the long run given the fact that Selenium's WebDriver specification is currently in draft form to become a W3C (World Wide Web Consortium) Web Standard. This will mean that any web browser software vendor would be required to support Selenium WebDriver by having their own implementation of it. Although Selenium has its own scripting language, it is not restricted to write in that language since it can work with other language bindings to support whatever scripting language the testers are

comfortable with, including C#, Java, JavaScript, PHP, Python, among others. Additionally, Selenium offers record-and-playback capabilities. Because of the rapid test development that Selenium enables, it is quite popular for quick-cycle development methodologies such as Agile or Extreme Programming. Moreover, Selenium is also popular with IT staff who automate repetitive, web-based administrative tasks. Furthermore, Selenium supports most operating systems, among them Windows, Macintosh, Linux, Unix, and etc. Also, Selenium can be integrated with Jenkins or Hudson for continuous integration (Kumykov, Luo, Shoop, & Zhang, 2016). Finally, Selenium can be used for Android, iPhone, Blackberry, and etc. based application testing, which means that it will be useful should Barclays start adopting mobile GUI web apps.

Despite its numerous features, Selenium is not a complete, comprehensive solution to fully automate the testing of web applications. It requires third-party frameworks, language bindings, and so on to be truly effective. Because native “Selenese” test scripts are not user-friendly in terms of readability, they are difficult to modify. Additionally, Selenium has technical issues with browsers which are not Mozilla Firefox. Furthermore, it does not support conditionals, loops, and has trouble finding locators without the help of additional tools such as Firebug.

4.3.2 Internet Explorer Object

On the other hand, all of Barclays’ test cases are easier to debug using Internet Explorer Object, because everything would be built in-house. It is very user-friendly to choose to apply in-house development tool because all codes will be written in one language, which is VBScript language. Since VBScript is very similar to QTP language, it is easier to translate the QTP part of

code. It would also be easier for the VBScript experienced staff to handle the new tool since they already know how to write in VBScript. Additionally, the codes to get different objects in QTP are similar to the codes used in Internet Explorer Object, so it would be more simple to manage.

However, there are also disadvantages of using this in-house development tool. First, Barclays would have to manually write all the functions to operate the entire testing framework. This would indirectly cost the company money as they need to hire and train people to write all the functions and build the entire library. It will also take time to develop a new in-house tool, even though we have a general idea on how to get this done. Finally, it is harder to debug the code due to the lack of wider online user community. In order to check the bug, we would have to run the code and then check for the error manually.

Chapter 5: Recommendations and Conclusions

5.1 Conclusion

After comparing the ten alternatives, there is only one tool satisfies all four criteria — Selenium Webdriver. By using Selenium Webdriver as an automation testing software, we first need to write code to use the programming language JAVA to get objects. Then, we need to link JAVA to VBScript by invoking JAVA files by using code that would be written in VBScript. To fully replace the current Barclays testing tool, QTP/UFT, we researched three ways to replace object repository to maintain the functionality of storing objects present in QTP.

However, the main disadvantage of Selenium is the inconsistency of obtaining the objects. Thus, we continued researching and found a way to develop an in-house development tool by using Internet Explorer Object. In this way, we can use VBScript to get the object instead of using QTP/UFT. There is no need of any plugins to connect the “get element” functions with the testing cases since all test cases are written in VBScript. This would largely increase the system consistency and save the team time to study new programming languages. This would be the best solution in the short run.

In comparing Selenium WebDriver against Internet Explorer Object, Selenium has the advantages of increasing the efficiency of web application testing, having a function library, supporting multiple operating system, and drafting to be the standard of the World Wide Web Consortium, which outweigh the advantages of Internet Explorer Objects, which is easier to debug and more consistent to run the VBScript test cases.

5.2 Recommendations

In the short run, we suggest to create an in-house development tool by using the VBScript scripting language, because it ties better to our current testing framework. In addition, the VBScript language is the QTP-liked scripting language. It is easier to convert the QTP/UFT piece of code to VBScript language. No other programming language would interfere with the changes. As a consequence, the migration from QTP to the in-house development tool can happen smoothly.

However, in the long run, we suggest replacing QTP/UFT with Selenium WebDriver, because it is a trend that all manual tests are going to be automated. Especially for doing the regression test that needs to be executed over and over again. The selenium would be useful in the long term. If Barclays will develop the mobile GUI web app in the future, Selenium WebDriver would also be useful for the testing purposes. In addition, we recommend the team to start writing testing cases in Java and use Selenium WebDriver for GUI Web app testing purposes in future new projects.

