

9-2016

A Comparative Study of Automated Software Testing Tools

Nazia Islam

Nazia Islam, isna1301@stcloudstate.edu

Follow this and additional works at: https://repository.stcloudstate.edu/csit_etds



Part of the [Computer Sciences Commons](#)

Recommended Citation

Islam, Nazia, "A Comparative Study of Automated Software Testing Tools" (2016). *Culminating Projects in Computer Science and Information Technology*. 12.

https://repository.stcloudstate.edu/csit_etds/12

This Starred Paper is brought to you for free and open access by the Department of Computer Science and Information Technology at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Computer Science and Information Technology by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

A Comparative Study of Automated Software Testing Tools

by

Nazia Islam

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Computer Science

September, 2016

Starred Paper Committee:

Dr. Jie Meichsner, Chairperson

Dr. Donald Hammes

Dr. Susantha Herath

Abstract

Software testing is an integral phase in Software Development Life Cycle (SDLC) process. Testing assesses the functionalities of a software item and quality of the product. Automated software testing utilizes different tools to execute testing activities. In this paper, I have discussed the features of automated and manual testing as well as analyzed three automated software testing tools: Selenium, UFT/QTP and WATIR. In brief, I have presented a detailed description focusing on multiple feature set, efficiency, simplicity and usability of each tool. I also evaluated, tested and compared the different aspects of Selenium, UFT/QTP and WATIR. Finally, this research allowed me to draw some solid differences between automated and manual testing as well as learn and explore various characteristics of automated testing tools by having real-world experience of testing effectively.

Table of Contents

List of Figures	5
List of Tables	6
Chapter 1: INTRODUCTION.....	7
1.1 Software Testing	8
1.2 Objective of Research	9
1.3 Terminologies.....	10
Chapter 2: BACKGROUND AND RELATED WORK	12
2.1 Related Work.....	12
2.2 Software Testing Techniques	13
2.3 Software Testing Tools	19
2.4 Selenium.....	21
2.5 QTP/ UFT.....	29
2.6 WATIR.....	33
Chapter 3: METHODOLOGY	38
3.1 Selected Tools	38
3.2 Evaluation Metrics	38
3.3 Target Application.....	40
3.4 General Testing Approach	40
Chapter 4: TESTING AND RESULT ANALYSIS.....	41

	4
4.1 Test cases.....	41
4.2 Method used to locate HTML element	44
4.3 Test using Selenium IDE:	47
4.4 Test using Selenium Webdriver	54
4.5 Comparison between Selenium IDE and Selenium Webdriver	63
4.6 Test using UFT	65
4.7 Test using WATIR Webdriver	72
4.8 Comparison among Selenium, WATIR, and UFT/QTP	78
Chapter 5: CONCLUSION	83
REFERENCES	86

List of Figures

Figure-1: HTML element locating procedure on web page.....	44
Figure-2: HTML source code on e-services login page.....	47
Figure-3: Selenium IDE executing command on course search page	50
Figure-4: Selenium IDE test fails for not matching actual value on page with test input value ..	51
Figure-5: Test case passes on updating the target alert message on Selenium IDE.....	52
Figure-6: When all the test cases pass from login to logout	53
Figure-8: Selenium Webdriver test script written in java eclipse IDE_part2	59
Figure-9: Error during executing test in Selenium Webdriver	60
Figure-10: UFT test script-1	68
Figure-11: UFT test script-2	69
Figure-12: Test result after executing test in UFT	70
Figure-13: Start of test scripting with WATIR Webdriver in Ruby command prompt	75
Figure-14: WATIR Webdriver test scripts.....	76
Figure-15: Error while testing with WATIR Webdriver	77

List of Tables

Table 1: Manual vs. Automated Testing.....	17
Table 2: Basic differences between QTP and UFT	30
Table 3: Features of the first Machine	39
Table 4: List of Test cases	41
Table 5: Selenium IDE Test script.....	48
Table 6: Selenium IDE vs. Selenium Webdriver.....	62
Table 7: Features of the second Machine	70
Table 8: Comparison among Selenium-WATIR-UFT/QTP.....	77

Chapter 1: INTRODUCTION

Software end users are more informed and demanding than before. The quality of software determines the success of any software product. This provides a tremendous opportunity for software quality assurance in software industries and that is driven by the customer satisfaction. Developing quality and defect-free products under time and budget constraints have become crucial. To implement such products, with minimum or no error is very difficult, that's why the idea of software testing has come into existence[1]. Software testing has become an essential and extensive activity in the software industry. Testing is the critical part of software development process and indicates the eventual review of the specification, design, and coding[2]. Nowadays, probably no single company exists without performing software testing.

Software testing is a method, which is executed for evaluating the functionality and the correctness of the software product, to determine whether it meets the expected features and quality. Testing is an essential phase and an inevitable part of Software Development Life Cycle (SDLC). A different compound definition of software testing states that “the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior”[3]. IEEE provides the definition of software testing as, “the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or identifies the difference between expected result and actual result” [4]. So, software testing means executing a software program or system to detect any bugs or missing functionality which differs from the expected result or requirements [5].

If appropriately executed, testing improves the performance or accuracy of a software system.

Locating the defects in software and making the corrections before releasing the software

product, helps business to save their extra maintenance cost.

All the software testing activities are executed in two ways: manual testing and automation testing. Manual testing is the elementary software testing; it is performed manually by navigating into the software application, following a test plan or test cases (overall test scenario in the form of actions needed to perform while testing). However, in automated testing, the testing activity can be performed utilizing different testing tools without the need for looking at different parts of the application manually.

Previously, software testing was executed merely by manual testers. But by manual testing, some errors can be easily ignored or some remain uncovered due to human errors. Thus, it has become almost impossible to test any software program/application manually in order to gain the highest level of accuracy. Automated testing has been developed to overcome the deficiency of manual testing [6]. The evolution of automation testing has made the testing process much faster and nowadays it has become more popular and is the preferred method of testing applications and websites [7]. The base of automation test is the tools used to perform the testing. In this paper, different aspects of three popular automation testing tools have been studied and analyzed to evaluate the tools.

1.1 Software Testing

One very general conception is that software testing is needed to find the bugs. In reality, there could be various reasons for conducting tests. One of the most important reasons is to improve the quality of applications by making sure the software is meeting all the requirements as designed and working as expected.

Software testing is necessary to ensure whether the application functions smoothly. It is more important for the web based applications, as testing prevents the application from having down

time [8]. Most of the industries spend around 50 percent of their total time and more than 50 percent of the total cost towards testing during the software development life cycle [9]. So, performing the test early during the development process helps to prevent the occurrence of unnecessary bugs or unwanted changes during the maintenance phase. If the bugs or defects are found later by the end users after releasing the software product then the testing and the maintenance cost goes significantly higher. So, if the testing is done on a regular basis during the SDLC process, it is a good way to verify how many defects the application product contain and the level of risk is associated with this product. For the companies, it is much cheaper to fix a bug if it's caught before releasing the product [10].

Software testing intends to assess the ability of an application or program and determines whether it meets the quality criteria or not. There are different specifications, which need to be established while developing a software or application, such as portability, reliability, security, efficiency, usability etc. All these are also required to be validated and accomplished through testing. [11].

Finally, there are also three main purposes of software testing, which focuses on the following:

Detection: Detection of errors or defects during testing.

Prevention: Preventing or shrinking the number of errors in the system to improve the overall efficiency and performance of the system.

Demonstration: It demonstrates how a system can be managed to run with several risks which are small and acceptable.

1.2 Objective of Research

The goal of this research paper is to introduce the various features and presentations of software testing tools (Selenium, WATIR, and UFT/QTP) as well as assess and compare these tools to

determine their usefulness. This study also allows drawing the basic comparison among automated and manual testing to represent the significance of automated testing in software industries. To accomplish the goal of this research, the following steps are to be performed.

- Identifying a set of tools to be evaluated.
- Selecting the target application to be tested by all these tools.
- Testing the target application using the selected automated testing tools and gather resulting data.
- Developing a set of comparisons to be used to assess the tools.
- Performing an analysis of each tool and comparing each other based on an ideal feature set and depending on the result of test execution.
- Outlining inferences and making recommendations based on the outcome of research.

The web application to be tested for this research is the students' E-services website for St Cloud State University. The main focus of the testing will be the course registration process.

1.3 Terminologies

This section describes the different terms used in this paper.

Expected result: The standard test result, which is defined after requirement analysis and prior testing.

Actual result: The real result of testing, after executing the test with input data to the software [4].

Test case: It is a manuscript that provides a process to perform testing.

Each test case is an action, a combination of an expected result, actual result and pass or fail criteria. [8].

Test suite: It is a collection of test cases.

Test Script: Test scripts are usually used for automated testing. It's a program written to execute testing of the application.

Test Plan: In the industries, a test plan is a document, which defines the scope of testing, test cases, test results and any test related activities.

Regression testing: When any changes occur to software/application, then it's required to verify that all the previous functionalities work accurately with the newly added functionality. This process of testing is called regression test.

Functional testing: Functional testing of software is done when the whole system is ready to check the system's performance with respect to its specified requirements [9]. It verifies that the features or functionalities of the application are working as defined.

Load test: Load testing is executed to evaluate the system's performance with various amount of load applied (for example usual load and peak load).

Unit test: In unit testing, each unit of source code (classes, functions etc.) are tested to verify whether all the individual parts of the program are working appropriately. The script and the execution of unit tests are done by programmers.

Chapter 2: BACKGROUND AND RELATED WORK

The first part of this chapter mainly focuses on the related works performed in the area of the automated software testing tools. It summarizes some of the researched works, with emphasis on Selenium, QTP/UFT, and WATIR. The rest of this chapter describes the different software testing techniques and explains various aspects of the automated testing tools used for this research.

2.1 Related Work

The research of Jagannatha and Niranjana Murthy, emphasizes the different components of Selenium such as Selenium IDE, Selenium RC, Selenium WebDriver and Selenium Grid. It also focuses on the need for the Selenium testing framework, most commonly used commands, and makes the comparison with the QTP testing tool [10]. However, this research concentrates more on Selenium, it does not discuss much QTP. The study here is more focused on Selenium, which is also mentioned as the best tool for automated testing of websites.

The research performed by Vishawjyoti and Sachin Sharma [11], focuses mainly on Selenium IDE, a component of Selenium. The research contains set of test cases and properly mentions the way of recording test cases in IDE. However, the study is just confined to Selenium IDE.

Research conducted by Nisha Gogna [12], presented the basic features of the automation tool: WATIR and Selenium. Gogna mentioned that Frames and pop-ups are accessible using WATIR. However, Selenium requires the user to have advanced language skills in order to test for frames and pop-ups. This is indeed a good paper to learn about Selenium and WATIR, but lacks in the area of comparison between these two tools.

Harpreet Kaur and Gagan Gupta [13] showed a comparative study among Selenium, Test Complete, and QTP tools. That study includes different aspects but does not provide the automation features, such as record and play-back, cross-platform or browsers support features.

Rigzin Angmo and Monika Sharma [14] conducted a performance evaluation of Selenium suite with WATIR Webdriver. Although that research covers most of the comparison criteria, the one thing that's absent in the research is comparisons with any commercial testing tool. There would be more cases to compare while bringing both open source and commercial tools into consideration.

T. J. Naidu and N. A. Basri conducted a research, [15] where both the tools are open source (Selenium and SAHI). Finally, they mentioned, Selenium is aimed to be best for programmers while SAHI is aimed at non-programmer testers.

In the research conducted by Abha Jain and Manish Jain, the different features of QTP and Ranorex have been analyzed and compared. It's a good research to follow but it lacked comparison with any open source tools.[16]

Another research was conducted by Meenu and Yogesh Kumar, where the comparison was made among UFT/QTP, Selenium, Test Complete and SoapUI. All these tools were compared to a number of features and based on the comparison it declared SoapUI is the best tool [17].

However, SoapUI is a commercial tool, does not provide support for multiple languages, operating systems, and windows applications.

2.2 Software Testing Techniques

There are different traditional methods used to perform software testing. Here the most common techniques are described:

Black Box Testing

The core idea behind Black box testing is that the tester does not need to know the internal structure of code or details of how the whole program is built. In this case, the tester only works with the interface of the program. As the name indicates, the tester does not have vision or

knowledge of implementation. The goal of this testing is to verify that how accurate the application or the program works with the specified requirement or set of inputs as well as produces correct outputs. The tester does not necessarily need to have much programming knowledge or internal logic of code but needs to know the expected output of the program[18].

White Box Testing

White box (also called, clear box) testing is completely opposite of the black box testing. In this testing, tester requires having some level of programming skills. The internal structure of the system, as well as details of code, should be clearly understood by the tester. As most of the software defects or bugs are caught and also resolved during white box testing (during the unit test), it's a very effective way of testing. Here, the tester does not need to implement the program but should have programming knowledge to find the bug or anomaly within the code. A good example for white box testing is “unit testing”, where the code developed for a particular module, that needs to be tested prior to integrating it with rest of the modules. Thus, unit testing is very efficient in minimizing overall bugs in a system. Test- driven development (TDD) utilizes the white box testing technique[19].

Manual Testing

In manual testing, the basic level of testing, the tests are performed following the test cases and directly interacting with the application. In this testing, the tester prepares the test cases. Test cases are written in simple English language, which illustrates the features or functionalities to be tested and the expected result. As the tester needs to perform all these activities manually, the whole process of testing can become too lengthy. However, for some particular complex system or application manual testing is preferable and effective since some critical and rare bugs are discovered during manual testing. The tester for manual testing a plays similar role to an end

user of that software application and explores the different parts of the application by testing and making sure about the correctness of application[13]. Some of the drawbacks for manual testing are [16]

- Time inefficient and labor intensive.
- Very flat learning curve.
- Lacking the advantage of reusability.
- Not an iterative process, or multiple iterations do not necessarily provide better accuracy[20].
- Manual tests provide limited visibility, as the tester does not have much knowledge of how the code works.
- Tests have to be repeated by the software developer, tester and finally by the business analyst (to verify that the User Interface is working properly).
- No validation process to verify that the test is actually performed correctly; test cases are manually written whether the test passes or fails.

Automated Testing

With the boom of test automation within the industries, testing has become very efficient. Test automation eliminates the burden of user initiation and difficulty of performing various types of testing such as regression and load/ performance testing. With the advancement of automated testing, complex testing tasks became much easier than before, as it allows performing the test with numerous sets of data and multiple times without intervention of human [20].

Automation testing requires some initial investment for the software and compatible hardware resources but could potentially be more economical since it reduces the human efforts in testing. [21]. The process of automated testing can be conducted in different stages. But in general, can be divided into the four basic ways:

1. Preparing the test plan or creating the test cases
2. Preliminary selection of testing tool
3. Writing/generating the test script
4. Performing the test using automated test scripts

Objectives of Automation

The main principle of automated testing is to minimize the testing effort and time. It also improves the efficiency, while reducing the direct involvement of humans in executing test as well as generating test results. Test automation provides the reusability of codes, by using the same scripts for multiple times, only by changing the input set, as needed[20]. Automation testing also reduces the future maintenance cost of the application, as it simplifies the test process of regression testing, thus more testing can be performed during different SDLC stages and less maintenance cost are issued for post-maintenance phase[11]. Some example where automation can be beneficial are

Simplifying Regression testing: In regression testing, when new releases/ bug fixes take place it needs to be assured that the newly added changes or bug fixes have not introduced new bugs in the system and it is functioning properly with all the existing features. Hence, this test needs to be performed regularly. It becomes very effective for company cost, resources and time if automation is used.

Executing same tests multiple times: When the test cases are needed to be executed a multiple number of times with different input sets, automated testing is very efficient in that case, by executing test scripts.

Time and budget constrain: Test automation saves time and energy of tester, so in this case, the tester gets additional time to engage in other activity. Automation is a very productive and profitable idea for industries nowadays. As most of the time tester has knowledge of

programming and can start working with the tool immediately. So automation is cost-effective for companies since they don't need to spend extra money in training the employees and it also saves time.

