Natural Language based Test Automation Model for Web Applications

S.P.S. Sendanayaka

2021

Natural Language based Test Automation Model for Web Applications

A dissertation submitted for the Degree of Master of Computer Science

S.P.S. Sendanayaka

University of Colombo School of Computing

2021

DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Student Name: S.P.S. Sendanayaka

Registration Number: 2017/MCS/075

Index Number:17440755

_____Piyumi 29/11/2021_____

Signature of the Student & Date

This is to certify that this thesis is based on the work of Mr. /Ms. _____ under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by,

Supervisor Name: Dr. Jeevani Goonatillake

___J. Goonetillake_____

Signature of the Supervisor & Date 29/11/2021

I would like to dedicate this thesis to...

To my family…

ACKNOWLEDGEMENT

My undivided gratitude for the following people who guided me, encouraged by and supported me to complete this project successfully.

- Dr. Jeevani Goonatillake, my project supervisor for the continuous guidance and support I received from the very beginning itself.

- Dr. Lasanthi De Silva, for suggestions and guidance in implementing the automation model.

- The QA Engineers and domain experts who participated in the requirement elicitation survey and evaluation of prototype for contributing their valuable time and effort to this project.

- To my family and friends for always supporting me to the through the good and bad days alike.

ABSTRACT

Test Automation is a latest technology in the software industry. People favor the test automation over the manual testing due to significant advantages such as, automation is faster than the manual testing, reduction of the human intervention, quality and improve the accuracy. However, the initial cost of the test automation is high. The main reason for this is the lack of technical expertise required for test automation within the industry. This results to organizations have to invest time, money and effort to train the less technical resources. In current, organizations are highly concerned with finding means of resolving this issue.

The proposed model addresses this issue by introducing a Natural Language Processing based test automation model for web applications which can be used for non-technical resources with the minimum technical expertise. It allows a user to enter test steps in English. The model is capable of processing the test steps entered by the user and converting them into automation scripts which can be executed using the automation framework that is built. Furthermore, result reports can also be obtained by the execution of generated scripts. Overall accuracy of the new model has been determined to be 84.33 %.

Keywords:
Web Applications, Test Automation, Test Automation Framework, Natural Language Processing, Quality Assurance

TABLE OF CONTENTS

Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1 **INTRODUCTION**

1.1 Motivation

Software test automation has become more popular in the software testing as it has many advantages than manual testing. Test automation is the practice of running tests automatically, managing test data, and utilizing results to improve software quality. Most of the organizations in the IT industry advice to their QA engineers to learn and move to the test automation. For test automation requires specific scripting knowledge and engineers have to practice it. An organization might have technical engineers and as well as non-technical engineers. If an engineer who has the technical knowledge, he or she can easily move to the test automation.

However, this task might be bit hard to non-technical engineer due to lack of technical knowledge. This is the point where the problem occurs. That is, **how do we carry out the test automation with such non-technical resources and this project aims to find a solution to this problem**.

1.2 Statement of the problem

Testing is an important stage in the software development life cycle. However, when considering the past decade, manual testing of software has dominated on software quality assurance. “Software testing is a costly and time-consuming activity in the software development life cycle” (Rankin, C, 2002). It had been observed that writing testing code costs a lot and also takes as much time as it takes to develop the software product (Li & Wu, 2004). Organizational spending on quality assurance of applications has rapidly increased over the past years. (World Quality Report, 2014-2015). Therefore, people favor the test automation because of its many advantages, mainly the reduction of human intervention and the ability to reuse the same tests over and over again using the simple execution of pre-defined scripts at minimum cost and significantly faster than the manual testing.