Even though Selenium WebDriver supports multiple browsers, we will be focusing more on IE and Google Chrome since Barclays only supports these two. Having said this, it is important to consider the possibility of adopting a headless browser testing model, or a browser simulator (i.e. HTMLUnit) since Selenium WebDriver also supports this. Significant processing resources can be saved by using a headless browser. And since everything would run in the background, this would then create an even more seamless and “real” automated testing environment. Additionally, one could start automating on a headless server (i.e. continuous integration servers

such as Jenkins).

Through these recommendations, we hope that it would save the team the yearly cost of their UFT licenses and prevent any other execution issues that they are currently facing when dealing with UFT. These recommendations could eventually be useful to other Barclays teams, should Barclays decide not to use UFT in the future.

Chapter 6: Reflecting on our Experience at Barclays

6.1 Discussion of Design in the Context of the Project

Our project at Barclays was an opportunity for us to apply what we have learned at WPI. WPI challenged us to think critically and to work in a fast-paced environment. This allowed us to adapt more easily into Barclay's challenging working environment. As a major global investment bank with a long history, Barclays continuously strives to be a forward-thinking organization in a highly competitive industry. Due to demanding nature of the business, Barclays places a large emphasis on employing cutting-edge technologies, while still striving to maintain the legacy systems they have in place. In order to further Barclays' cutting-edge technologies, we worked with the quality assurance team to improve their current GUI automation testing framework.

6.2 Discussion of Constraints Considered in the Design

At the beginning of the project, we were tasked to find an alternative automation tool that could replace the current testing software in order to reduce costs and increase usability. To achieve this, we had to developed several potential solutions to the issue. Before we started our research, we had to better understand the demands and constraints of this project. After consulting subject matter experts, we devised four constraints that helped us come up with a solution that would not only satisfy their needs, but also be fully compatible with their legacy code.

We explored software tools that were open source and easily modifiable. By using a free and open-source software, we potentially saved the company tens of thousands of dollars in their web application development. Additionally, we ensured that our solutions would support VBScript scripting language because existing test cases were written in this language.

6.3 Discussion of the Need for Life-long Learning

Embedded within WPI's motto, "Lehr und Kunst", is the idea of embracing lifelong learning. Learning doesn't have to stop after graduation. It is not confined to academic learning, since it also means applying what we have learned in order to improve our lives and the lives of those around us.

For the project, after reviewing the original framework of the testing process, we were able to come up with solutions that would satisfy the constraints given to us while still achieving the intended goal. We learned that in such a complex environment, it is hard to eradicate all legacy code.

Despite the fact that we could not upgrade the entire GUI automation testing framework, we feel that our project was a success nonetheless. All in all, we were able to grow and see the importance of applying the knowledge and skills instilled on us by WPI. We were able to witness firsthand the importance of a project-based curriculum and its effects in fostering effective career development.