Load or performance testing: While testing the load handling ability of an application, at least many virtual test user accounts need to be created and tested simultaneously. Using only manual testing this job seems too critical. However, with the help of automated testing this difficult testing can be performed effectively.

Frequently changing functionalities: Test automation works best in the situation, where the requirements are frequently changing and complex test cases are needed to be executed repeatedly. For any application, where the GUI does not change much but the code changes frequently, test automation can be more effective in that case [22].

Comparison between Manual and Automated testing

Table 1: Manual vs. Automated Testing

Manual testing	Automated testing
1. The process of manual testing is slow and lengthy, which takes a long time to complete the test.	1. Automated tests can be performed in a faster way than manual tests; because once the test scripts are generated, it can be executed any number of times.
2. The manual test requires vast human effort. As the testing is time consuming, it requires more testers to accomplish the task.	2. Automated tests need one-time effort to write the test scripts, as it can be executed without having human effort. So, it requires fewer investments for testers.

Table 1 continued.

Manual testing	Automated testing
3. It cannot be perfect or fully reliable test, as testing is done manually by testers and no one can avoid the human errors.	3. It can be performed using automated tools, so there's very little chance of mistakes while executing the test.
4. A manual test is not good for executing bulk amount of test cases.	4. Test automation is good when the test suite is huge.
5. Manual testing is good for functional tests and exploratory tests (where the testing is performed by discovering the various features of application in order to find bugs).	5. Test automation is well suited for regression test and almost all kinds of non-functional testing, such as Load test, performance test, which are very difficult to execute with manual tests.
6. Manual testing can only avoid the cost of automation tools.	6. With automated testing, the recurring and bigger costs can be avoided. For example, maintenance cost and cost of manual labors.
7. Most of the time new or critical bugs can be found by executing manual tests only	7. Automated tests do not help much in finding new bugs, as the same scripts are run a number of times.
8. In manual testing there always a tester needed to perform all the test steps manually.	8. Test scripts can be run automatically without the need of the tester to be present in front of a computer during the time of test execution.

2.3 Software Testing Tools

A testing tool needs to be selected in order to start the testing. As there are a huge variety of testing tools available, selecting the proper tool does not depend on just a single or a couple of things. While selecting the testing tools many things need to be analyzed in order to make sure how much the intended tool is able to meet the expectations. Some important parameters for selecting appropriate testing tool have been outlined here [14]

- Type of application to be tested (windows/web/ mobile)
- The type of testing (i.e. regression, unit, load, performance etc.).
- The whole test scenario or testing scope needs to be analyzed
- Associated cost to provide training for employees.
- The cost of the tool itself, when it's a commercial one, it's necessary to buy licenses for the testers.

Other important factors are reusability, reliability, and cost. All the factors need to be considered to obtain maximum benefit by utilizing the tool for testing. Additional features of the testing tool also need attention (i.e. Ability to perform, record and run, providing test results), version of the automated tool as well as the important parameters associated with the tools, including the following[23]:

- Type of testing that the tool supports
- Different available features within the tool
- Associated cost for licensing (if commercial)
- Support for browsers and Operating system and programming languages
- Easy to work with (provides easy execution of tests)
- Integration with other test management tools

All these criteria need to be verified before finalizing the tool. So that the maximum utilization of the testing tool can be made.

There are many commercial or open source tools available, among them, some of the general categories of tools are listed below:

- 1. Unit testing tools:** Unit testing validates that individual units of source code are working properly. Few good examples for unit testing frameworks are Junit, TesNG, both of these are based on Java programming language.
- 2. Functional testing tools:** These are the tools that allow functional testing (making sure if the application is working as expected). With the help of the tool, automated scripts can be generated in order to handle functionality changes in the application. Some of the functional testing tools are also able to perform regression testing. Here are some widely used functional testing tools: Selenium, WATIR, UFT/QTP, Sahi [24].
- 3. Load testing tools:** It is testing the performance of the application. Some of the load testing tools are JMeter, HP LoadRunner[24][23].

All the above criteria are applicable while selecting tools for an industrial purpose. However, for researching purpose, one can select any tool, depending on the availability of the testing materials and resources. For this paper, I have chosen to work with Selenium, UFT/ QTP, and WATIR for my research purpose. Nowadays Selenium is one of the most powerful and leading functional testing tool for automation. Most of the industries are using either Selenium or UFT/QTP. UFT/QTP also works with windows based desktop applications but it is a commercial tool. On the other hand, WATIR is not as popular as Selenium or QTP/UFT but it's a good tool to work with Ruby.

2.4 Selenium

Selenium is an open source test automation tool, which supports different types of testing in web applications. Selenium is not just a single tool but it consists of four tools: Selenium IDE, Selenium RC, Selenium WebDriver and Selenium Grid. Selenium provides the ability to create test scripts in different programming languages and the ability to perform different kinds of testing, such as functional, regression test, etc. Selenium can be also integrated with various frameworks to provide a hybrid framework, which makes the testing simpler[14].

2.4.1 Brief History of Selenium Project

Selenium first came into existence on 2004, by Jason Huggins. He implemented a JavaScript library, which was able to drive the webpage automatically as well as able to run tests against multiple browsers. That library ultimately became Selenium Core, which generated the functionality of Selenium Remote Control (RC) and Selenium IDE. Earlier there was no other tools, which could perform the test automation activities in multiple browsers and programming languages, except for Selenium RC. Although Selenium was a marvelous tool, it also had some limitations. For example, it was a JavaScript based automation tool and because of security restrictions of browsers in supporting JavaScript, Selenium RC was allowed to perform only a limited number of functionalities or actions [25].

Later, the Selenium and WebDriver, another tool, merged together and built the Selenium web driver. The combination of both of these tools provided a set of great features and functionalities.

2.4.2 Selenium IDE

The Selenium IDE (Integrated Development Environment) is a plug-in for the Firefox browser, which is used to generate test scripts. It's the simplest tool in the Selenium package and provides an easier way to automate the tests, using the recording and playing back feature. The feature

provides a Graphical User Interface (GUI) to record user's actions. Unlike other Selenium tools, Selenium IDE does not need any programming language skills to execute testing. Selenium IDE has its own command language it uses for testing, called Selenese[25].

It's a simple tool to use for automation. It has limited functionalities and most importantly it does not support writing test code and supports only specific browsers (Mozilla Firefox). To overcome all these issues and getting efficient testing support, Selenium RC or Webdriver can be used.

2.4.3 Commonly Used Selenium IDE Commands

There are many commands in Selenium IDE. Below are some of the basic and widely used Selenium commands.

- **open:** Starts browsing by opening a page with URL.
- **type :** Sends text input to an element
- **click/ click and wait:** performs a click operation; Clicks and waits for a new page to load.
- **verify:** Performs a soft assertion against an expected value.
 - o **verifyTitle/assertTitle:** verifies an expected page title.
 - o **verifyTextPresent:** verifies expected text is somewhere on the page.
- **select() :** This command is used to select a label from a drop down box or a combo box.
- **check () :** This command will check the box when there is a checkbox on the testing webpage[26].
- **waitFor** – Waits until the specified element is found on the webpage or the timeout is reached.

2.4.4 Selenium RC

Selenium Remote Control (RC) is used to create test scripts for User Interface (UI) testing in various programming languages, such as Java, C#, PHP, Python, Ruby, and PERL. Selenium RC runs tests inside JavaScript web browser, it is now available on all web browsers.

Selenium RC has two basic components:

1. RC Server - The RC server bundles Selenium Core, which is a set of JavaScript codes that control the browser, and automatically loads it into the browser. It performs as an HTTP proxy, which verifies any HTTP messages passed between browsers and the application.
2. RC Client – Provides an interface between the programming language and server.

It also provides an alternative as well as a better solution for users of Selenium IDE [27].

However, Selenium RC struggles while executing simultaneous tests, as it works slowly. There are also many complex features of Selenium RC, which made RC deprecated after the development of Selenium Webdriver.

2.4.4 Selenium Grid

The core idea of Selenium grid is running multiple tests concurrently across different browsers, operating systems, and machines. Grid uses a hub-node concept, where the test is run in a central machine called a 'hub' and the parallel execution of the test is conducted in different remote machines, called nodes. Grid has two basic versions, such as Grid 1 (older) and Grid 2 (newer); Grid 1 is capable of supporting Selenium RC commands only, but Grid 2 can support both RC and Webdriver commands.

Since Grid 1 was lacking competent features and configurations, Grid 2 came into existence by adding the latest, convenient features as well as correcting the issues with grid 1.

One of the enhancements in Selenium grid 2 is, with one remote control (RC), it can automate up to 5 browsers ,where grid 1 could automate only 1 browser per RC [28].

Due to its complexities and running with only limited features, grid 1 has been gradually deprecated.

2.4.5 Selenium Webdriver

Selenium Webdriver is the most widely used tool within the Selenium package. Selenium Webdriver provides a simpler, more concise programming interface, which addresses most of the issues of the Selenium-RC API [29]. Selenium Webdriver supports many more powerful features which are not supported in other Selenium tools [30].

Advantages of Selenium Web Driver

1. Multiple Web Browsers support

It provides an option to execute test scripts against different web browsers such as (IE, Chrome, Firefox, Opera, Safari), to perform testing.

2. Variety of Programming Languages

Selenium Webdriver also allows writing scripts in Java, Python, C#, Perl, Ruby and PHP. So, the tester can choose any of the programming languages.

3. Multiple Testing Frameworks

With the features of Selenium, different other frameworks can be combined to make hybrid and enriched framework. For source code compilation, Selenium provides Maven, Ant framework; it also provides TestNG for unit, functional testing and report generation.

4. Defect management

Selenium Webdriver provides defect management with the help of Jenkins framework by allowing users to enter bugs into JIRA (Defect management tool).

5. Free of cost

One of the greatest advantages of Selenium is, it's free. The only cost associated with Selenium testing is that the companies using Selenium as a primary testing tool might require training the employees for the first time. [31][32].

Finally, Selenium Webdriver remediates many complex features of Selenium RC, some of the major issues are below:

Webdriver classes are better organized and offer a cleaner API than Selenium RC, which provides great support for web application testing.

Unlike Selenium RC, Webdriver testing does not require to start the server for executing test scripts[30].

Webdriver is faster than Selenium RC and uses the browser's own engine to control it. It interacts with web page elements in a more realistic way. For example, Selenium RC uses the command as 'Selenium.type' and 'Selenium.typeKeys' and both of the commands perform the same thing (typing text in the textbox), wherein Webdriver uses 'sendKeys' for type related commands.

Selenium RC works using JavaScript injection, which can be also used for hacking purpose (Directly interacting with live web application from client side). Selenium Webdriver overcomes this issue by using a different driver for every web browser. For firefox browser, Webdriver uses firefox driver, for IE it uses IE driver and for chrome it uses the chrome driver.

For all these above-mentioned issues, Webdriver executes the test in a faster way.

Webdriver also supports testing mobile devices such as iPhone, iPad, and android phones as well as tablets[33].

Disadvantages of Selenium [32]

1. Supports only web application testing:

Selenium supports web applications well enough but it doesn't support windows based applications.

2. Expertise required in programming language

As Selenium supports a variety of programming languages, it becomes easier for the tester to write scripts in his preferred programming language. However, the professional must have adequate expertise in the specific programming language to write test scripts.

3. Voluntary Assistance is required for Selenium Community

While using Selenium, testing professionals will need support and assistance to handle technical issues. As Selenium is an open source technology, users (who are good programmers) provide additional time to help of the Selenium community forums to resolve the technical issues.

4. Additional Tools Required to Generate Reports [35]

Despite being effective in testing web applications comprehensively, Selenium still lacks inbuilt reporting capabilities. Testing professionals have to use additional tools to generate test reports while testing with Selenium. Additional framework or plug-ins like JUnit or TestNG are used to generate test reports in Selenium.

5. Captcha and Bar code readers cannot be tested using Selenium.

In spite of all these limitations, Selenium is still effective in reducing the test cycles drastically. The reduced release cycle further help software companies in reducing the overall project cost. [32].

2.4.6 Basics of Selenium Webdriver scripting:

Selenium Webdriver has the default driver as Firefox (Mozilla Firefox browser). Following are some of the basic commands used in Webdriver scripting.

1. Creating New Instance of Firefox browser

```
Webdriver driver = new FirefoxDriver();
```

2. Open the expected URL in Browser

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

3. Get the page title

```
driver.getTitle();
```

“getTitle()”, returns the title of the web page as a string.

4. Clicking on any element or button of webpage

```
driver.findElement(By.id("abc.")).click ();
```

In this example, locator “id” is used to find the HTML element on the web page.

5. Typing text in textbox area

```
driver.findElement(By.name("search")).sendKeys(" Name");
```

This syntax finds a web element with textbox and types text in that area using 'sendKeys'.

6. Shows the URL for current web page:

```
driver.getCurrentUrl();
```

This command displays the actual URL (current) of webpage [34].

7. Ending Browser Session

```
driver.close(); [35]
```

Selenium Webdriver uses “close()” to close the browser session.

The following section, briefly describes the different web elements, which are used to locate HTML elements in web page.

2.4.7 Locators in Webdriver:

In Selenium Webdriver, everything is related to web elements as it is a web application automation tool. Web elements are presented on the web page as HTML elements. To perform operations on a web element we need to locate the elements exactly. When a web element is located, the tool continues to execute the script but if the locator type is not the correct, then throws an exception. The command for locating elements using Webdriver starts with “*findElement*”.

driver.findElement (By. <Locator>);

In the above statement, “By” is the class, where different static methods can identify the element. As there are different type of locators in HTML webpage, it needs to specify the “Locator” to identify web element, as following –

- a) **id** – Locating any element using “id”, is the most preferred and effective way. This type of locator is more explicit, in this case, the testing tool considers the first element with matching id attribute. Usually, ids are unique.

findElement(By.id(“someId”))

- b) **name**- *findElement(By.name(“someName”))*

Locator ‘name’ will locate the first element with a matching name attribute. If several elements exist on that web page with the same name attribute, it’s better to choose some other type of locator.

- c) **XPath**- When there's no other way to uniquely identify an appropriate id or name attribute for the element that needs to be located, then XPath locator is used. There are two types of XPath, absolute XPath and relative XPath. All XPath locators start with "/"
- The example below is taken from the testing.

```
findElement(By.xpath("//html/body/div/table/tbody/tr/td[2]")), [XPath is discussed in chapter -4]
```

d) **linkText**

This locates hyperlinks in the web page by using the text of a link.

```
findElement(By.linkText("Google"))
```

- e) **By.partialLinkText** - Locates the link element with partial matching visible text

```
findElement(By.partialLinkText("Goo"))[35]
```

Above, I captured some of the basic commands in Selenium WebDriver. As I have used Java, the commands are actually following general Java programming syntax and styles.

2.5 QTP/ UFT

Quick Test Professional (QTP) or Unified Functional Testing (UFT) both are automated functional testing tools provided by Hewlett Packard (HP). HP developed QTP in 2006 and UFT was released first in 2012. UFT is actually the latest version of QTP. QTP is the graphical user interface (GUI) record-playback automation tool. QTP/UFT enables testing in standard windows applications and web applications, where the web applications consist of web objects, ActiveX controls, Java applets, Visual Basic applications, .NET framework applications and multimedia objects. QTP/UFT uses Visual Basic scripting language for writing tests [19].

UFT/QTP facilitates generating tests scripts by recording operations. When navigating through the application, UFT/QTP records each step and creates the test script. Once a test is generated,

it's possible to make some amendments in the test script. After running a test, a resultant report is generated showing the overall test pass/fail with details of which steps the test succeeded or failed[36]. QTP/ UFT tests can be created in another way without recording, which is creating the test scripts by writing Visual Basic (VB Script) code.

Table 2: Basic differences between QTP and UFT

Criteria	UFT	QTP
Type of Testing	Provides both GUI and API (service) testing.	Provides automated GUI testing only [38].
Browser Support	All the latest versions of Internet Explorer till 11.0 and few version of Google Chrome and Firefox	Supports up to Internet Explorer version 9 [7].
Operating System	Windows XP, Vista, 2003, 7, 8, 8.1, 10 and Windows Server 2008/Windows Server 2012	Windows XP/2003/Vista/7 and Windows Server 2008 [23]

How does UFT/QTP locate elements in an application or webpage?