Recent research has indicated that QA and Testing budgets contain the most part of labor costs of an organizations. As of 2015, 35% of an average organizational QA budget was allocated for human resources while 33% was allocated for hardware and infrastructure and 32% was allocated for tools (World Quality Report, 2014-2015). The main reason why most part of the budget is assigned for individual is that in order to automate a process, automation engineers should have experience in the scripting language used by the tool and also need to have

specialized knowledge in the automation tool. Organizations spend time, money and effort on training human resources in using these different tools for test automation. This demand for specialized resources for test automation conflict with the initial purpose of test automation which is to reduce the cost and duration of the quality test cycle. Software companies are highly concerned with developing cost-effective test automation tools to overcome this problem.

The purpose of this research is to explore a potential solution for this problem. The proposed solution would be a test automation model for web applications that can be used by any non-technical person with sufficient knowledge regarding an automation scenario. This model would use natural language processing technique to extract actions in the web-based scenario given by the user. Then map the actions with the UI object repository input by the user as a data source and generate an automation project accordingly. The generated project contains automation scripts that can then be executed to generate a result report.

The new model does not require any specialized technical knowledge. This reduce the complexity would save time and minimize initial cost of test automation, otherwise organizations have to spend significant cost to train the human resources who are non-technical for scripting purposes.

1.3 Research Aims and Objectives

1.3.1 Aim

The aim of the project is, ***to design, develop and evaluate a test automation model for web applications which is capable of generating automation scripts by extracting user actions from test scenarios input in natural language (English), to be used by any non-technical resource.***

1.3.2 Objectives

The objectives of the proposed model are as follows,

- Makes the test automation easier to the less technical engineers by providing script less automation.
- Reduce the time and cost of the organizations in order to train the less technical engineers to do the automation.

1.4 Scope

The proposed model will handle UI automation scenarios which are defined by the QA engineers by English language.

1.5 Structure of the Thesis

The final report has been structured as follows,

- Chapter 1 – Introduction

This chapter consists of a basic introduction to the problem domain and the proposed solution. It includes the project aim, scope and objectives. The features of the prototype and the resource requirements have also been discussed.

- Chapter 2 - Literature Review

This chapter consists a literature review of research articles and related documents regarding the problem background, tools, technologies and related work carried out pertaining to the problem domain.

- Chapter 3 – Methodology

This chapter consists the design of the proposed model and the implementation.

- Chapter 4 - Evaluation and Results

This chapter consists of the evaluation of the project using various evaluation techniques.

- Chapter 5 – Conclusion and Future work

This chapter provides the concluding remarks including the limitations of the project, challenges faced, leaning outcomes and the future enhancements.

Chapter 2 - LITERATURE REVIEW

2.1 Literature Review

2.1.1 Manual Testing vs Test Automation

Software testing is a concept that continues to evolve in the software development industry. As Burnstein (2003) points out, the process of software testing is utilized in determining defects and faults in software products in order to guarantee that a particular product meets its standards as expected with regards to pre-defined attributes. These attributes can be the functional and non-functional requirements specified at the beginning of the development of a particular software. As the complexity of software applications is increasing, it is important to maintain quality of software at a scale parallel to the complexity level. The goal of quality assurance of a software can be achieved in two different ways, either by manual testing or through test automation (Mayer, 1998)

Manual Testing is defined as a process through which quality assurance engineers (QA engineers) execute manual test cases with the objective of detecting bugs or defects by comparing the expected and actual outcomes of the software (Herath et.al, 2015). The IEEE Standard 829 (1983) defines a test case as a document "specifying inputs, predicted results and a set of execution conditions for a test item". In manual testing, QA engineers create manual test cases using test scenarios identified from system requirements and use cases. Although this method of testing is deemed fit as a cost effective and more reliable method of testing on a smaller scale (Base36.com, 2016), the error count is high when practiced on a larger scale owing to the fact that it requires manual input, analysis and evaluation. Torkar (2006) points out that humans often get tired of repetitive processes. This caused the inclination of the software testing trend towards automated testing (Torkar, 2006).