Works Cited

- ATG, O. (2016). *Oracle ATG Programming Guide*. Retrieved 1 5, 2017, from https://docs.oracle.com/cd/E23095_01/Platform.93/ATGProgGuide/html/s0204propertiesfileformat01.html
- BZW - History - 21st Century - Lehman Brothers Acquisition*. (2017). Retrieved 1 5, 2017, from Wikipedia Acquisition (Creative Commons): http://www.liquisearch.com/bzw/history/21st_century/lehman_brothers_acquisition
- Chavan, A. (2016, 7 16). *Saturday, July 16, 2016 Highlight Element in Selenium WebDriver*. Retrieved 12 14, 2016, from Quality Perspective: https://qaperspective.blogspot.com/2016_07_01_archive.html
- Delaware, B. B. (2016). *Barclays banking history*. Retrieved 1 2, 2017, from Barclays : <https://www.banking.barclaysus.com/our-history.html>
- Group, C. O. (2014). *Barclays*. Retrieved 1 2, 2017, from OMICS International: <http://research.omicsgroup.org/index.php/Barclays>
- IBM. (2016, 3). *Rational Functional Tester*. Retrieved 11 2, 2016, from IBM Software: <http://www-03.ibm.com/software/products/en/functional>
- Kumykov, K., Luo, T., Shoop, A. K., & Zhang, B. (2016, 1). *Development of an Operational Process for Continuous Delivery*. Retrieved 1 2, 2017, from WPI: https://web.wpi.edu/Pubs/E-project/Available/E-project-012216-150216/unrestricted/WallSt_Barclays2_MQP.pdf
- Microsoft. (2015). *Use UI Automation To Test Your Code*. Retrieved 11 12, 2016, from Microsoft Developer Network: <https://msdn.microsoft.com/en-us/library/dd286726.aspx>
- Oracle. (2008). *Oracle Application Testing Suite*. Retrieved 11 25, 2016, from Oracle Technology Network: <http://www.oracle.com/technetwork/oem/app-test/index.html>
- Pro, S. (2017). *The Tester's Web Automation Tool*. Retrieved 1 3, 2017, from Sahi: <http://sahipro.com/>
- Reilly, S., & Tupelo-Schneck, R. (2010, 1). *Digital Object Repository Server: A Component of the Digital Object Architecture*. Retrieved 12 21, 2016, from D-Lib Magazine: <http://www.dlib.org/dlib/january10/reilly/01reilly.html>
- SeleniumHQ. (2014). *SeleniumHQ Browser Automation*. Retrieved 11 25, 2016, from <http://www.seleniumhq.org/about/platforms.jsp>
- Software, S. (2017). *Automated Software Testing TestComplete Platform: Testing For Desktop, Mobile, Web, & Paged Applications*. Retrieved 1 01, 2017, from SmartBear: <https://smartbear.com/product/testcomplete/overview/>

TestPlant.com. (2017). *eggPlant Functional*. Retrieved 1 03, 2017, from TestPlant:
<http://www.testplant.com/eggplant/testing-tools/eggplant-developer/>

White, B., & Dash, E. (2008, 9 17). *Barclays Reaches \$1.75 Billion Deal for a Lehman Unit* . Retrieved 12
23, 2016, from New York Times:
<http://www.nytimes.com/2008/09/18/business/worldbusiness/18barclays.html>

Appendix 1

Using JAVA to get the object:

```
package mypackage;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.WebDriverWait;

public class myclass {

    public static void main(String[] args) {

        System.setProperty("webdriver.chrome.driver", "C:\\selenium-java-2.35.0\\chromedriver_win32_2.2\\chromedriver.exe");

        WebDriver driver = new ChromeDriver();

        driver.get("http://www.google.com");

        WebElement element = driver.findElement(By.name("q"));

        element.sendKeys("Cheese!");

        element.submit();

        //driver.close();

    }

}
```


Appendix 2

VBScript code invoking JAVA:

```
call PerformSearch("qtpvbsdemojar.jar", "Search", "C:/wpiintern/IEDriverServer.exe",  
"http://www.google.com", "Google", "test")
```

```
Public Sub PerformSearch(sJarFile, sFunction, sPath, sURL, sTitle, sInput)
```

```
    sCmdString = "java -jar " & Chr(34) & sJarFile & Chr(34) & " " & trim(sFunction) & " "  
    & Chr(34) & sPath & Chr(34) & " " & Chr(34) & sURL & Chr(34) & "_& " " & Chr(34) & sTitle  
    & Chr(34) & " " & Chr(34) & sInput & Chr(34)
```

```
    sError = ExecCmdString(sCmdString)
```

```
End Sub
```

```
Public Sub PerformSearch2(sFunction, sJarFile, sMainFunction, sFunc, sPath, sTitle, sInput)
```

```
    sCmdString = "java -cp " & Chr(34) & sJarFile & Chr(34) & " " & trim(sMainFunction) & " "  
    & Chr(34) & sFunc & Chr(34) & " " & Chr(34) & sPath & Chr(34) & " " & Chr(34) & sURL  
    & Chr(34) & "_& " " & Chr(34) & sTitle & Chr(34) & " " & Chr(34) & sInput & Chr(34)
```

```
    sError = ExecCmdString(sCmdString)
```

```
End Sub
```

```
Private Function ExecCmdString(sCmdString)
```

```
    Set oExec = CreateObject("WScript.Shell")
```

```
    Set oOutExec = oExec.Exec(sCmdString)
```

```
    ExecCmdString = oOutExec.StdErr.ReadAll
```

```
    If InStr(ExecCmdString, "FAIL") = 0 and InStr(ExecCmdString, "Exception") = 0
```

```
Then
```

```
    ExecCmdString = ""
```

```
End If
```

```
Set oOutExec = Nothing
```

```
Set oExec = Nothing
```

```
End Function
```