UFT or QTP recognizes each element on the web page as an object and it uses Object Repository (OR) that stores the objects information in QTP or UFT. Object repository works as an interface between test script and test application, which identifies objects during testing.[37].

2.5.1 Features of QTP / UFT

- Recording Efficiency & Playback of the scripts

UFT/QTP provides the facility of recording the test and generating test scripts, where the script can be executed multiple times for testing the application.

- Ease of Use: Since UFT/QTP provides the recording interface and playback the recorded script; it turn out to be easier to learn. Recording the test script and playing it a number of times

does not require deep knowledge of Visual Basic scripting. Each line of the test script is clear and understandable to users [38].

- **Better Object Identification Mechanism:** One of the greatest features of UFT/ QTP is the object repository for identifying objects. The well-defined object repository stores the information of all objects being used in the script.

- **Language support:** UFT/QTP both provides the option to record test and create code for testing in Visual Basic (VB) scripting language.

- **Browser Support:** UFT/QTP supports Internet Explorer (IE) browser version up to 9 only, where UFT supports latest IE browser version 10, 11 as well.

- **Use of Keyword Driven testing:** UFT/QTP provides the keyword driven testing technology. In keyword driven testing, there are set of actions as well as logic to read keywords are created in excel document. Then the UFT/QTP driver executes the test by connecting with excel document. Once, this framework is created, the test becomes faster and easy to execute.

- **Data-driven testing:** The idea behind data-driven testing is executing/reusing the test script with multiple data sets. The input data is stored usually in MS Excel, Access or XML files. Therefore, the test script can extract input parameter from the files. This method of data driven testing is offered by QTP/UFT, which is an efficient way of using test scripts with multiple sets of input data.

- **Support of Variety of Environments:** UFT/QTP supports various add-ins for functional testing, such as Windows, Web, .Net, Visual Basic, ActiveX, Java, SAP, Siebel, Oracle, PeopleSoft, and Web services [38].

- **Test run result reports:** Once the test is executed, UFT/ QTP provides a summary of test execution and a final pass/ fail result. [39].

Advantages of Using QTP/UFT

- It provides an option to test both windows and web applications.
- QTP/ UFT can be integrated easily with test management tools like Quality Center (QC).
- QTP/UFT provides better support or services for any technical issues related to the testing tool because it's a commercial tool [31].

Disadvantages of QTP/UFT

Following are some of the limitations of QTP/UFT, for not using as a primary testing tool in the companies.

- **Scripting Language:** It only allows creating the test with Visual Basic scripting and that's not an object-oriented language.
- **Limitations with browser and OS support:** QTP is not compatible with many of the latest browser types and versions as well as Operating systems. Only UFT supports the latest Internet Explorer browsers and Windows.
- **High licensing and add-ins Cost:** QTP/UFT is remarkably costly. There is not only the licensing cost but also cost is associated with upgrading to the latest features and technologies [38].

Worst case is possible. If the QTP is not upgraded and the associated platforms (OS and browsers) are upgraded to latest version, then it becomes quite impossible to use of the QTP for test automation. That's why UFT came into existence; it supports some of the latest technologies, which QTP does not. Still, the organization needs to upgrade the UFT version, at the same time they upgrade their other applications associated with UFT [31].

2.6 WATIR

WATIR, which means Web Application Testing in Ruby (pronounced water). It is developed to perform the test for automating the web browsers using object oriented-scripting language Ruby.

WATIR consists of several smaller components. Among them the most commonly used are WATIR classic and WATIR web driver.

WATIR supports a number of browsers, such as Internet Explorer, Firefox, Chrome, and Opera. It has been used for functional testing, regression, and system testing.

WATIR Classic

This is the original WATIR that drives Internet Explorer.

Brief Working Principle

WATIR makes the proper use of Ruby's Object Linking and Embedding (OLE) capabilities, which is a built-in feature of Ruby and developed over Component Object Model (COM) architecture. It makes possible to drive the Internet Explorer. COM allows the communication between Ruby and Internet Explorer which is also called inter-process communication. COM also permits the dynamic object creation and then manipulation. That's how Ruby handles the browser[40] [16].

WATIR Webdriver

WATIR-Webdriver provides a clean syntax and was inspired by Selenium Webdriver frameworks. It has similar frameworks in other languages (Watij for Java and Watin for .NET/C#). It allows driving of many web browsers – such as Chrome, Internet Explorer, Safari and Firefox[41]. It has control over the webpages, which are built in HTML and JavaScript [42].

The WATIR-Webdriver is also called WATIR 2 as it combines WATIR (classic), Webdriver as well as some additional features[43].

Basic commands in WATIR Webdriver:

The following section outlines some of the commonly used WATIR Webdriver commands.

- *This command is essential to get started with WATIR Webdriver.*

require 'WATIR-Webdriver'

- Starting a new browser using Firefox web browser

browser = WATIR::Browser.new :firefox

-For opening new session to a specific site (here, it's google)

browser.goto ("http://www.google.com")

WATIR syntax can be defined basically in two different ways. For identifying fields where data can be inserted:

-For setting a text field

browser.text_field(:id, ".....").set ("....")

or

browser.text_field(:id=>, '.....').set

-For clicking a button in browser

browser.button(:value, ".....").click()

or

browser.button(:value=>, '.....').click

-For clicking on a link,

browser.link(:text, ".....").click()

-For setting a radio button,

browser.radio(:value => '.... ').set

Locating HTML elements

Locating HTML web elements using WATIR Webdriver works similarly as Selenium Webdriver. The main difference is the WATIR Webdriver has a different syntax.

WATIR Frameworks

This section briefly describes the frameworks compatible with WATIR:

RSpec

This ruby framework is required when a lot of testing needs to be done. RSpec provides good testing practices for ruby test professionals.

Cucumber

The Cucumber framework allows to write the whole test scenario in a plain English language and after that, the final automated test cases are written in Ruby. The plain text used is written in a domain-specific language (a language is used for simplifying specific software domain/systems) and it bridges the gap between technical and non-technical users[44].

Test/Unit

Test/Unit is a unit testing framework for Ruby. It allows to write test scripts, debug and evaluate the code in order to make it easy to maintain. The main idea behind the unit test of this tool is to write a test method, that makes certain assertions about the code (assertions means: statements of expected outcome)[45].

Advantages of WATIR

- One of the advantages is that WATIR uses object-oriented ruby language and ruby is easy to get started with.
- WATIR has an automatic wait, which means it waits for a page to fully load, then tries to identify the web element on the page. Unlike Selenium or UFT, WATIR does not require

wait command. It requires using wait command when the webpage is built with AJAX [43].

- WATIR has a straightforward API with a number of features and can be integrated with other frameworks as well to provide more functionalities. So, testing becomes easier and simpler with WATIR [40].

Limitations of WATIR

- WATIR only allows testing of web applications.
- WATIR does not come with the record and playback of testing, where most of the automated testing tools provide this option.
- Because it's not flourished as much as Selenium, it has only limited number of resources available online. So it is not always helpful in the cases when someone is stuck and needs help to resolve technical issues.

Chapter 3: METHODOLOGY

This chapter discusses the method used for testing. It includes the selection of testing tools, the various criteria to consider while performing the comparison among tools and the specification of machine used for testing.

3.1 Selected Tools

A number of open source and commercial testing tools for windows, web and mobile applications have been used by software industries. The fundamental purpose of these tools is similar, but they differ in functionality, features and usability. For my comparative study, I have selected to work with three open source tools: Selenium, WATIR and the trial version of commercial UFT tool as QTP was not compatible with my Operating system and browsers.

3.2 Evaluation Metrics

- **Script generation-** the process of test script generation or creation using the tool; whether it is a record playback tool or provides an ability to create scripts using different programming languages. Most of the testing tools have the ability to record the test steps and playback the recorded actions.
- **Versatility-** whether the tool supports relevant technologies or integration with other tools which might make the automation process more efficient.[46]
- **Preparation for automation** – configuring or installing the tool and preparing it for automation testing; how successful the tool was in executing the tests.
- **Utilization-** how the tool is utilized for different types of testing; how easy or complex it is to use the tool, based on test results.
- **Various browsers or operating system (OS) support-** providing support to different browsers and OS. This is one of the core features to be analyzed for each automated

testing tool.

- **Test result generation** – after executing the entire test, a final test result should be generated, which provides the availability of the complete test logs and a pass or fail result in the form of a test report. It is essential for executing a large set of test cases in industries.
- **Cost**- cost is one of the most important factors in automated testing. Most of the software industries prefer a cost effective tool. Three different testing tools have been discussed in this paper and only one of them is a commercial costly testing tool.
- **Miscellaneous factors**- there are also some other criteria to compare, including flexibility in using programming languages, database testing compatibility, test execution speed as well as the learning curves for the beginners [39].

Specifications of Testing Machines (for Selenium and WATIR)

The following table (Table-3), refers different features of the machine used to accomplish the tests.

Table 3: Features of the first Machine

Features	Machine selected to perform the test
Manufacturer of computer used	Dell
Operating System	Windows 8.0, 64 bit OS
RAM	4 GB
Processor	Core i3,CPU 2.4GHz
Browser used	Mozilla Firefox version 41.0.2 ; for testing with Selenium IDE, Webdriver, and WATIR

3.3 Target Application

For this research, I have chosen widely used university website to accomplish the testing task on the course registration process. This is the e- Services website for St Cloud State University.

(<https://webproc.mnscu.edu>).

3.4 General Testing Approach

The first step is to write the test cases. Therefore, a number of test cases have been created in order to perform the automated functional testing for course registration process using Selenium (IDE and Webdriver), WATIR and UFT. The overall testing was done using the test cases. For Selenium Webdriver writing test scripts were performed in java and within eclipse IDE; UFT has its own IDE. Finally, for WATIR, each line of the test scripts has been written and executed one by one using Ruby command prompt (irb).

The versions I used for each of the tools are the following:-

Selenium IDE (as a browser plugin) - 2.9.1

Ruby version for WATIR Webdriver -2.00- p645

UFT version- 12.5

Selenium Webdriver client version – 2.45.0 (for Java)

After executing the testing with all the tools, various features and aspects of all these tools will be discussed as well as the comparison among these tools will be presented in the next chapter.

Chapter 4: TESTING AND RESULT ANALYSIS

In this chapter, the details of the testing procedures will be described, which includes the creation of test cases, test scripts, explanations of testing performed, observations after conducting the tests as well as a comparison performed among all the testing tools.

4.1 Test cases

In Table-4, I encompassed the overall test scenario (includes expected result to perform, actual result and overall result as pass/fail) of the course registration process, in the form of test cases.

Table 4: List of Test cases

Expected Result	Actual Result	Overall Result
1. Able to redirect to the SCSU- e-Services website (while testing with Selenium); Able to redirect to SCSU- record and registration web page and able to click to e-Services link (while testing with WATIR/ UFT).	Was able to see the homepage of e-Services ; Was able to see SCSU – Record and Registration home page and then link to e-Services was visible.	Pass
2. Able to click on the StarID field on the homepage and enter the ID (as8834hl)	Was able to see Star ID (as8834hl) field typed automatically	Pass
3. Able to click password(*****) field and enter password for sign in	Was able to see password (*****) typed automatically	Pass

Table 4 Continued

Expected Result	Actual Result	Overall Result
4. Able to click on the LOGIN button for sign in.	Was able to see the successful login and the student ID displayed on the upper right side of home page.	
5. Able to see 'Courses and Registration' on the left side of the page, and able click into that link	Was able to see the page with title: Courses and Registration	Pass
6. Able to click on the link for 'Search for a Course' on the left widget	Was able to see the course search page	Pass
7. Able to see a dropdown box for 'Subject' field and select a course 'Chemistry'	Was able to select 'Chemistry' from drop down	Pass
8. Able to click on the search button	Was able to see search result page displayed with all the offered courses for 'Chemistry'	Pass
9. Able to select and click on the desired course : 'Forensic science'	Was able to see the 'Forensic science' course detailed page displayed with all the info of that course	Pass
10. Able to click on the 'Add' button on the upper left side of the page	Was able to see a pop-up message box appeared after the adding the course, with displaying either of the following	Pass

Table 4 Continued

	message 'Course has been added to your wish list' or 'The course is already in your wish list' (if the test is run multiple times, without logging out).	
11. Able to 'accept' or click 'ok' to the above appeared 'Alert pop-up' to leave that page or change the action	Was able to see the alert pop-up handled and disappeared	Pass
12. Able to click on the link for 'Continue to review my plan' in order to verify whether the course is added to wish list	Was able to add the course under the wish list section	Pass
13. Able to select that course by clicking the checkbox	Was able to see the course got selected	Pass
14. Able to click on the 'Remove selected course from the wish list' button , in order to remove that course from the wish list	Was able to see the selected course removed from the list and a message appeared as 'Course has been removed from wish list'	Pass
15. Able to click on the logout link on the upper right of the page	Was able to see that it logs out and brings back to the Logout Successful page.	Pass

4.2 Method used to locate HTML element

In order to write the code for testing, regardless of tools (Selenium IDE, Selenium web driver or WATIR) used, HTML content for the webpage is needed to generate and understand. To identify web page elements fast and in an accurate way, different browser tools can be utilized. I have used 'Firebug', which is a Firefox browser add-on. It just needs to be configured in the browser. In order to find an HTML attribute, an individual needs to hover over that area on the web page then, Firebug highlights the HTML code for that small portion.

For example, in Figure-1, on e-services login page, I went through the process of locating web element using HTML tag name for StarID field. The bold circle outlined area in the left corner is the tab for 'Firebug' tool. Using Firebug, I hovered over the 'StarID' field (outlined in bold) and it showed the corresponding HTML code below in highlighted section. From that section, it is possible to find out the exact HTML tag name or attribute, which should be unique for that field.

ST. CLOUD STATE UNIVERSITY

St. Cloud State University
720 Fourth Ave. S.
St. Cloud, MN
56301-4498
USA

Phone:
320-308-0121

Toll-free:
1-877-654-7278

Telecommunications Device
for the Deaf:
1-800-627-3529

Please login to continue.

The '*' indicates a required field.

* StarID: [Need Login Help?](#) Need an ID? [Sign Up Now](#)

* Password:

Institution: St. Cloud State University

Display Name: Display and print your name until next login. To protect your identity, you may wish to print only at secured locations.

Login

https://www.mozilla.org/en-US/firefox/central/

Console HTML CSS Script DOM Net Cookies FirePath

Edit input#userName.input <td <tr <tbody <table#loginTable <form#loginForm <div#main.has-sideba

```
<th class="nophone">* StarID:</th>
<td>
<input id="userName" class="input" type="text" title="StarID
is
required." value="" tabindex="1" maxlength="60" name="userNam
e">
<span class="nophone"> </span>
<span class="nophone">
```

Figure-1: HTML element locating procedure on web page

But this is not the case always, as most of the page element attributes are not static. These attributes can change anytime, then the code will fail. That's why it's required to look for an alternative way and this can be done using 'XPath' from HTML source code. XPath is a syntax and uses path expressions to navigate in the XML documents.

There are two types of XPath available:

Absolute XPath: It defines the specific location of the web element (finds element in the form of a tree). An absolute XPath begins with a root node. The advantage of using absolute XPath is that it identifies the element very fast. On the other hand, if a small change occurs in the source code (if any other web element or HTML tag name is changed/ added in that webpage), the

absolute XPath will fail. Here's the example of absolute XPath for StarID field:

html/body/div/div [id()='userName']

Relative XPath: It doesn't need to start from the root node. A relative XPath can begin from the current location and is prefixed with a '//'. The syntax for writing a relative XPath,

//tag name[@attribute = 'value']

Here is the relative XPath for StarID field: **//*[@id='userName']**.

This one is very straightforward, as already there is a unique 'id' attribute available for StarID.

XPaths can be complicated, when there is no unique attribute (id, name, value Etc.) available for a specific field. The benefit of choosing the target elements as XPath is that the HTML tag names are likely to stay unchanged for a longer period and therefore requires less maintenance on code.