TA is preferred over manual testing as means of overcoming the complications mentioned above. People have defined the TA process in different ways. TA as defined by Herath et al. (2005) is the process of executing pre-defined scripts to generate result reports in order to find defects in software and to determine software quality. If the actual and expected outcomes of a program align then the program can be certified as bug-free. Dustin et.al (1999) outlines TA as a process where the testing activities are automated which includes the development of test

cases, execution and verification of the test scripts through the use of automated tools. Gao et al. (2003) highlights that software TA refers to the effort put into automating operations of the testing process of software through distinct policies and strategies. Organizations opt to adopt TA with the common objectives of reducing delivery time by optimizing the test cycle, detect bugs in the early stages of the SDLC, increase test coverage and accuracy while reducing the cost of testing.

It is difficult however to conclude which out of the two software testing methods is better. Rather, it is about determining the ideal scenario for either manual testing or automated testing. Berner et.al (2005) announces an interesting point regarding this matter in the paper Observations and Lessons Learned from Automated Testing. It states that the cost effectiveness of TA is directly related to the number of times a test suite is executed. Additionally, based on an interview conducted by Asfaw (2015) with two experts in the namely Samuel, a Computer Scientist and Hanip, a professor of a Technology Institute, Samuel identifies a good candidate for automated testing to be repetitive, recurring, tedious and likely to cause errors if tried by manual means. Based on that remark, web applications can be viewed as ideal candidates for TA because testing of web applications tend to be a very long, tedious, repetitive and error-prone task. Hanip adds to Samuel stating that a test case which is very lengthy or in need of higher accuracy would also count as a proper candidate for TA. These factors can be used in establishing whether a test case needs to be automated or if it could be tested using manual methods.

2.1.2 Test Automation and Web Applications

A web application is defined as an application program that has content stored remotely in a separate server and distributed over the Internet through a browser using a Graphical User Interface (GUI) (SearchSoftwareQuality, 2016). Web applications have continued to become more and more complex over the past years. QA Labs Inc. (2000) had noted that there is a wide range of refinement in web applications from a typical company website with a simple structure to applications like eBay and Amazon with complex search engines and e-commerce facilities. Increasing reliance on complex web-based solutions demand for the usage of proper methodologies and guidelines in the development of applications in order to guarantee on time delivery within the allocated budget with a high level of quality. However, despite being one of the fastest growing classes of software systems in the present, (Arora & Sinha, 2012) web applications have continued to degrade in quality.

QA Labs Inc. (2000) states the main reason to the reduction in quality is the over-pressurized deadlines that web developers are required to meet with. Unlike traditional development of any other software, the go-to market time of web-based applications is comparatively low. Furthermore, QA Labs Inc. discusses how system requirements of a web application are subject to change throughout the development cycle because most clients are not satisfied even with their own initial requirements once implemented. Several adjustments, especially to the GUI of the web application, would have to be made before launching the application. Manual testing of such applications with constant requirement alterations within limited timeframes can be a difficult and costly task.

Mayer (1998) and Pepe (2000) insists that a relationship exists between utilization of TA strategies and quality assurance of web applications. This stems from the fact that the many benefits of automation can be applied in web application testing domain in order to optimize time consumption and cost of determining quality. For example, the reduction of test cycle duration which is a benefit of TA stands out as an ideal solution in dealing with the tight timelines of web application development. Furthermore, the reusability of test scripts, reduction of test costs and the increased test coverage are other means of improving quality of web applications while overcoming the challenges that exist in the particular domain.

This infers that any automation tool chosen for the purpose of TA in web applications should primarily have the features that highlight the benefits of TA mentioned above.

2.1.3 Challenges in Test Automation

Although TA proves to be beneficial and continues to evolve, organizations seem rather reluctant to adopt it completely. The World Quality Report 2015-16 (Muthukrishnan & Margaliot, 2015) reveals that only 45% which is less than half of overall test cases have been automated by organizations by 2015. Similarly, Bogdanov (2015) observes that only 49% of 600 participants inclusive of QA leads, project managers, QA engineers and automation engineers on a survey pertaining to research on automated software testing trends actively used automated testing. The following challenges as illustrated in Figure 2.1 depict the common challenges according to The World Quality Report 2015-16 that discourage organizations from TA.