Appendix 3

Sample code to open a browser (it checks to see which browser to use):

```
public String openBrowser(String object, String data, String configPath) {
    APP_LOGS.debug("Opening browser");
    if (CONFIG.getProperty(data).equals("Mozilla"))
        driver =new FirefoxDriver();
    else if (CONFIG.getProperty(data).equals("IE"))
        driver =new InternetExplorerDriver();
    else if (CONFIG.getProperty(data).equals("Chrome"))
        driver =new ChromeDriver();
    else if (CONFIG.getProperty(data).equals("Safari"))
        driver =new SafariDriver();

    driver.manage().window().maximize() ;

}

public String clickButton(String object, String data,String configPath){
    OR=new ObjectRepLocator(ObjectRepo);
    APP_LOGS.debug("Clicking on Button");
    try{
        driver.findElement(By.xpath(OR.getLocator(object,configPath))).click();
    }catch (Exception e){
        return Constants.KEYWORD_FAIL+ " Not able to click on button"+e.getMessage();
    }
    return Constants.KEYWORD_PASS;
}
```

Sample code to run test cases in Google Chrome using the Selenium WebDriver:

```
import java.io.IOException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class ChromeTest {

    /**
     * @param args
     * @throws InterruptedException
     * @throws IOException
     */
    public static void main(String[] args) throws InterruptedException, IOException {
        // Telling the system where to find the Chrome driver
```

```

System.setProperty(
    "webdriver.chrome.driver",
    "E:/chromedriver_win32/chromedriver.exe");

WebDriver webDriver = new ChromeDriver();

// Opening google.com
webDriver.navigate().to("http://www.google.com");

String html = webDriver.getPageSource();

// Printing result here.
System.out.println(html);

webDriver.close();
webDriver.quit();
}
}

```

Sample Java code implementing a Google search in Selenium:

```

// Initialize an instance of WebDriver
WebDriver driver = new HtmlUnitDriver ();
// Load Google.com
driver . get ( "https://www.google.com" );
// Proceed to print the title
System . out . println ( "Page title: " + driver . getTitle ());
// Enter the search word into the query box and submit the search
WebElement element = driver . findElement ( By . name ( "q" ));
element . sendKeys ( "Test" );
element . submit ();
// Print the page title
System . out . println ( "Page title: " + driver . getTitle ());
// Proceed to quit the WebDriver
driver . quit ();

```

Appendix 4

The find element by ID function can be written in either way:

```
WebElement Username = driver.findElement(By.id("Username"));  
//Action can be performed on Input Button element  
element.submit();
```

or

```
By locator = By.id("UserName");  
WebElement UserName = driver.findElement(locator);  
By name
```

The function to get the element by name can be written as:

```
WebElement element = driver.findElement(By.name("firstname"));  
//Action can be performed on Input test element  
Element.sendKeys("ToolsQA");
```

The function to get the element by class name can be written as:

```
WebElement parentElement = driver.findElement(By.className("button"));  
WebElement childElement = parentElement.findElement(By.id("submit"));  
childElement.submit();
```

The function to get the element by tag name:

```
WebElement parentElement = driver.findElement(By.tagName("button"));  
//Action can be performed on Input Button element  
Element.submit();
```

The function to get the element by linktext and partial link text:

```
WebElement parentElement = driver.findElement(By.LinkText("Partial Link Test"));  
Element.clear();  
//Or can be identified as  
WebElement element = driver.findElement(By.partialLinkText("Partial"));  
Element.clear();
```

The function to get the element by xpath:

```
WebElement Username = driver.findElement(By.xpath("Element XPATHEXPRESSION"));  
element.submit();
```

Appendix 5

```
<ObjRep>
<url>http://www.google.com</url>
<search_TxtFld>q</search_TxtFld>
<submt>btnG</submt>
</ObjRep>
```