The following screenshot shows the method of locating the absolute XPath for StarID field.

```

<html>
  <head>
  </head>
  <body>
    <div data-role="page">
      <div id="masthead" class="nophone">
        <p class="skiplink">
          <p id="masthead-inst-name" class="masthead"> St. Cloud State University </p>
          <div id="masthead-inner"></div>
        </div>
      <div id="container" class="login-page">
        <div id="content-container">
          <div id="main" class="has-sidebar">
            <h1 class="nophone">Please login to continue.</h1>
            <form id="loginForm" action="/esession/login.do" method="post">
              <input type="hidden" value="http://webproc.mnscu.edu/registration/secure/search/basic.html;
                jsessionid=552E778819E3285A61E36A2DF54E05A9?campusid=073" name="postAuthUrl">
              <input type="hidden" value="" name="viewLoginForwardName">
              <input type="hidden" value="0073" name="rcId">
              <p class="warning nophone"> </p>
              <div id="phoneimage" style="display: none">
              <div class="errors"></div>
              <p class="nophone">The '*' indicates a required field.</p>
              <div id="loginIdMsg" class="warning hideBlock">
                <noscript> <div class="warning"> You should use your StarID to login to eServices. Logging in with the
                  TechID or Username will be discontinued soon. You can <a href="http://starid.mnscu.edu/go/activate
                    /">Activate your StarID now<a/> and use it to access the application. </div> </noscript>
              <table id="loginTable">
                <tbody>
                  <tr>
                    <th class="nophone">* StarID:</th>
                    <td>
                      <input id="userName" class="input" type="text" title="StarID is
                        required" />
                    </td>
                  </tr>
                </tbody>
              </table>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>

```

Figure-2: HTML source code on e-services login page

It starts with the HTML root node 'head', then inside the 'body', there are two main div tags to expand and then 'id' = username is found.

4.3 Test using Selenium IDE:

-Configuring the tool

The Selenium IDE tool can be downloaded directly and configured as a browser plugin for Mozilla Firefox. There's no need to install this tool.

-Testing Details:

Once the recording mode is activated, after starting play, then any data entered in the browser will automatically be recorded in the Selenium IDE, in an unencrypted form. For example, entered user IDs, passwords, and URL changes are recorded as clear texts. Scripts in Selenium IDE consists of commands, target, and value. *Command* is an action to perform; the *target* is the destination field, which is the HTML web element entered against each command; *values* are very specific and entered against target field [11].

All the commands, targets and values in Selenium IDE can be entered either manually or by recording a script. For each test step, Selenium IDE compares and matches the entered 'target', and 'value' fields with the actual ones in the webpage. If the actions match, it passes the test, otherwise test fails. Selenium IDE generates its test result or logs based on the execution of each test case, which can be found under the 'log' tabs in the bottom.

I entered all the actions in Selenium IDE manually, following the HTML source code on the webpage and based on the test cases.

In Table -5, I have enlisted the scripts, which were executed in Selenium IDE against each test case.

Table 5: Selenium IDE Test script

Command	Target	Value
1. Open	https://webproc.mnscu.edu/esession/authentication.do?campusId=073&postAuthUrl=http%3A%2F%2Fwebproc.mnscu.edu%2Fservices%2Flogin.html%3Fcampusid%3D073	
2. Type	id=username	as8834hl

Table 5 Continued

Command	Target	Value
3. Type	id=password	*****
4. clickAndWait	xpath=//input[@value='Login']	
5. clickAndWait	link=Courses & Registration	
6. clickAndWait	link=Search for a Course	
7. select	id=subject	label=Chemistry (CHEM)
8. click	//option[@value='CHEM']	
9. clickAndWait	class=btn-primary	
10. clickAndWait	link=Forensic Science	
11. click	xpath=//*[@id='main']/table[1]/tbody[2]/tr/td[1]/div/a[1]/img	
12. assertAlert	The course is currently in your Wish List	
13. clickAndWait	link=Continue to Review My Plan >	
14. click	name=selectCourse_002163_20165_0073_0073_false	
15. clickAndWait	xpath=(//*[@id='ViewCartForm']/p/input[2])	
16. clickAndWait	link=Logout	

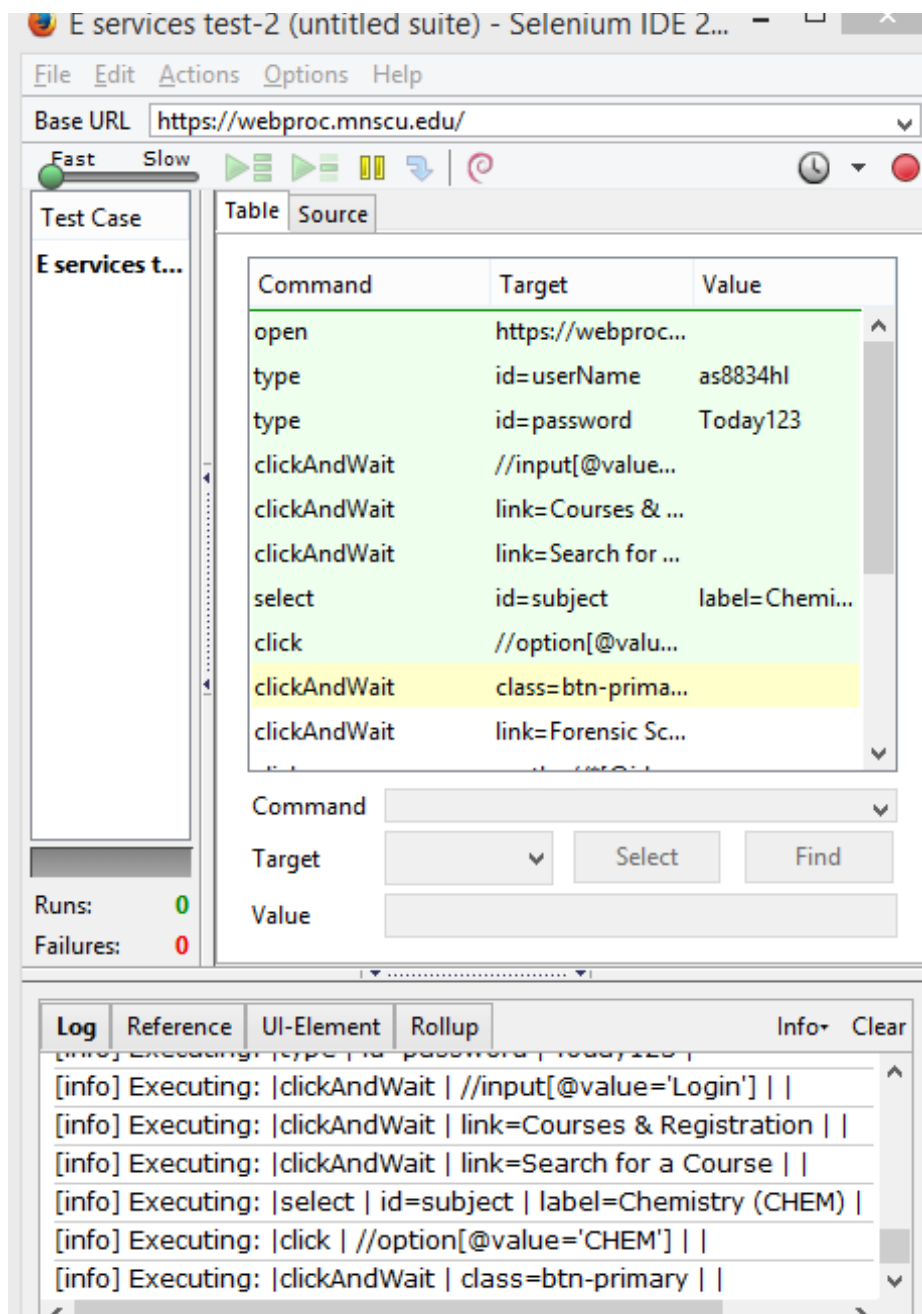


Figure-3: Selenium IDE executing command on course search page

The screenshot shows a Selenium IDE test suite for 'Eservices test'. The test case 'Eservices t...' contains the following commands:

Command	Target	Value
type	id=password	Today123
clickAndWait	//input[@value...	
clickAndWait	link=Courses & ...	
clickAndWait	link=Search for ...	
select	id=subject	label=Chemi...
click	//option[@valu...	
clickAndWait	class=btn-prima...	
clickAndWait	link=Forensic Sc...	
click	xpath=//*[@id=...	
assertAlert	The course is cu...	

The log shows the following error:

```
[error] Actual value 'Course has been added to Wish List' did not match 'The course is currently in your Wish List'
```

The test case failed, and the test suite completed with 1 played and 1 failed.

Figure-4: Selenium IDE test fails for not matching actual value on page with test input value

Figure-3, in the above, is a snapshot of performing the test in Selenium IDE. The IDE also provides the test execution logs; here it shows, each line of test script execution without any error.

The Figure-4 shows a test case that failed to execute. It generated an error log at the end of the test in Selenium IDE. It fails while testing the pop-up alert on the webpage. It failed because the actual text of the pop-up alert was different than the text provided in 'target' field of Selenium IDE as input.

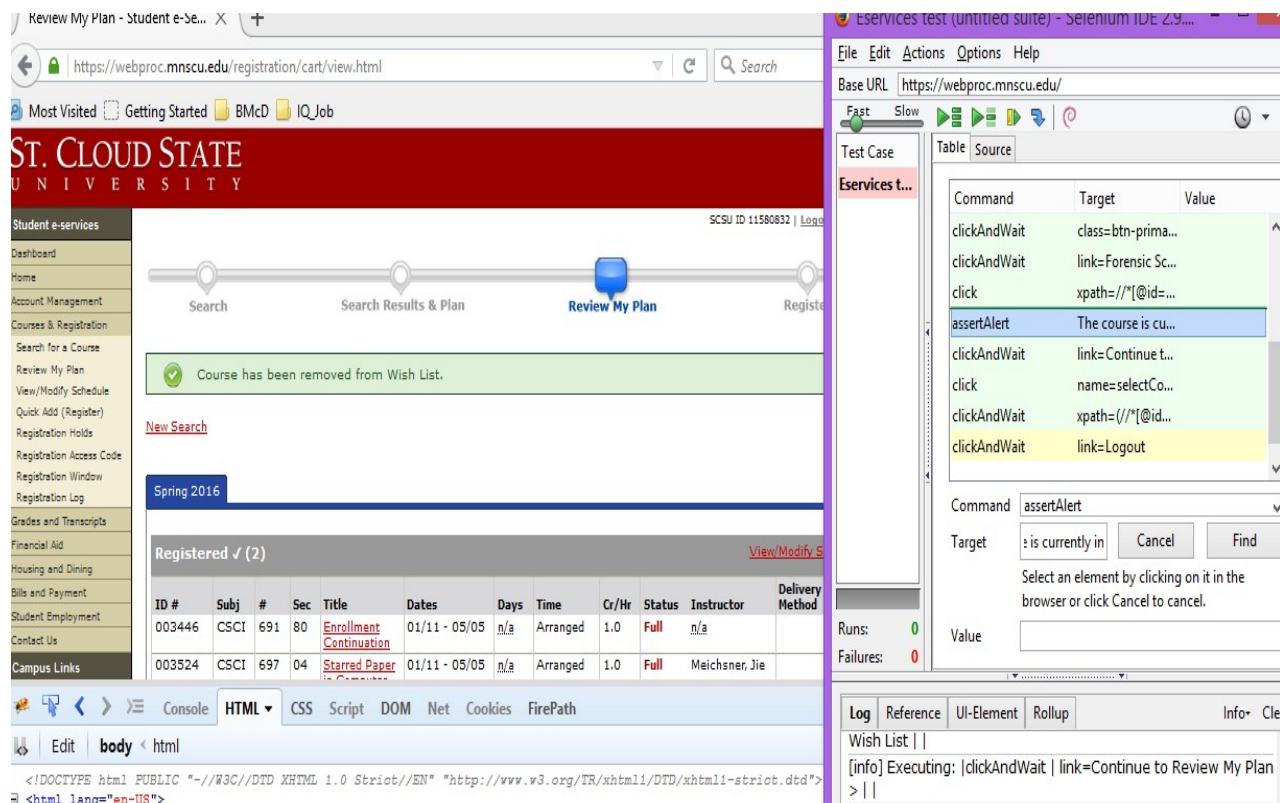


Figure-5: Test case passes on updating the target alert message on Selenium IDE

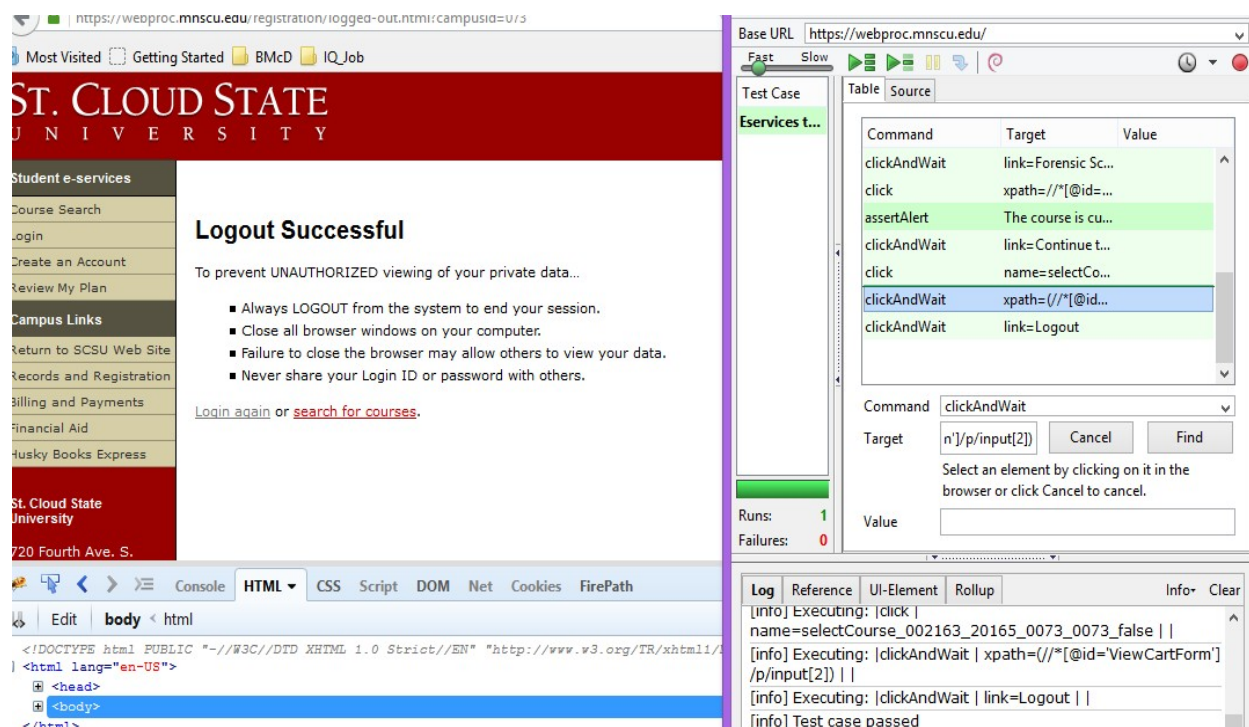


Figure-6: When all the test cases pass from login to logout

Figure-5, shows the test passes successfully in Selenium IDE while providing the correct input for the pop-up alert text. Figure-6, shows a snapshot of executing the final test case was successful in Selenium IDE and it was the logging out activity from e-Services website.