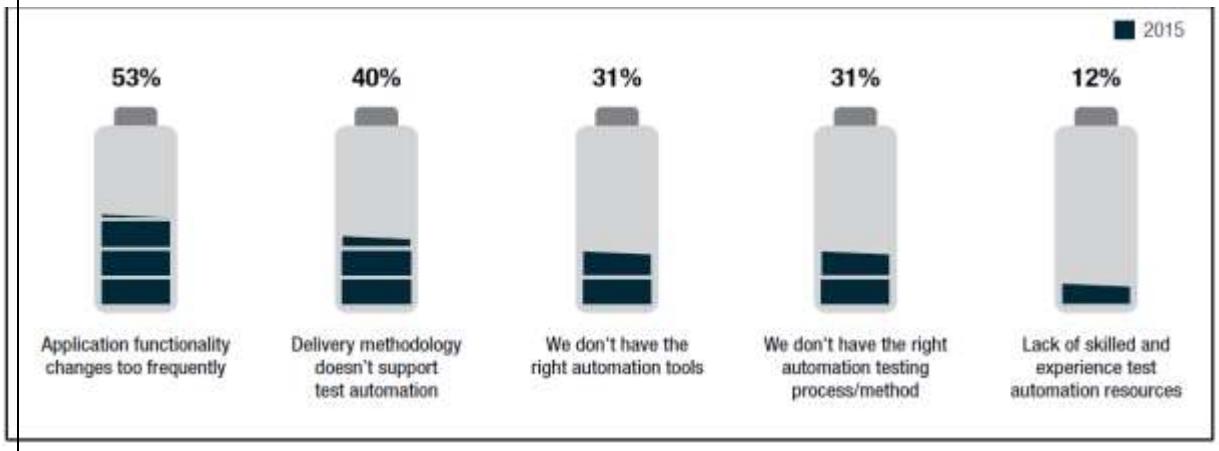


Figure 2.1 Test Automation Challenges (World Quality Report, 2015-2016)

An article by Optimus Information (2015) highlights TA challenges as follows,

- Unfeasible estimations such as time saved through TA
- Cost of automation tools and the time spent specializing on them
- Typical QA teams not having sufficient technical knowledge required in scripting
- Availability of a lot of unreliable automation tools with bugs in the current software market

Further research on related articles, surveys, questionnaires and other posts supported in isolating three common challenges in TA.

1. High initial and maintenance cost

   TA is a long-term investment for organizations. The initial cost of adopting TA tools and methods require a significant amount of money. Script maintenance and constant tool updates can also prove to be costly. However, with time, this cost can be easily recovered through automation.
2. Lack of skilled resources with technical experience

   TA engineers need to be familiar with automation scripting. They should also have specialized skills using different kinds of automation tools available in the market today. Most tools available, commercial and open source alike, requires a significant amount of scripting knowledge (coding) to carry out the functional tests and the testing of custom controls of the application. It would be optimal for organizations if they could find a tool, which requires minimal or no coding at all, that can be used by their manual testers on need basis.

3. Lack of reliable automation tools

   Since there is no particular standard for a software test automation tool, engineers tend to use various approaches in developing frameworks for TA. This often results in poor quality automation tools which are highly unreliable and require a lot of maintenance effort.

Evidently the TA market needs a tool that would overcome these challenges and promote automated testing as means of keeping up with the increasing delivery pace of software applications.

2.1.4 Lack of Technical expertise as a challenge

In addition to the sources mentioned above which highlight the challenges of TA, The World Quality Report 2014-15 (Muthukrishnan & Margaliot, 2014) highlights that as of 2014, 35% of the average organizational QA budget is allocated for acquiring and training human resources. This is because automation tools available in the industry today are complex and require programming knowledge in designing and developing test cases. (Fecko and Lott, 2002) mentions that TA demands the following set of skills,

a) Ability to utilize TA tools,
b) Software design and development skills,
c) Understanding on the application under test

in order to successfully automate a test case as expected. Upon analyzing the TA tools in the market, it is notable that most tools agree with Fecko's observation. Automation tool vendors often conduct training programs on purchase about using the tool as means of overcoming this problem. The next section focuses on exploring and evaluating more reliable and efficient means of overcoming the challenge of technical expertise and skills in automation tools, particularly in the web application domain, in order to promote TA within the industry.