To retrieve this particular instance:

```
public void objRepository(String eleName){
    try{
        File file=new File("F:\\Test.xml");
        DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
        DocumentBuilder db=dbf.newDocumentBuilder();
        Document doc=db.parse(file);
        doc.getDocumentElement().normalize();
        //System.out.println("The node name is: "+doc.getDocumentElement().getNodeName());

        NodeList nList=doc.getElementsByTagName("ObjRep");
        //System.out.println("The length is: "+nList.getLength());
        for(int i=0; i<nList.getLength(); i++){
            Node nNode=nList.item(i);
            if(nNode.getNodeType()==Node.ELEMENT_NODE){
                Element ele=(Element) nNode;
                System.out.println(ele.getElementsByTagName(eleName).item(i).getTextContent());
            }
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

To utilize the Object Repository:

```
WebDriver d = new FirefoxDriver();
d.get(objRepository(url));
d.findElement(by.name(objRepository(search_TxtFld))).sendKeys("test");
d.findElement(by.name(search_TxtFld(submt))).click();
```

Appendix 6

```
public class ObjectMap {
    Properties prop;
    public ObjectMap (String strPath) {
        prop = new Properties();
        try {
            FileInputStream fis = new FileInputStream(strPath);
            prop.load(fis);
            fis.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

The class contains a single method `getLocator`, which returns a `By` object that is used by the Selenium browser driver object (such as a `HtmlUnitDriver` or a `FirefoxDriver`):

```
public By getLocator(String strElement) throws Exception {
    // retrieve the specified object from the object list
    String locator = prop.getProperty(strElement);

    // extract the locator type and value from the object
    String locatorType = locator.split(":")[0];
    String locatorValue = locator.split(":")[1];

    // for testing and debugging purposes
    System.out.println("Retrieving object of type " + locatorType + " and value " +
        locatorValue + " from the object map");

    // return a instance of the By class based on the type of the locator
    // this By can be used by the browser object in the actual test
    if(locatorType.toLowerCase().equals("id"))
        return By.id(locatorValue);
    else if(locatorType.toLowerCase().equals("name"))
        return By.name(locatorValue);
    else if((locatorType.toLowerCase().equals("classname")) ||
        (locatorType.toLowerCase().equals("class")))
        return By.className(locatorValue);
    else if((locatorType.toLowerCase().equals("tagname")) ||
        (locatorType.toLowerCase().equals("tag")))
        return By.className(locatorValue);
    else if((locatorType.toLowerCase().equals("linktext")) ||
        (locatorType.toLowerCase().equals("link")))
        return By.className(locatorValue);
}
```

```

        return By.linkText(locatorValue);
    else if(locatorType.toLowerCase().equals("partiallinktext"))
        return By.partialLinkText(locatorValue);
    else if((locatorType.toLowerCase().equals("cssselector")) ||
(locatorType.toLowerCase().equals("css")))
        return By.cssSelector(locatorValue);
    else if(locatorType.toLowerCase().equals("xpath"))
        return By.xpath(locatorValue);
    else
        throw new Exception("Unknown locator type " + locatorType + "");
}

```

Getting the objects from the objects map:

```

public static void main (String args[]) {

// Create a new instance of the object map
    ObjectMap objMap = new ObjectMap("objectmap.properties");

// Start a browser driver and navigate to Google
    WebDriver driver = new HtmlUnitDriver();
    driver.get("http://www.bing.com");

// Execute our test
    try {

// Retrieve search text box from object map and type search query
        WebElement element =
driver.findElement(objMap.getLocator("bing.homepage.textbox"));
        element.sendKeys("Alfa Romeo");

// Retrieve search button from object map and click it
        element = driver.findElement(objMap.getLocator("bing.homepage.searchbutton"));
        element.click();

// Retrieve number of search results using results object from object map
        element = driver.findElement(objMap.getLocator("bing.resultspage.results"));
        System.out.println("Search result string: " + element.getText());

// Verify page title
        Assert.assertEquals(driver.getTitle(), "Alfa Romeo - Bing");

```

```
} catch (Exception e) {  
    System.out.println("Error during test execution:\n" + e.toString());  
}  
  
}
```


Appendix 7

1) Right click on 'x' package then select New > Class and name it as "ProductListing_Page".

```
public class ProductListing_Page {  
}
```

2) Create another public static class inside the above class 'x' and name it as Product_1.

```
public class ProductListing_Page {  
    public static class Product_1  
    }  
}
```