Observation for Selenium IDE:

1. Setup and configuring with the browser is easy and fast, although it works with only Firefox browser.
2. The execution time of test cases or whole test scenario can be controlled by adjusting the speed of Selenium IDE, from slow to fast.
3. Sometimes the test cases fail because of not adjusting the speed of execution appropriately and there's no certain way to fix the speed always. If it fails because of executing test very fast, it does not show the exact reason in 'log' tab. It needs some

- troubleshooting, to figure out the actual reason for failing test cases.
4. Each time when the test case is run, it needs to start from the beginning; it can't start from the point where it was paused last time. So, when the numbers of test cases are huge, it takes too much time to run multiple times, but provides better recording efficiency, if the whole test scenario is recorded in one attempt (although, that's not a common scenario).
 5. It's not very reliable when the value of different attributes (such as id, name) are changing frequently. So, for reusing the test script, it needs to be updated from time-to-time.

4.4 Test using Selenium Webdriver

Configuring Selenium Webdriver:

As I have tested Selenium Webdriver using java language, it requires both java and eclipse IDE to be installed first and then downloaded 'Selenium client and Webdriver language bindings' for java. In order to start testing with Webdriver, I needed to configure the Selenium library and jar files to the project build path. After creating the 'Referenced Libraries' along with the project, all the library files need to be displayed inside reference libraries. Then only it allows executing java test codes.

Scripts in Selenium Webdriver:

The section below is the java code for Selenium web driver which was executed in eclipse IDE.

```
import org.openqa.selenium.Alert;  
  
import org.openqa.selenium.By;  
  
import org.openqa.selenium.firefox.FirefoxDriver;  
  
import org.openqa.selenium.support.ui.Select;
```

```

public class SelHusky1 {

/**

    * @param args

    * @throws InterruptedException

    */

    public static void main(String[] args) throws InterruptedException {

        // TODO Auto-generated method stub

        /**Create firefox driver to drive the browser**

        FirefoxDriver dr = new FirefoxDriver();

        /** Open e-Services logon page website homepage**

        dr.get("https://webproc.mnscu.edu/esession/authentication.do?campusId=073&postAuthUrl=http

        %3A%2F%2Fwebproc.mnscu.edu%2Feservices%2Flogin.html%3Fcampusid%3D073");

        /**Enter username in the StarID field**

        dr.findElement(By.id("userName")).sendKeys("as8834hl");

        /**Enter password field**

        dr.findElement(By.id("password")).sendKeys("Today123");

        /**clicking to the Login button, which brings to e-services homepage**

        dr.findElementByXPath("//input[@value='Login']").click();

        /**Clicking to the link of courses and registration **

        dr.findElementByLinkText("Courses & Registration").click();

        /**Waits for the next web element to load **

        Thread.sleep(30000);

        /**Clicking to the link of Search for a course**

```



```

dr.findElementByXPath("//div[@id='main']/ul/li/a").click();

/**Selecting a subject(here it's chemistry) from dropdown box **

Select course=new Select(dr.findElement(By.id("subject")));

course.selectByValue("CHEM");

/**Searching the courses for selected subject**

dr.findElementByXPath("//form[@id='AdvancedSearchForm']/input[5]").click();

/**Waits for correct page to be displayed.....**

Thread.sleep(30000);

/**Looks for Quantitative Analysis course and enters to course content**

dr.findElementByXPath("//*[ @id='Forensic Science002163']/a").click();

/**Waits for the next web element to load **

Thread.sleep(300);

/**Add course to the wish list**

dr.findElementByXPath("//*[ @id='main']/table[1]/tbody[2]/tr/td[1]/div/a[1]/img"

).click();

/**Setting the alert command to handle the dialogbox,after adding the

course**

Alert alert = dr.switchTo().alert();

/**Will Click on OK button**

alert.accept();

/**Entering to link of Continue to review my plan **

dr.findElementByLinkText("Continue to Review My Plan >").click();

/**Waits for the next web element to load **

```

```
Thread.sleep(30000);

/**Selecting the course which was just added**

dr.findElementByName("selectCourse_002163_20165_0073_0073_false").click();

/**Removing that course from wish list**

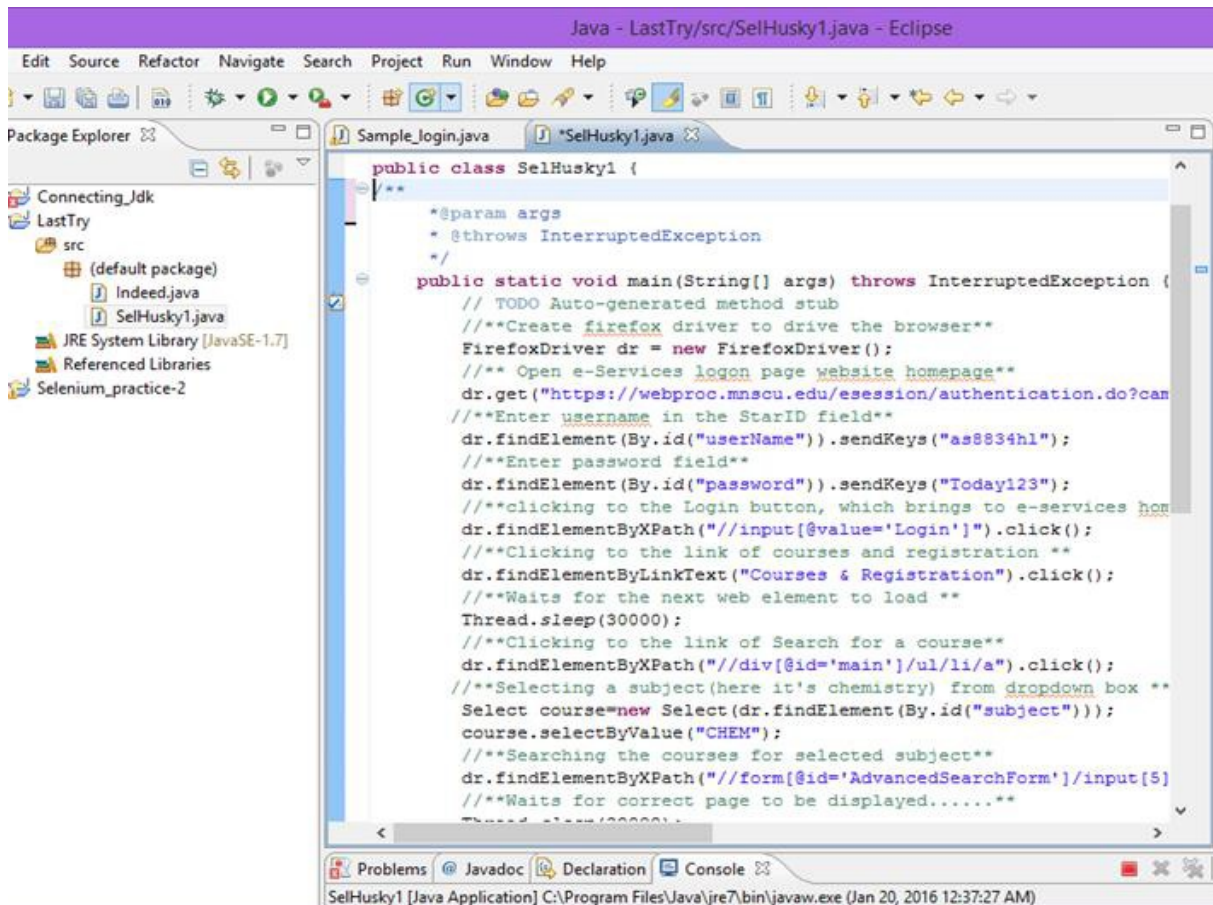
dr.findElementByXPath("//*[@id='ViewCartForm']/p/input[2]").click();

// **Logout from e services**

dr.findElementByLinkText("Logout").click();

}

}
```



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project named 'LastTry' with a source folder 'src' containing 'SelHusky1.java'. The main editor window shows the code for 'SelHusky1.java'. The code is a public class with a main method that uses Selenium WebDriver to automate a login and search process on a website. The code includes comments and uses various Selenium methods like 'sendKeys', 'click', 'findElement', and 'selectByValue'. The status bar at the bottom indicates the application is running as 'SelHusky1 [Java Application]'.

```
public class SelHusky1 {  
    /**  
     * @param args  
     * @throws InterruptedException  
     */  
    public static void main(String[] args) throws InterruptedException {  
        // TODO Auto-generated method stub  
        /**Create firefox driver to drive the browser**  
        FirefoxDriver dr = new FirefoxDriver();  
        /** Open e-Services logon page website homepage**  
        dr.get("https://webproc.mnscu.edu/esession/authentication.do?can  
        /**Enter username in the StarID field**  
        dr.findElement(By.id("userName")).sendKeys("as8834h1");  
        /**Enter password field**  
        dr.findElement(By.id("password")).sendKeys("Today123");  
        /**clicking to the Login button, which brings to e-services hom  
        dr.findElementByXPath("//input[@value='Login']").click();  
        /**Clicking to the link of courses and registration **  
        dr.findElementByLinkText("Courses & Registration").click();  
        /**Waits for the next web element to load **  
        Thread.sleep(30000);  
        /**Clicking to the link of Search for a course**  
        dr.findElementByXPath("//div[@id='main']/ul/li/a").click();  
        /**Selecting a subject (here it's chemistry) from dropdown box **  
        Select course=new Select(dr.findElement(By.id("subject")));  
        course.selectByValue("CHEM");  
        /**Searching the courses for selected subject**  
        dr.findElementByXPath("//form[@id='AdvancedSearchForm']/input[5]  
        /**Waits for correct page to be displayed.....**  
        Thread.sleep(30000);  
    }  
}
```

Figure-7: Selenium Webdriver test script written in java eclipse IDE_part1

```

Sample_login.java *SelHusky1.java
dr.findElement(By.id("userName")).sendKeys("as8834h1");
/**Enter password field**
dr.findElement(By.id("password")).sendKeys("Today123");
/**clicking to the Login button, which brings to e-services hom
dr.findElementByXPath("//input[@value='Login']").click();
/**Clicking to the link of courses and registration **
dr.findElementByLinkText("Courses & Registration").click();
/**Waits for the next web element to load **
Thread.sleep(30000);
/**Clicking to the link of Search for a course**
dr.findElementByXPath("//div[@id='main']/ul/li/a").click();
/**Selecting a subject (here it's chemistry) from dropdown box **
Select course=new Select(dr.findElement(By.id("subject")));
course.selectByValue("CHEM");
/**Searching the courses for selected subject**
dr.findElementByXPath("//form[@id='AdvancedSearchForm']/input[5]
/**Waits for correct page to be displayed.....**
Thread.sleep(30000);
/**Looks for Quantitative Analysis course and enters to course
dr.findElementByXPath("//*[id='Forensic Science002163']/a").cli
/**Waits for the next web element to load **
Thread.sleep(300);
/**Add course to the wish list**
dr.findElementByXPath("//*[id='main']/table[1]/tbody[2]/tr/td[1
/**Setting the alert command to handle the dialogbox, after addi
Alert alert = dr.switchTo().alert();
/**Will Click on OK button**
alert.accept();
/**Entering to link of Continue to review my plan **
dr.findElementByLinkText("Continue to Review My Plan >").click()

Sample_login.java *SelHusky1.java
/**Looks for Quantitative Analysis course and enters to course
dr.findElementByXPath("//*[id='Forensic Science002163']/a").cli
/**Waits for the next web element to load **
Thread.sleep(300);
/**Add course to the wish list**
dr.findElementByXPath("//*[id='main']/table[1]/tbody[2]/tr/td[1
/**Setting the alert command to handle the dialogbox, after addi
Alert alert = dr.switchTo().alert();
/**Will Click on OK button**
alert.accept();
/**Entering to link of Continue to review my plan **
dr.findElementByLinkText("Continue to Review My Plan >").click()
/**Waits for the next web element to load **
Thread.sleep(30000);
/**Selecting the course which was just added**
dr.findElementByName("selectCourse_002163_20165_0073_0073_false"
/**Removing that course from wish list**
dr.findElementByXPath("//*[id='ViewCartForm']/p/input[2]").clie
// Logout from e services
dr.findElementByLinkText("Logout").click();

}
Problems @ Javadoc Declaration Console
SelHusky1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Jan 20, 2016 12:37:27 AM)

```

Figure-8: Selenium Webdriver test script written in java eclipse IDE_part2

The above figures (Figure-7 and Figure-8) present the complete test scripts of Selenium Webdriver written in java and within Eclipse IDE.

Explanation of the code used for testing

As I have chosen to test with only Firefox browser, Firefox plugins for Selenium must be imported in beginning of writing the test code. The following packages are imported in the code:

Importing Selenium Packages

To start with the code we need to first import following packages: –

1. **org.openqa.selenium** - Contains the Webdriver class needed to instantiate a new browser loaded with a specific driver.
2. **org.openqa.selenium.firefox.FirefoxDriver** – Contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser.

3. **org.openqa.selenium.By**

The 'By' class is used to locate any element on the webpage and this class has subclasses. Depending on the finding mechanism, there are many subclasses (i.e. ByName, ByXpath, ByLinktext..etc).

4. **org.openqa.selenium.support.ui.Select**

Represents a SELECT tag, providing helper methods to select and deselect options.

Initiating a Driver Object

For testing, I have used firefox browser, the Driver object is instantiated as below –

```
FirefoxDriver dr = new FirefoxDriver();
```

The default Selenium driver “FirefoxDriver()” is launched by the Java Program.

Launching the Browser Session

“dr.get()” method is used for launching new browser session, where the URL is specified as a parameter in ‘get()’

Finding the element on the webpage

As discussed earlier, **findElement()** method is used to find any element on the webpage. In my test, I have used various type of locators, such as id, name, value, XPath, linktext etc.

Applying Wait method:

Sometimes the browser needs time to load a specific page, this is the reason for using wait command. In the code, I have used generic java wait method with **Thread.sleep ()**. The fraction of time used inside sleep (), is milliseconds and that can vary based on the wait time.

Following up Alerts :

I have used the following method to check if the alert is present on the webpage:

```
alert = dr.switchTo().alert();
```

"switchTo().alert()" method is used to access elements within alert box.

To close the alert pop-up with clicking 'OK', the following method is used:

```
alert.accept();
```

Selenium alert interface has own methods for handling alerts, such as accept (), dismiss (), respectively for accepting or dismissing. It needs to import the following package:

```
import org.openqa.selenium.Alert;
```

Observation for Selenium Webdriver:

1. Configuring Selenium Webdriver with java and eclipse takes a little more time and it needs to follow a systematic process to configure properly.
2. Sometimes, I encountered a java error, “Unable to locate element” (error in Figure-9), which occurred due to the fast loading of webpages. Although, a

human can view the availability of certain link/button/text in the webpage but it takes little more time for the testing tool to compare and match the defined input in the test code to the html element on the web page. In order to avoid that error, I have used the `thread.sleep ()` method, where the browser waits for the definite amount of time entered in the `sleep()` function.

Another similar issue occurred, when the firefox browser was auto- updated to latest version and I executed the Selenium Webdriver code without upgrading Webdriver version and received the same error (in Figure-9).

Finally, I needed to revert back the browser upgrade, to execute the code without error.

3. The main difference between Webdriver and IDE is that the commands for IDE are too specific to each testing step, where the Webdriver coding style is more generic. For example, the `alert()` function is appropriate for any type of alert or pops up handling in the web page, regardless of the texts in the dialogbox. Webdriver codes are more reusable.
4. Selenium Webdriver does not provide any test result after executing the test.