2.2 Test Automation Approaches

2.2.1 Introduction to Test Automation Approaches

A test automation framework is essentially a combination of a set of tools, concepts and assumptions that build up an environment for the execution of automated tests. In other words,

it is a platform which facilitates the test automation process. There are several approaches to test automation frameworks today. Some of the common approaches are,

1. Modularity-driven approach
2. Data-driven approach
3. Keyword-driven approach
4. Hybrid approach

These approaches are further investigated and evaluated in order to find the most suitable approach in order to address the research problem.

2.2.2 Modularity-driven Approach

Modularity-driven approach for test automation is based on the concepts of encapsulation and abstraction. In order to achieve higher level of modularity, separate test scripts are written for each separate module of the application that is to be tested (Kelly, 2003). These modules are then organized in a hierarchical structure similar to what is shown in Figure 2.2 below. This arrangement ensures that each separate module is independent and modifications carried out on one module does not affect the other modules.

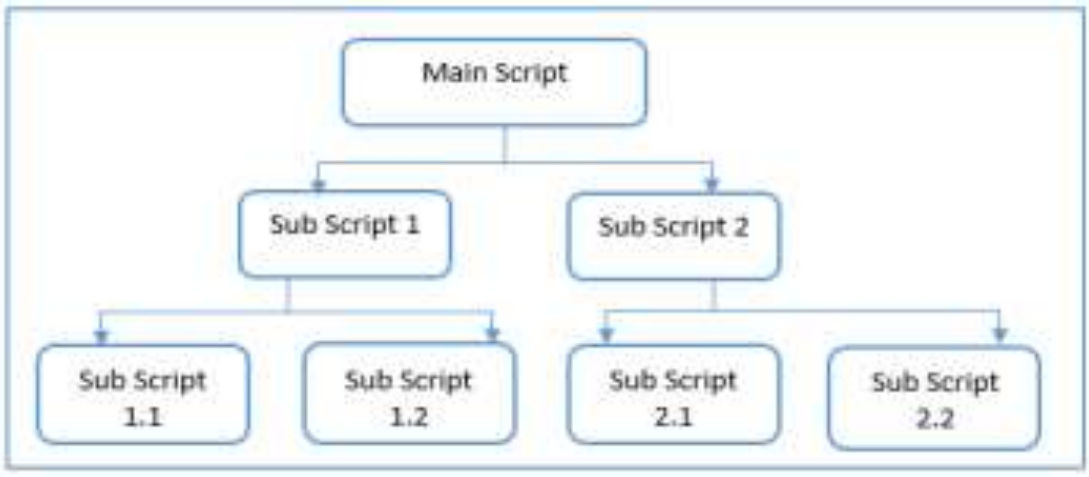


Figure 2.2 Architecture of Modularity Driven Approach

2.2.3 Data-driven Approach

Originally, test scripts used to have the test input and output values, also known as test data, hard-coded into them. This led to several problems. For example, a slight change in test data required modification of the whole script (Laukkanen, 2006). If it was a rather lengthy or unstructured script this would prove to be difficult. Another problem was that for a slightly varying data set, separate scripts were needed. This was an unnecessary effort in automation.

As a solution for these problems, test data was separated from scripts and are stored in external data sources such as databases, Excel sheets and CSV files. The execution of tests by reading data from the external sources are coded in the test script. In this framework approach, scripts act as drivers that simply retrieve data, execute the code and validate the results against the outputs that are specified in the external sources (Kelly, 2003). Figure 2.3 presents an example of an excel file which contains such externalized data.