3) Now create different Static Methods for each child element of Product_1. These methods will need an Argument (WebDriver) and a Return value (WebElement).

```
package x;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
public class ProductListing_Page {  
  
    public static class Product_1{  
        public static WebElement txt_Price(WebDriver driver){  
            WebElement element = null;  
            // Write Code to find element here  
            return element;  
        }  
  
        public static WebElement txt_SalesPrice(WebDriver driver){  
            WebElement element = null;  
            // Write Code to find element here  
            return element;  
        }  
  
        public static WebElement img_Product(WebDriver driver){  
            WebElement element = null;  
            // Write Code to find element here  
            return element;  
        }  
  
        public static WebElement txt_Name(WebDriver driver){  
            WebElement element = null;  
            // Write Code to find element here  
            return element;  
        }  
    }  
}
```

```
public static WebElement txt_Desc(WebDriver driver){  
    WebElement element = null;  
    // Write Code to find element here  
    return element;  
}
```

```
public static WebElement btn_AddToCart( WebDriver driver){  
    WebElement element = null;  
    // Write Code to find element here  
    return element;  
}  
}  
}
```

This piece of code for Selenium is similar to QTP's intelligent code completion structure.

Appendix 8

Accessing the website URL:

```
Dim URL
Dim IE
Set IE = CreateObject("internetexplorer.application")
URL = "https://www.google.com"
IE.Visible = True
IE.Navigate URL

WScript.Quit Main

Function Main
    Set IE = WScript.CreateObject("InternetExplorer.Application", "IE_")
    IE.Visible = True
    IE.Navigate "http://google.com"

End Function
```

Using VBS to access the web by getelement or by ID:

```
Dim objWshShell,IE,searchStr
Set objWshShell = Wscript.CreateObject("Wscript.Shell")
Set IE = CreateObject("InternetExplorer.Application")
searchStr = "test"
With IE
    .Visible = True
    .Navigate "http://www.google.com"
'Wait for Browser
    Do While .Busy
        WScript.Sleep 100
    Loop
    .document.getElementById("lst-ib").Value = searchStr
End With
objWshShell.SendKeys "{ENTER}"
```

Using VBS to load a web button (return values by click):

```
Dim objWshShell,IE,searchStr
Set objWshShell = Wscript.CreateObject("Wscript.Shell")
Set IE = CreateObject("InternetExplorer.Application")
With IE
    .Visible = True
    .Navigate "https://www.wpi.edu/"
```

```
'Wait for Browser
  Do While .Busy
    WScript.Sleep 100
  Loop
.document.getElementById("search-dropdown-button").Click
End With
```

Using VBS to load a web table:

```
Dim Browser,strOut
Set Browser = CreateObject("InternetExplorer.Application")
With Browser
  .Visible = False
  .Navigate "http://anees.amoeba.co.in/table.html"
'Wait for Browser
  Do While .Busy
    WScript.Sleep 100
  Loop
End With
Set daTable = Browser.Document.getElementById("daTable")
strOut = ""
For i = 0 To daTable.rows.length - 1
  For j = 0 To daTable.rows.item(i).cells.length - 1
    strOut = strOut & daTable.rows.item(i).cells.item(j).innerText & Chr(9)
  Next
  strOut = strOut & Chr(13) & Chr(10)
Next
Browser.Quit
Browser = Null
Msgbox "Table Data: " & Chr(13) & Chr(10) & strOut
```

Appendix 9

Code to convert object repository into XML format:

```
Creating the ObjectRepositoryUtil to work with OR  
Set objORU = CreateObject("Mercury.ObjectRepositoryUtil")
```

Exporting the OR into new XML file:

```
objORU.ExportToXML"C:\Users\TestMe.tsr","C:\Users\TestMe.XML"
```

```
'Destroying the object  
Set objORU = Nothing  
how to add the object by using VBScript  
Set myRepository = CreateObject("Mercury.ObjectRepositoryUtil")  
myRepository.Load "C:\QuickTest\Tests\Flights.tsr"  
myRepository.AddObject myLink, Browser("B").Page("P"), "myLinkName"
```

Another way that can be used to get the object from XML file, we use the following functions:

```
Set objExcel = GetObject( , "Excel.Application")
```

```
For Each objWorkbook In objExcel.Workbooks  
    WScript.Echo objWorkbook.Name  
Next
```

```
ObjExcel.Quit
```

Notice the comma before 'Excel.Application'. The script gets a reference to a running Excel application, lists open workbooks names and quits Excel.