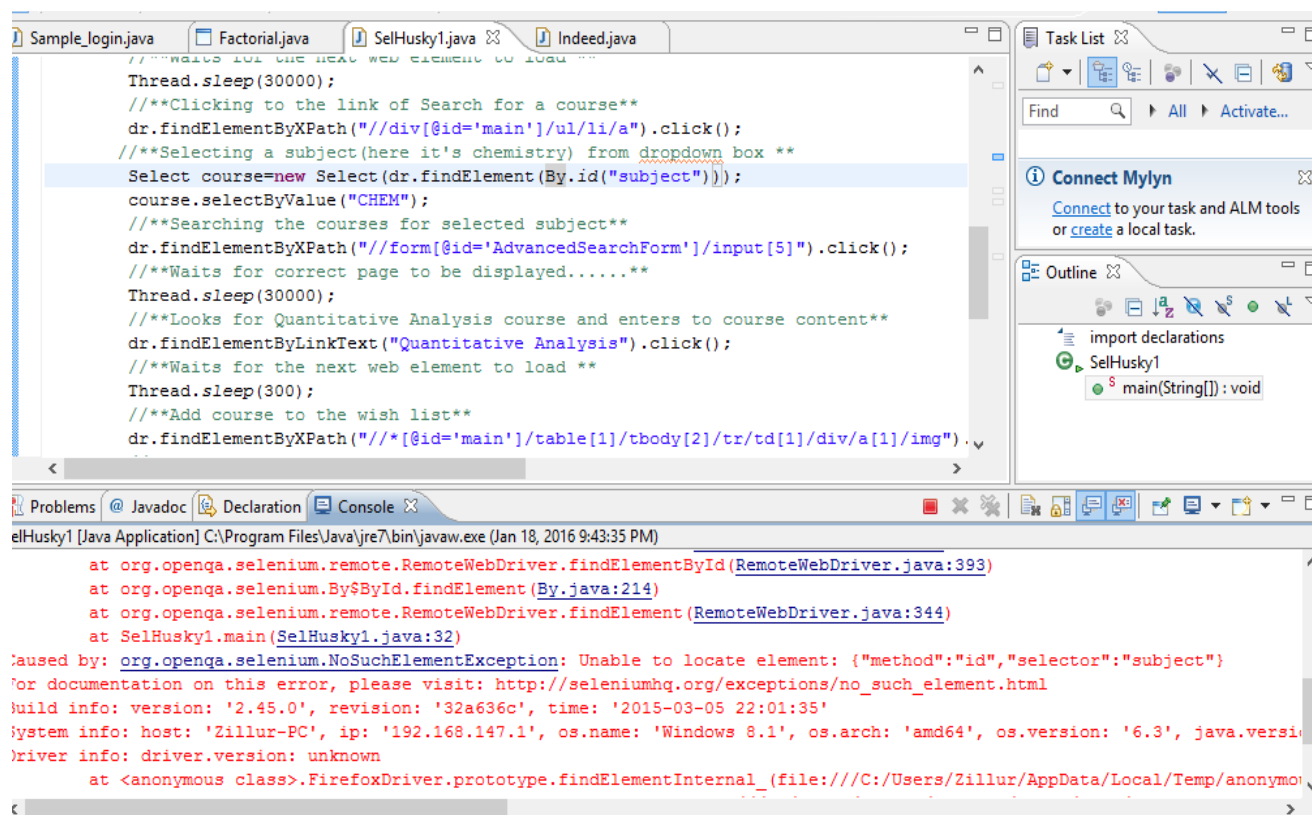


Figure-9: Error during executing test in Selenium Webdriver

4.5 Comparison between Selenium IDE and Selenium Webdriver

Table 6: Selenium IDE vs. Selenium Webdriver

Selenium IDE	Selenium Webdriver
1. It's mainly a record and playback tool	This tool provides option to create test scripts.
2. It runs only on Mozilla Firefox	It uses number of browsers: Firefox, IE, Chrome, Safari, Opera
3. It uses IDE specific commands, which is called 'Selenese' (the language used to script in selenium IDE).	It utilizes different programming languages, such as Java, Python, C#, PHP, Ruby, and Perl.

Table 6 Continued

Selenium IDE	Selenium Webdriver
<p>4. A test recorded using IDE, might not always pass if it is saved and played again; Because IDE just cares about the elements on the web page at the time of recording, which frequently changes.</p>	<p>Once a script is written and successfully executed using Selenium web driver, it can be run later and has fewer chances of failing; As Webdriver coding style is more generic.</p>
<p>5. It explicitly shows if a test is a pass (in Green) or fails (in Red).</p>	<p>It does not show anything if a test is pass or fail. The console log shows only the number of error messages if the test fails.</p>
<p>6. It's easier to use, as a little programming knowledge is needed only. It requires understanding the HTML source code for webpage.</p>	<p>Selenium Webdriver requires good skills in any of the programming languages it offers.</p>
<p>7. This tool can be easily configured in Mozilla Firefox browser and it's a Firefox browser add-on</p>	<p>Webdriver using java can be configured in Eclipse IDE.</p>
<p>8. In IDE there is a built-in option of adjusting test case execution speed</p>	<p>In Webdriver, there are different ways to manage the test case execution speed. While using java, it can be handled by synchronization (explicit and implicit wait) and also by using thread sleep() method.</p>

4.6 Test using UFT

Installation of UFT:

Installing UFT or QTP for free of cost, is possible only for 1 month, with the unlicensed version. In order to download the tool, an account in HP needs to be created. The installation process for UFT is simple and easy, it provides option to install number of features such as ALM (Application Lifecycle Management) plug-in, Visual Basic add-in .NET add-in, web add-in, java add-in, oracle add-in and some more within the installation package. After the installation, it's recommended to use just a few feature at a time for faster execution. For my testing, I just used web and visual basic add-in.

Scripts in UFT:

'On Browser IE 11, open the SCSU- Record and Registration webpage and click to the e-Services link

```
Browser("SCSU - e-Services").Page("Records and Registration").Link("eServices").Click
```

'Opens the e Services student login page; Entering StarID

```
Browser("SCSU - e-Services").Page("Login Page").WebEdit("userName").Set "as8834hl"
```

'Entering password

```
Browser("SCSU - e-Services").Page("Login Page").WebEdit("password").Click
```

```
wait(2)
```

' The line below added by UFT automatically for encrypting the password

```
Browser("SCSU - e-Services").Page("Login Page").WebEdit("password").SetSecure
```

```
"56a6f3de0800dde2ed5818872346949ec3ef33573aac"
```

'Click to login button

```
Browser ("SCSU - e-Services").Page ("Login Page").WebButton("Login").Click
```

wait(2)

'On successful login, click to courses and registration link

Browser("SCSU - e-Services").Page("Student Portal - 2.8.0").Link("Courses & Registration").Click

wait(2)

'On courses and registration link, click again to Search for a Course link

Browser("SCSU - e-Services").Page("Courses and Registration").Link("Search for a Course").Click

wait(2)

'On course search page, select the subject of study and it's chemistry

Browser("SCSU - e-Services").Page("Course Search - Student").WebList("subject").Select "Chemistry (CHEM)"

wait(2)

'Click search for classes

Browser ("SCSU - e-Services").Page ("Course Search - Student").WebButton("Search>").Click

wait(2)

'Select and click for any of the classes offered for Chemistry (Selected class name is Forensic Science)

Browser("SCSU - e-Services").Page("Course Search Results").Link("Forensic Science").Click

wait(2)

' Enter into the selected class(Forensic Science) and click to add the class

Browser("SCSU - e-Services").Page("Course Detail - Student").Image("plus-icon").Click

wait(2)

' An alert pop-up appears, after adding the course

' Below dialog box is to handle the pop up

```
Browser("SCSU - e-Services").Dialog("Message from webpage").WinButton("OK").Click  
wait(2)
```

' Click Continue to review my plan link to view the added class

```
Browser("SCSU - e-Services").Page("Course Detail - Student").Link("Continue to Review  
My").Click
```

```
wait(2)
```

'Select the added class to remove

```
Browser("SCSU - e-Services").Page("Review My Plan - Student").WebElement  
("selectCourse_002163_20165_0073").Click
```

```
wait(2)
```

' To turn ON the check box, while selecting the course for removal

```
Browser("SCSU - e-Services").Page("Review My Plan - Student").
```

```
WebCheckBox("selectCourse_002163_20165_0073").Set "ON"
```

```
wait(2)
```

' Click the button to remove the selected course

```
Browser("SCSU - e-Services").Page("Review My Plan - Student").WebButton("Remove  
Selected Course(s)").Click
```

```
wait(2)
```

' After successful removal of that course, Clicking logout link

```
Browser("SCSU - e-Services").Page("Review My Plan - Student_2").Link("Logout").Click
```

iers\Hello\Documents\Unified Functional Testing\UITest1- Eservices test record1

The screenshot shows the UFT test script editor with the following content:

```

1
2 'Testing the ST Cloud State University - Course registration process using UFT 12.5.....
3
4 ' On Browser IE 11, open the SCSU- Record and Registration webpage and aclick to the e Services link
5 Browser("SCSU - e-Services").Page("Records and Registration").Link("eServices").Click
6 'Opens the e Services student login page; Entering StarID
7 Browser("SCSU - e-Services").Page("Login Page").WebEdit("userName").Set "as8834h1"
8 'Entering password
9 Browser("SCSU - e-Services").Page("Login Page").WebEdit("password").Click
10 wait(2)
11 ' The line below added by UFT automatically, for encrypting the password
12 Browser("SCSU - e-Services").Page("Login Page").WebEdit("password").SetSecure "56a6f3de0800dde2ed5818872346949ec3ef33573aac"
13 ' Click to login button
14 Browser("SCSU - e-Services").Page("Login Page").WebButton("Login").Click
15 wait(2)
16 'On successful login, click to courses and registration link
17 Browser("SCSU - e-Services").Page("Student Portal - 2.8.0").Link("Courses & Registration").Click
18 wait(2)
19 'On courses and registration link, click again to Search for a Course link
20 Browser("SCSU - e-Services").Page("Courses and Registration").Link("Search for a Course").Click
21 wait(2)
22 'On course search page, select the subject of study and it's chemistry
23 Browser("SCSU - e-Services").Page("Course Search - Student").WebList("subject").Select "Chemistry (CHEM)"
24 wait(2)
25 'Click search for classes
26 Browser("SCSU - e-Services").Page("Course Search - Student").WebButton("Search >").Click
27 wait(2)
28 ' Select and click for any of the classes offered for Chemistry
29 Browser("SCSU - e-Services").Page("Course Search Results").Link("Forensic Science").Click

```

Figure-10: UFT test script-1

```

rs\Hello\Documents\Unified Functional Testing\GUITest1- Eservices test record1
Search Design Record Run Resources ALM Tools Window Help
UFT Report: GUITest1- Eservices test record1 - Res3 Action1 X Start Page
Main
20 Browser("SCSU - e-Services").Page("Courses and Registration").Link("Search for a course").Click
21 wait(2)
22 'On course search page, select the subject of study and it's chemistry
23 Browser("SCSU - e-Services").Page("Course Search - Student").WebList("subject").Select "Chemistry (CHEM)"
24 wait(2)
25 'Click search for classes
26 Browser("SCSU - e-Services").Page("Course Search - Student").WebButton("Search >").Click
27 wait(2)
28 ' Select and click for any of the classes offered for Chemistry
29 Browser("SCSU - e-Services").Page("Course Search Results").Link("Forensic Science").Click
30 wait(2)
31 ' Enter into the selected class(Forensic Science) and click to add the class
32 Browser("SCSU - e-Services").Page("Course Detail - Student").Image("plus-icon").Click
33 wait(2)
34 ' An alert pop-up appears , after adding the course
35 ' this is below dialogbox is to handle the pop up
36 Browser("SCSU - e-Services").Dialog("Message from webpage").WinButton("OK").Click
37 wait(2)
38 ' Click Continue to review my plan link to view the added class
39 Browser("SCSU - e-Services").Page("Course Detail - Student").Link("Continue to Review My").Click
40 wait(2)
41 'Select the added class to remove
42 Browser("SCSU - e-Services").Page("Review My Plan - Student").WebElement("selectCourse_002163_20165_0073").Click
43 wait(2)
44 ' To turn ON the check box, while selecting the course
45 Browser("SCSU - e-Services").Page("Review My Plan - Student").WebCheckBox("selectCourse_002163_20165_0073").Set "ON"
46 wait(2)
47 ' Click the button to remove the selected course
48 Browser("SCSU - e-Services").Page("Review My Plan - Student").WebButton("Remove Selected Course(s)").Click
49 wait(2)
50 ' After successful removal of that course, Clicking Logout link
51 Browser("SCSU - e-Services").Page("Review My Plan - Student_2").Link("Logout").Click

```

Figure-11: UFT test script-2

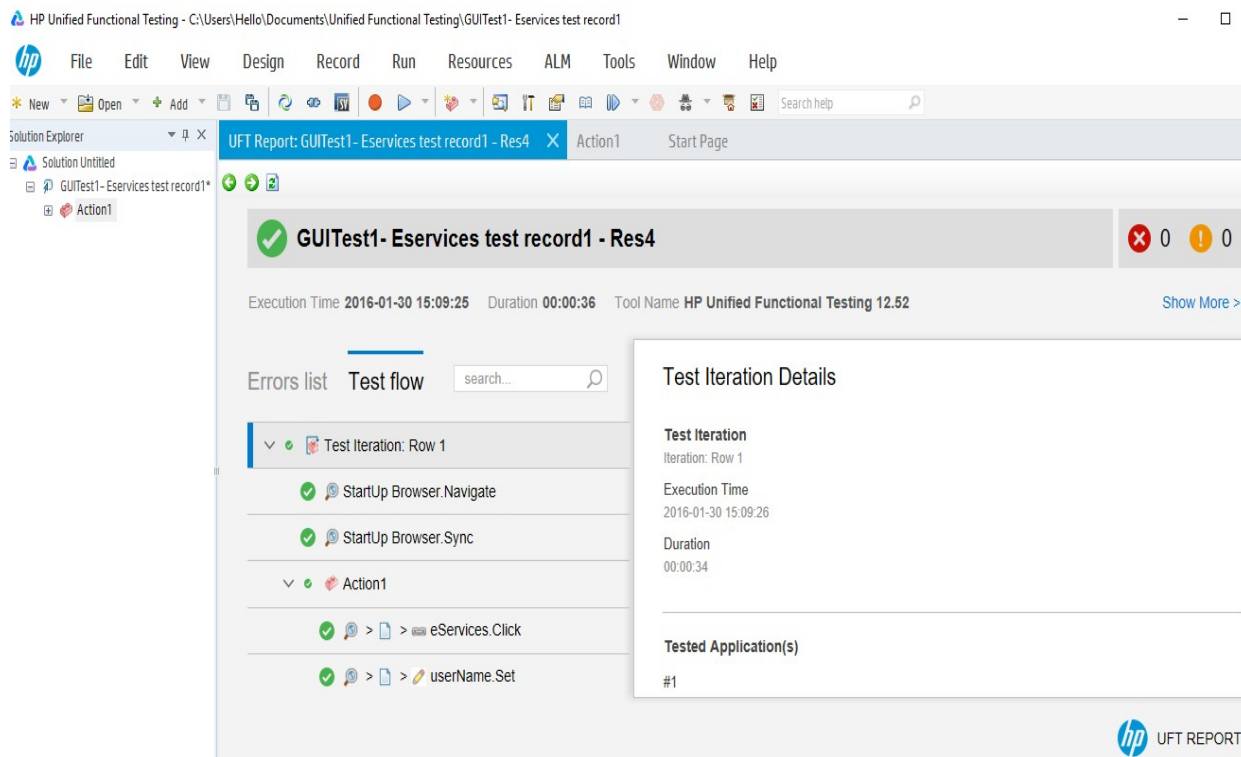


Figure-12: Test result after executing test in UFT

The above screenshot (Figure-10), presents the test scripts in UFT. The test scripts were generated by recording the test cases of course registration process. Figure-11, shows the result, after executing the entire test successfully. The result includes test execution time, duration and overall result (pass or fail). Since the test was a 'pass', so there was no additional information displayed in test result page.

Explanation of UFT Script:

Different versions of the system, OS and browser have been used to perform the test using UFT tool, which is presented in the following table.

Table 7: Features of the second Machine

Operating System	Windows 10, 64 bit
Browser	Internet Explorer (IE) 11.0
UFT	12.52 version (latest)
Computer	HP, core i3, 1.9 GHz processor, 8GB memory

UFT is the latest version of QTP, which supports some advanced new features. As mentioned earlier, the core reason behind using UFT tool is that UFT supports all the latest operating systems and browsers, whereas QTP supports only the older versions of browsers and operating systems, which is not feasible for testing.