	A	B	C	D	E
1	Test Case	Input 1	Input 2	Operator	Expected Output
2	tc001	2	5	+	7
3	tc002	2	0.4	-	1.6
4	tc003	6	3	/	2
5	tc004	3	3	*	9
6	tc005	-1	5	+	4
7					

Figure 2.3 Sample Test Data File for a calculator program

2.2.4 Keyword-driven Approach

Kelly (2003) notes that this approach for automation frameworks is independent of application and automation tools on which the automation is carried out. Furthermore, she studies that the keyword approach is very similar to a manual test case. This means that this approach can be used by both manual and automation test engineers easily.

First step of keyword-driven approach is to isolate a set of keywords from each test case. Then, corresponding functionalities are defined for each keyword. This is generally done using an Excel sheet and function-libraries (Sangave and Nandedkar, 2014). The Excel sheet consists of a table which represents each manual test case. The function libraries contain the implementation required to invoke the action represented by a specific keyword using an automation tool. It is similar but more advanced when compared with the data-driven approach which is why (Fewster and Graham, 1999) points out keyword-driven approach as a logically extended version of data-driven automation frameworks. Figure 2.4 illustrates the sample Excel sheet when using keyword approach for the example shown in Figure 2.3.

	A	B	C	D
1	Test Case	UI Element	Action/Keyword	Test Data
2	tc001	Button	Click	2
3		Button	Click	+
4		Button	Click	5
5		Button	Click	=
6			Verify	7
7	tc002	Button	Click	2
8		Button	Click	-
9		Button	Click	0.4
10		Button	Click	=
11			Verify	1.6

Figure 2.4 Sample table with keywords, UI elements and test data for a calculator program

2.2.5 Hybrid Approach

The hybrid approach is the combination of all the approaches mentioned above. Kelly (2003) mentions that the objective of building such a framework is to put together the benefits of each individual approach while leaving out the challenges in them. Figure 2.5 expresses the basic structure of hybrid approach highlighting how each approach is combined,

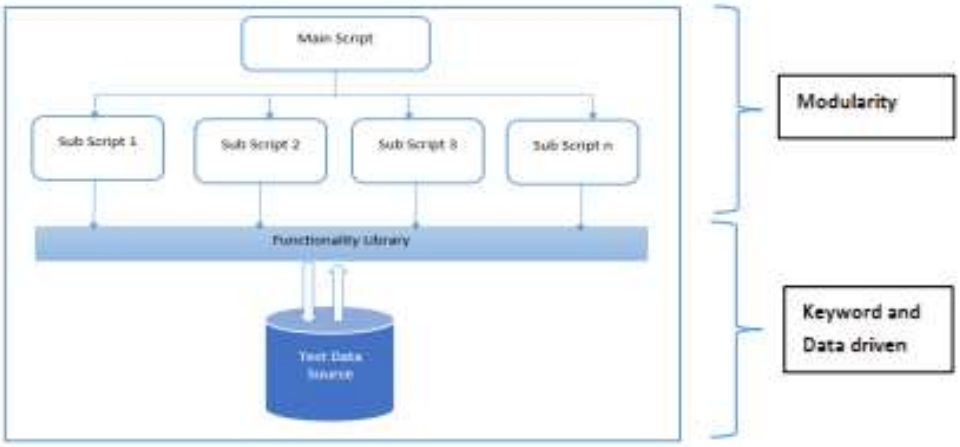


Figure 2.5 Architecture of the Hybrid Test Automation Framework

2.2.6 Comparison of Test Automation Framework Approaches

Following table (Table 2.1) provides a summary of benefits and challenges of each test automation framework approach discussed above, gathered from various literature sources.