Testing with UFT is a little different than using Selenium or WATIR, as it generates code in Visual Basic scripting language and identifies any web element in the form of ‘object’. When it tests a web application, UFT records everything and saves the objects in the object repository. If any changes occur in the web page, it needs to change the name of the specific element in the object repository, instead of changing the script. Then object repository modifies the script automatically.

UFT executes test script very fast. In order to adjust the test execution speed, I have used the wait () method, which takes parameter in seconds. In UFT, each line of the test script is generated in a hierarchical manner. It starts with the ‘browser’ class, then a web ‘page’ class and then performs any specific action with WebEdit, WebButton, Link (classes) on objects using ‘click’ or ‘set’ method. The web elements recognized as objects in QTP/UFT, are stored in the object repository. Using the object repository, the object names can be customized as well. As, QTP / UFT provides default names to the objects, while recording.

Observation for UFT Test:

1. UFT records or executes scripts very fast.
2. UFT utilizes the idea of object repository for identifying web element.
3. It generates test result, after running and completing the test, which shows the duration of testing time as well as lines of code, whether it passes or fails.

4.7 Test using WATIR Webdriver**Installing and configuring WATIR:**

For installing WATIR, first of all Ruby needs to be installed. I installed Ruby version of 2.0.0-p645. Then from control panel, systems -> Advanced system settings-> Environment variable -> Path should be appended with the Ruby's current location. The following command needs to be executed in command prompt to install Ruby gems, 'gem update --system'. Finally, to install WATIR Webdriver, 'gem install WATIR-Webdriver' needs to be run.

Scripts in WATIR Webdriver:

Using command prompt, executed following commands:

****To start with Ruby****

irb

****code will be written using WATIR Webdriver****

'require WATIR-Webdriver'

**** Starts a new session in firefox browser****

irb(main):050:0> browser = WATIR::Browser.new :firefox

****WATIR response after opening a new browser session****

=> #<WATIR::Browser:0x.f81a18b4 url="about:blank" title="">

****Redirects to the St Cloud State, Records and registration website****

```
irb(main):051:0> browser.goto("https://stcloudstate.edu/registrar/")
```

****WATIR response: successfully loads the URL ****

```
=> "https://stcloudstate.edu/registrar/"
```

****Clicks to the link of e Services ****

```
irb(main):053:0> browser.link(:text, "eServices").click()
```

```
=> nil
```

****Setting username field for e- Services sign in ****

```
irb(main):054:0> browser.text_field(:id, "userName").set("as8834hl")
```

```
=> ""
```

****Setting password field for e- Services sign in ****

```
irb(main):055:0> browser.text_field(:id, "password").set("Today123")
```

```
=> ""
```

****Clicks to the Login button****

```
irb(main):056:0> browser.button(:value, "Login").click()
```

```
=> nil
```

****Clicks to the link of Courses & Registration ****

```
irb(main):057:0> browser.link(:xpath, "//*[@id='app-links']/ul/li[4]/a").click()
```

```
=> nil
```

****Clicks to the link of Search for a course****

```
irb(main):058:0> browser.link(:xpath, "//*[@id='main']/ul/li[1]/a").click()
```

```
=> nil
```

****Selecting a subject (here it's chemistry) from dropdown box ****

```
irb(main):059:0> browser.select_list(:id, "subject").select("Chemistry (CHEM)")
```

=> "Chemistry (CHEM)"

****Searching the courses for selected subject****

irb(main):060:0> browser.button(:value,"Search >").click()

=> *nil*

****Enters to the search course results page and clicks to the link for 'Forensic Science' course****

irb(main):061:0> browser.link(:xpath,"//[@id='Forensic Science002163']/a").click()*

=> *nil*

****Add course to the wish list****

irb(main):062:0> browser.img(:xpath,"//[@id='main']/table[1]/tbody[2]/tr/td[1]/div/a[1]/img").click()*

=> *nil*

****Setting the alert to handle the dialog box, after adding the course****

irb(main):063:0> browser.alert.ok

=> ""

**** Entering link to -Continue to review my plan ****

irb(main):064:0> browser.link(:xpath,"//[@id='main']/span/a").click()*

=> *nil*

**** Select (click to the checkbox) the course, just added****

irb(main):065:0>

browser.checkbox(:name,"selectCourse_002163_20165_0073_0073_false").set

=> *nil*

****Removing that course from wish list****

```
irb(main):066:0> browser.button(:value,"Remove Selected Course(s) from Wish List").click()
```

```
=> nil
```

**** Clicks to the Logout link ****

```
irb(main):067:0> browser.link(:text,"Logout").click()
```

```
=> nil
```

Explanation of WATIR Webdriver script:

WATIR Webdriver works similarly like Selenium Webdriver. It also uses HTML source codes to locate web elements. The main difference is in the syntax, although the WATIR Webdriver functions in a simpler way [41].

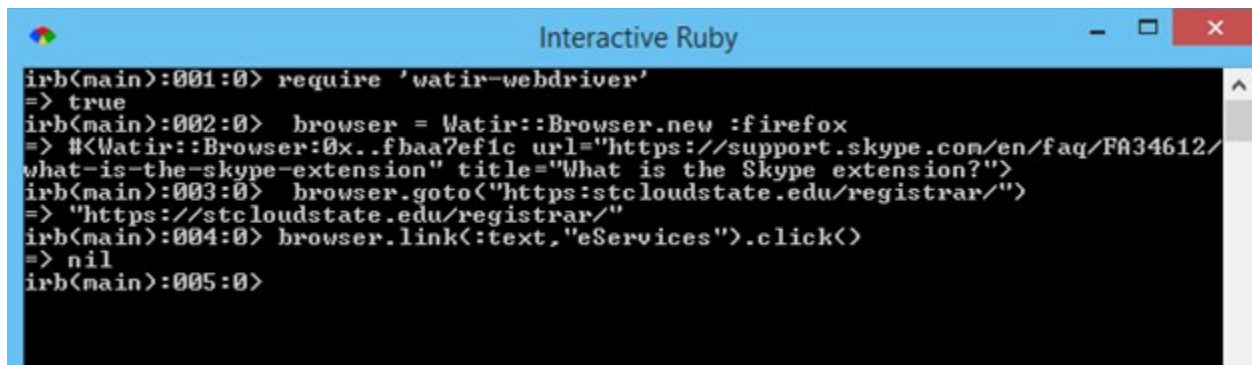
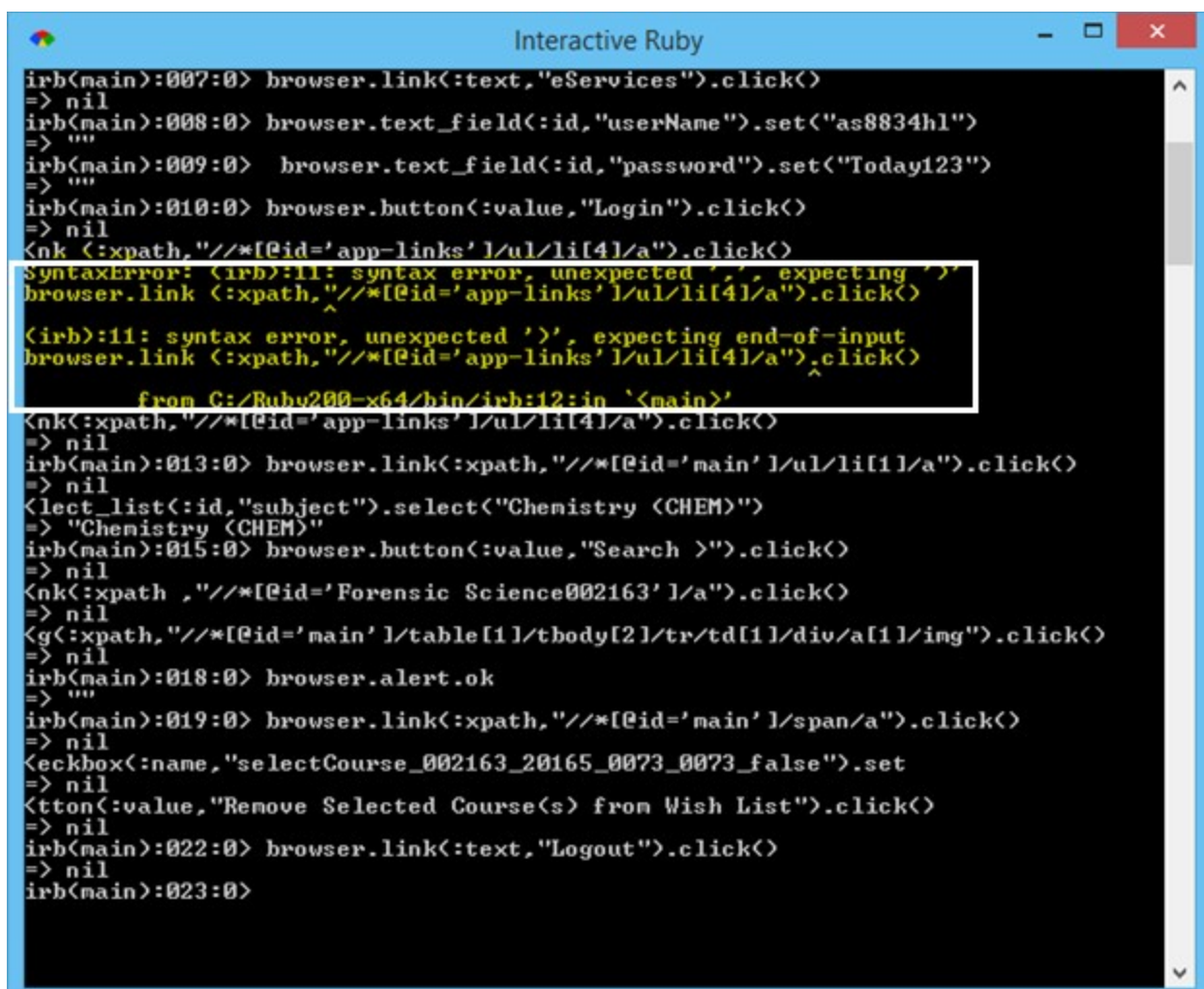
A screenshot of a terminal window titled "Interactive Ruby". The terminal shows a series of commands and their outputs. The first command is `require 'watir-webdriver'`, which returns `true`. The second command is `browser = Watir::Browser.new :firefox`, which returns a `Watir::Browser` object. The third command is `browser.goto("https://stcloudstate.edu/registrar/")`, which returns the URL. The fourth command is `browser.link(:text,"eServices").click()`, which returns `nil`. The terminal prompt is `irb(main):001:0>` through `irb(main):005:0>`.

Figure-13: Start of test scripting with WATIR Webdriver in Ruby command prompt



```

Interactive Ruby
irb(main):007:0> browser.link(:text, "eServices").click()
=> nil
irb(main):008:0> browser.text_field(:id, "userName").set("as8834hl")
=> ""
irb(main):009:0> browser.text_field(:id, "password").set("Today123")
=> ""
irb(main):010:0> browser.button(:value, "Login").click()
=> nil
<nk (:xpath, "//*[@id='app-links' ]/ul/li[4]/a").click()
SyntaxError: (irb):11: syntax error, unexpected ',', expecting ')'
browser.link (:xpath, "//*[@id=' app-links' ]/ul/li[4]/a").click()
^
(irb):11: syntax error, unexpected ')', expecting end-of-input
browser.link (:xpath, "//*[@id=' app-links' ]/ul/li[4]/a").click()
^
from C:/Ruby200-x64/bin/irb:12:in `<main>'
<nk (:xpath, "//*[@id=' app-links' ]/ul/li[4]/a").click()
=> nil
irb(main):013:0> browser.link(:xpath, "//*[@id='main' ]/ul/li[1]/a").click()
=> nil
<lect_list (:id, "subject").select("Chemistry (CHEM)")
=> "Chemistry (CHEM)"
irb(main):015:0> browser.button(:value, "Search >").click()
=> nil
<nk (:xpath, "//*[@id='Forensic Science002163' ]/a").click()
=> nil
<g (:xpath, "//*[@id=' main' ]/table[1]/tbody[2]/tr/td[1]/div/a[1]/img").click()
=> nil
irb(main):018:0> browser.alert.ok
=> ""
irb(main):019:0> browser.link(:xpath, "//*[@id='main' ]/span/a").click()
=> nil
<checkbox (:name, "selectCourse_002163_20165_0073_0073_false").set
=> nil
<button (:value, "Remove Selected Course(s) from Wish List").click()
=> nil
irb(main):022:0> browser.link(:text, "Logout").click()
=> nil
irb(main):023:0>

```

Figure-14: WATIR Webdriver test scripts

Figure-12 shows the initiation of the test using scripting in WATIR Webdriver. The first line in command prompt shows the inclusion of WATIR Webdriver.

Figure-13 shows each line of test script execution in WATIR Webdriver. If the test script runs successfully, it provides a null response (" " or 'nil'). Otherwise, it shows error.

```

checkbox<:name,"selectCourse_002163_20165_0073_0073_false").set
Watir::Exception::UnknownObjectException: unable to locate element, using {:name
=>"selectCourse_002163_20165_0073_0073_false", :tag_name=>"input", :type=>"chec
kbox"}
  from C:/Ruby200-x64/lib/ruby/gems/2.0.0/gems/watir-webdriver-0.8.0/lib/w
atir-webdriver/elements/element.rb:533:in `assert_element_found'
  from C:/Ruby200-x64/lib/ruby/gems/2.0.0/gems/watir-webdriver-0.8.0/lib/w
atir-webdriver/elements/element.rb:505:in `assert_exists'
  from C:/Ruby200-x64/lib/ruby/gems/2.0.0/gems/watir-webdriver-0.8.0/lib/w
atir-webdriver/elements/checkbox.rb:27:in `set?'
  from C:/Ruby200-x64/lib/ruby/gems/2.0.0/gems/watir-webdriver-0.8.0/lib/w
atir-webdriver/elements/checkbox.rb:18:in `set'
  from (irb):18
  from C:/Ruby200-x64/bin/irb:12:in `<main>'
checkbox<:name,"selectCourse_002163_20165_0073_0073_false").set
=> nil

```

Figure-15: Error while testing with WATIR Webdriver

Observation for WATIR Webdriver:

1. I used command prompt for executing WATIR Webdriver test code. In the command prompt, at a time each line of test code is executed instead of executing the whole code, which provides better efficiency.
2. In case of any error or test script fails to execute, the testing does not need to restart the testing from the beginning (see in Figure-13). Once the issue is resolved it starts executing subsequent test cases.
3. There are actually not much online support or resources available for testing with WATIR.
4. Several times I encountered an error (**WATIR::Exception::UnknownObjectException:unable to locate element**) for using either incorrect HTML attributes name (id, name, value) or non-matching HTML elements (Link, button, text_field) on the webpage (see Figure-14). WATIR is very specific about spacing in the code. The test might fail because of more/less spacing while executing a command.

5. If the source code of web application is written in HTML, then WATIR Webdriver can be used for test automation.

4.8 Comparison among Selenium, WATIR, and UFT/QTP

The key features among the testing tools have been compared and the result is summarized in the following table (Table-8). Most of the comparison criteria encompassed here are based on available features of the testing tool as well as some of the criteria are also based the test results.

Table 8: Comparison among Selenium-WATIR-UFT/QTP

Criteria	Selenium	WATIR	QTP/UFT
Browser	Multi-Browser support, such as Internet Explorer (IE), Firefox, Chrome, Opera, Safari.	WATIR-Webdriver supports Chrome, Firefox, Internet Explorer(IE) and Opera	QTP supports only older versions of IE, Firefox, chrome and safari. UFT supports the latest versions of IE and some of the Firefox and chrome browsers.
Programming language support	Java, Ruby, Perl, Python, and C#.	Ruby	Visual Basic scripting
OS Support	Windows, Mac, Linux.	Windows, Mac, Linux.	All Windows; Also UFT supports the latest windows 10.

Table 8 Continued

Criteria	Selenium	WATIR	QTP/UFT
Framework support	C# : Nunit Java: JUnit/TestNG Python: pyunit, py.test, robot Javascript:WebdriverJS, WebdriverIO, Nightwatch JS	Cucumber, RSpec, Test/Unit	Data Driven automation framework Keyword Driven automation framework
Pop-up support	Selenium has built-in support for pop-up dialog boxes.	Pop-ups are accessible using API	Support testing with pop-up dialog box.
Identification of web elements	Locates web element from HTML source code using a link, text, checkbox, XPath etc. and attributes (id, name, value).	Uses the same method as Selenium Webdriver.	Recognizes everything as an object (uses object repository) on the webpage.
Installation process	Selenium IDE is just a Firefox plug-in, so it's simple and quick; Selenium web driver requires JDK (for java) and IDE installed.	Configuring WATIR web driver using ruby command prompt is simple, requires prior installation of ruby.	Install process allows to include required add-ins for testing. Installation is quick and simple.

Table 8 Continued

Criteria	Selenium	WATIR	QTP/UFT
Script execution speed	Depends on wait method. Executes too fast, if wait is not used.	Slower than Selenium Webdriver and UFT	Depends on wait method. Executes too fast, if wait is not used.
Ease of use	Easy to get started with Selenium IDE, because it provides the recording of test. However, Selenium Webdriver requires programming skills.	Basic level of scripting is not too difficult with WATIR.	Easy to get started with recording and playback mode of the tool.
Learning Curve	Learning curve rises high, switching from Selenium IDE to Selenium Webdriver. Good programming knowledge required to create test scripts in Webdriver.	Learning curve is not as high as Selenium Webdriver / UFT. Good knowledge of Ruby is required for advanced level.	Deep learning curve; As it requires test code creation in Visual Basic scripting, for both windows and web based applications.
Type of Web application support	Supports HTML, flash contents in web the application.	WATIR supports only the web applications, built with HTML code.	HTML based web applications and Flex add-in for supporting flash.

Table 8 Continued

Criteria	Selenium	WATIR	QTP/UFT
Parallel executions	Supports parallel execution; with the concept of Selenium grid.	It uses WATIR Grid, which supports parallel execution of water test cases by using threads.	Supports test execution on multiple browsers at the same time, in only one machine.
Types of testing done	Mostly functional and regression	Mostly regression and functional.	Functional and regression.
Application support	Web and mobile Applications	Web and mobile Applications	Windows, web and mobile applications
Cost of using	Free of cost, as its open source	Free of cost, as its open source	Costly, as having the only commercial version. And a 30-day trial version available for free.
Different mobile device support	Two major mobile platform iOS and Android	Android and iOS device support	Supports iOS, Android, blackberry and windows phone
Support for file upload	Supports file uploading, during test	Supports file uploading during test	QTP/UFT provides file uploading with HP Quality Center tool (a test management tool)

Table 8 Continued

Criteria	Selenium	WATIR	QTP/UFT
Database Support	Selenium can't connect with the database by itself. Depends on the programming language it's using if that supports database; For example, Java has JDBC API, to connect with database (requires JVM to run in system)	With the help of ruby, it can connect to database	With the help of Visual Basic script and ODBC, it can connect to the database.
Test result generation	Selenium Webdriver can generate test result by integrating with other testing frameworks (such as TestNG).	With the help of RSpec framework WATIR can generate HTML report of test results.	After each test execution, QTP/UFT provides test run results, which shows, whether each line of test script passes/ fails as well as the duration of test execution.

Chapter 5: CONCLUSION

Traditionally, automated and manual testing are considered as different and separate approaches are used for their execution. In reality, they depend on each other, as the limitation of one is addressed by the other one. Manual testing can be useful for finding bugs in special cases where the requirement changes continuously, and situations where automated tests might not be the most effective. However, test automation has many advantages such as repeatability, consistency, better and effective handling of test cases (for situations where a large number of test cases need to be executed). The core difference between manual and automation testing is that test automation is most appropriate for the situation where repetitive work needs to be done (re- testing with same or different test data but the same test script). But it cannot eliminate all of the bugs in the application without the help of manual testing. So, automated tests are good at breadth but not much at depth [47].

In this paper, the analysis and comparison are made among different automated testing tools (Selenium IDE, Selenium Webdriver, WATIR and UFT/QTP) on various quality factors. After the overall analysis, it is not easy to rank these tools based on a number of factors only. Selenium provides the freedom to work with all types and almost all the versions of browsers, operating system as well as flexibility to choose one among many programming languages. However, its access is limited to web applications only. Another automation tool WATIR has become also popular nowadays. It also supports almost all the browsers, but it lacks record and playback functionality, which could be a great functionality to get started with for beginners. Similar to Selenium, it also does not provide testing for windows applications. Finally, when it comes to UFT/QTP, it works well with both web and windows applications. It has a built-in mechanism of object identification as well as options to work with different add-ins. It also integrates with

Application Life cycle Management (ALM) tools (effective in managing several phases of SDLC). On the other hand, QTP is not compatible with latest versions of browsers and OS, although UFT supports all the latest versions of Internet Explorer and most of the Chrome and some of Firefox browsers as well as it performs API testing. The biggest limitation with UFT/QTP is licensing cost, which is too high.

Finally, all these tools have advantages, limitations as well as utilization for some certain types of testing, based on the scope of applications. None of these tools is absolutely perfect or best. Although, in recent years, Selenium has been preferred by most software industries and is used more than other tools by testers or developers. Selenium's variety of features and functionalities, ability to integrate with various frameworks, its number of different programming languages, and its free cost provides major advantages. However, not all the applications are web-based. For that reason, there will always be demand for UFT or tools which are able to perform testing in both web and windows based applications. Moreover, WATIR has been gradually overcoming its limitations and gaining popularity within the industries.

While concluding this research, I learned that, each software testing tool has its own distinguishing features and it's a matter of time to learn each different tool and know how to utilize the tools for automation testing. It also takes a lot effort to find out the tool that works best to meet the goal of testing. From my own perspective, an ideal tool should meet at least some criteria. First, the installation process should be simple and quick. Second, getting started with the tool for novices should not be too difficult and there should be adequate learning material available for the tool, which helps to create and execute a basic set of test scripts. Finally, the tool should be friendly to work with as well as it should be able to generate an overall test result

(pass/fail). So, if the testing fails, it makes easily understandable logs to troubleshoot, which saves a lot of time.

For future work, my goal is to extend this research by including a few more testing tools to analyze and compare. Then the selected tools should have latest features.

References

- [1] X. Wang and G. He, “The research of data-driven testing based on QTP,” *Computer Science & Education (ICCSE), 2014 9th International Conference on*. pp. 1063–1066, 2014.
- [2] M. Monier and M. M. El-mahdy, “Evaluation of automated web testing tools,” vol. 4, no. 5, pp. 405–408, 2015.
- [3] K. M. Mustafa, R. E. Al-Qutaish, M. I. Muhairat, P. S. K. and D. N. S. S. Rao, W. Xinbian, and H. Guangjun, “Classification of Software Testing Tools Based on the Software Testing Methods,” in *Computer Science & Education (ICCSE), 2014 9th International Conference on*, 2014, vol. 3, no. 7, pp. 1063–1066.
- [4] G. Saini, “Software Testing Techniques for Test Cases Generation,” vol. 3, no. 9, pp. 261–265, 2013.
- [5] “Software Testing Tutorial,” *www.tutorialspoint.com*. [Online]. Available: http://moodle.nccu.edu.tw/pluginfile.php/77731/mod_resource/content/1/software_testing%281%29.pdf. [Accessed: 20-Jan-2016].
- [6] “SOFTWARE TESTING on WordPress.com.” [Online]. Available: <https://softwaretestingupdates.wordpress.com/>. [Accessed: 15-Feb-2016].
- [7] R. Gupta, *Test Automation and QTP: QTP 9.2, QTP 9.5, QTP 10.0 and Functional Test 11.0*. Pearson Education India.
- [8] T. Bharti and E. Vidhu, “Functionality Appraisal of Automated Testing Tools,” vol. 3, no. 1, pp. 129–134, 2015.
- [9] K. Bahl, “Software Testing Tools & Techniques for Web Applications,” no. 5, pp. 315–318, 2015.

- [10] S. Jagannatha, M. Niranjnamurthy, M. Sp, and C. Gs, “Comparative Study on Automation Testing using Selenium Testing Framework and QTP,” vol. 3, no. 10, pp. 258–267, 2014.
- [11] Vishawjyoti* and S. Sharma, “STUDY AND ANALYSIS OF AUTOMATION TESTING TECHNIQUES,” *J. Glob. Res. Comput. Sci.*, vol. 3, no. 10, pp. 2010–2013, 2012.
- [12] N. Gogna, “Study of Browser Based Automated Test Tools WATIR and Selenium,” *Int. J. Inf. Educ. Technol.*, vol. 4, no. 4, pp. 336–339, 2014.
- [13] H. Kaur and G. Gupta, “Comparative Study of Automated Testing Tools : Selenium , Quick Test Professional and Testcomplete,” vol. 3, no. 5, pp. 1739–1743, 2013.
- [14] R. Angmo and M. Sharma, “Performance evaluation of web based automation testing tools,” *2014 5th Int. Conf. - Conflu. Next Gener. Inf. Technol. Summit*, pp. 731–735, 2014.
- [15] T. J. Naidu, N. A. Basri, and S. Nagenthram, “SAHI vs. Selenium: A comparative analysis,” *Proc. 2014 Int. Conf. Contemp. Comput. Informatics, IC3I 2014*, pp. 967–970, 2014.
- [16] A. Jain, M. Jain, and S. Dhankar, “A Comparison of RANOREX and QTP Automated Testing Tools and their impact on Software Testing,” no. 1, pp. 8–12, 2014.
- [17] Y. Kumar, “Comparative Study of Automated Testing Tools : Selenium , SoapUI , HP Unified Functional Testing and Test Complete,” vol. 2, no. 9, pp. 42–48, 2015.
- [18] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*. McGraw-Hill Higher Education, 2010.
- [19] U. Eriksson, “Test design techniques explained #1: Black-box vs White-box Testing,” *Jun. 08, 2015*. [Online]. Available: <http://reqtest.com/testing-blog/test-design-techniques->

- explained-1-black-box-vs-white-box-testing/. [Accessed: 18-Jan-2016].
- [20] P. V. N. Maurya and E. R. Kumar, “Analytical Study on Manual vs . Automated Testing Using with Simplistic Cost Model,” vol. 2, no. 1, pp. 23–35, 2012.
- [21] T. Xie and D. Notkin, “Improving Effectiveness of Automated Software Testing in the Absence of Specifications,” *2006 22nd IEEE Int. Conf. Softw. Maint.*, pp. 355–359, 2006.
- [22] “10 Tips you should read before automating your testing work,” *www.softwaretestinghelp.com*. [Online]. Available: <http://www.softwaretestinghelp.com/10-tips-you-should-read-before-automating-your-testing-work/>. [Accessed: 21-Jan-2016].
- [23] “QTP Introduction,” *www.tutorialspoint.com*. [Online]. Available: http://www.tutorialspoint.com/qtp/qtp_overview.htm. [Accessed: 21-Jan-2016].
- [24] “Software Testing Tools list,” *www.softwaretestingclass.com*. [Online]. Available: <http://www.softwaretestingclass.com/software-testing-tools-list/>. [Accessed: 21-Jan-2016].
- [25] “Selenium Documentation: Test Automation for Web Applications,” *www.Seleniumhq.org*. [Online]. Available: http://www.Seleniumhq.org/docs/01_introducing_Selenium.jsp. [Accessed: 21-Jan-2016].
- [26] “‘check’, ‘unchecked’, ‘checkAndWait’ and ‘uncheckedAndWait’ command example of Selenium IDE,” *Software testing tutorials and automation*. [Online]. Available: [http://software-testing-tutorials-automation.blogspot.com/search/label/check command](http://software-testing-tutorials-automation.blogspot.com/search/label/check%20command). [Accessed: 21-Jan-2016].
- [27] “Selenium 1 (Selenium RC) — Selenium Documentation,” *www.Seleniumhq.org*. [Online]. Available: http://www.Seleniumhq.org/docs/05_Selenium_rc.jsp. [Accessed: 21-

- Jan-2016].
- [28] “Selenium-Grid — Selenium Documentation.” [Online]. Available: http://www.Seleniumhq.org/docs/07_Selenium_grid.jsp. [Accessed: 07-Mar-2016].
- [29] “TOOLSQA | Selenium Webdriver,” *www.toolsqa.com*. [Online]. Available: <http://toolsqa.com/Selenium-Webdriver/>. [Accessed: 21-Jan-2016].
- [30] “What is Selenium Webdriver?” [Online]. Available: <http://scraping.pro/what-is-Selenium-Webdriver/>. [Accessed: 21-Jan-2016].
- [31] P. Malhotra, “Test Automation Tool Comparison – HP UFT/QTP vs. Selenium.”
- [32] “Selenium – Some Advantages and Disadvantages of the Tool | Application Software Testing on WordPress.com.” [Online]. Available: <https://qaandtestingservices.wordpress.com/2014/07/17/Selenium-some-advantages-and-disadvantages-of-the-tool/>. [Accessed: 21-Jan-2016].
- [33] “Obvious reason to move from Selenium RC to Webdriver.? - Stack Overflow.” [Online]. Available: <http://stackoverflow.com/questions/10779571/obvious-reason-to-move-from-Selenium-rc-to-Webdriver>. [Accessed: 21-Jan-2016].
- [34] “Selenium Webdriver Tutorials - Basic Action Commands And Operations With Examples.” [Online]. Available: <http://software-testing-tutorials-automation.blogspot.com/2014/01/Selenium-Webdriver-tutorials-basic.html>. [Accessed: 21-Jan-2016].
- [35] P. Bindal, C. Science, and P. College, “Test Automation Selenium Webdriver using TestNG,” vol. 3, no. 9, pp. 18–40, 2014.
- [36] P. Station and L. Controller, “QuickTest Professional User ’s Guide.”
- [37] “QTPWorld: Object Repository.” [Online]. Available:

- <http://www.qtpworld.com/index.php?cid=59>. [Accessed: 21-Jan-2016].
- [38] “Want to know key features of QTP at a Glance.” [Online]. Available: <http://www.softwaretestinggenius.com/want-to-know-key-features-of-qtp-at-a-glance>. [Accessed: 21-Jan-2016].
- [39] M. Kaur and R. Kumari, “Comparative Study Automated Testing Tools: TestComplete and QuickTest Pro,” *Int. J. Comput. Appl.*, vol. 24, no. 1, pp. 1–7, 2011.
- [40] N. Gogna, “COMPARATIVE STUDY OF BROWSER BASED OPEN SOURCE TESTING TOOLS WATIR AND WET,” *Int. J. Comput. Sci. Eng.*, vol. 3, no. 5, pp. 1910–1923, 2011.
- [41] “Read WATIRways.” [Online]. Available: <https://leanpub.com/WATIRways/read#leanpub-auto-unknownobjectexception---unable-to-locate-element>. [Accessed: 21-Jan-2016].
- [42] J. K. and P. Rogers, “Test automation of Web applications can be done more effectively by accessing the plumbing within the user interface. Here is a detailed walk-through of WATIR, a tool many are using to check the pipes.” [Online]. Available: http://www.kohl.ca/articles/WATIR_works.pdf. [Accessed: 21-Jan-2016].
- [43] “WATIR-Webdriver: Control the Browser.” [Online]. Available: <http://www.sitepoint.com/WATIR-Webdriver-control-browser/>. [Accessed: 21-Jan-2016].
- [44] “Frameworks.” [Online]. Available: <http://WATIR.com/frameworks/>. [Accessed: 21-Jan-2016].
- [45] “Module: Test::Unit (Ruby 1.8.7).” [Online]. Available: <http://ruby-doc.org/stdlib-1.8.7/libdoc/test/unit/rdoc/Test/Unit.html>. [Accessed: 21-Jan-2016].
- [46] R. Rattan and Shallu, “Performance Evaluation & Comparison of Software Testing Tool,”

vol. 3, no. 7, pp. 711–716, 2013.

- [47] A. Leitner, H. Ciupa, B. Meyer, and M. Howard, “Reconciling manual and automated testing: The AutoTest experience,” *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, 2007.