Sources	Approach	Benefits	Challenges

(Zambelich, 1998);(Kelly, 2003); (Laukkanen, 2006)	Modularity -driven	• Scripting can start parallel to the development phase because for functionality changes only the relevant scripts need to be updated • Script maintenance is easier because of modularity • High scalability as individual scripts can be combined to test a complex system	• Requires technical proficiency in the scripting languages • Extra effort needed to create separate test scripts for varying test data sets since modular frameworks have test data hardcoded into test scripts
(Fewster and Graham, 1999); (Nagle, 2000); (Pettichord, 2003); (Kelly, 2003); (Laukkanen, 2006); (Javvaji et al., 2011);	Data-driven	• A single test script can be utilized for varying sets of data since test data is retrieved from external sources • Reduction of overall test script count due to above benefit and thereby increasing maintainability • Test data can be prepared in the design phase itself •Editing scripts is easier since all the variables and test data reside externally, the script acts only as a driver	• Need to implement different driver scripts to process different sets of test data •Initial setup requires technical skills
(Nagle, 2000); (Pettichord, 2003) (Kelly, 2003)	Keyword driven	• No technical skills required to use keywords • Maintenance cost is less	• Test scripts get lengthier and more complex than when

(Laukkanen, 2006); (Pradhan, 2011);		• A single keyword can be used across multiple test scripts, so the code is reusable. • Independent of automation tools and applications under test	using data-driven approach because functionality libraries should be defined
(Nagle, 2000); (Kelly, 2003);	Hybrid	• Combination of all approaches provide the combined benefits and mitigated challenges in each of the individual approaches • Less error prone	•Hybrid frameworks tend to be more complex due to the combination of modular, data - driven and keyword-driven architectures

Table 2.1 Benefits and Challenges of Test Automation Approaches

2.2.7 Evaluation of approaches

Kelly (2003) and Laukkanen (2006) theorizes that modularity is vital for a test automation framework for improved maintainability and scalability. Proper organization of test cases in a hierarchical manner would indeed contribute to a more organized and maintainable test suite. However, a modularity-driven approach alone would not address the problem discussed in this research. Because the model would not have separate test scripts and page objects classes (methods) for modules as the model is focusing the script less automation. Furthermore, Laukkanen (2006) suggests that data-driven scripts support easy maintenance of a framework. Also, it allows the flexibility and variance of test data which is important in functional testing. Observingly, integration of modularity-driven and data-driven approaches together would result in a much better test automation framework with increased benefits. Yet clearly, the research problem would remain unexplored in the resulting framework.

Nagle (2000) studies that the keyword-driven approach is best suited for testers with low programming skills. He elaborates that this approach can be adopted irrespective of the automation tool used in executing the test scripts. In contrast however, programming skills are necessary to define function libraries for the keywords used in test case development. Also

comparing Figure 2.3 and Figure 2.4 it can be noted that the same test case requires more test steps in keyword-driven approach, thus additional effort in preparing test cases in required. Laukkanen (2006) however agrees with Nagle and points out that although the initial effort and cost is high in keyword-driven approach, test case modifications are relatively easier and this increases the flexibility and reusability of test scripts thus proving this approach to be quite beneficial. Therefore, a hybrid framework which integrates all three approaches as depicted in Figure 2.5 is concluded as the solution to the problem pertaining to the research. A framework that is modular, data driven and having keyword driven capabilities would have a complex architecture. But it would fulfill the initial requirement of TA ideally. The organized structure due to modularity and increased maintainability will add to requiring low programming skills giving the test engineers more time to focus on actually testing the application rather than designing and developing test cases.

2.3 Test Automation Techniques

2.3.1 Scripting Techniques

Scripting is the process of test script creation, that is, the conversion of a manual test case into a test script in order to automate a particular scenario. Figure 2.6 provides a sample test case to validate the Google login page with valid username and password.

Project Name:

Sample Test Case

Test Case ID: TC_10	**Test Designed by:** <Name>
Test Priority (Low/Medium/High): Med	**Test Designed date:** <Date>
Module Name: Google login screen	**Test Executed by:** <Name>
Test Title: Verify login with valid username and password	**Test Execution date:** <Date>
Description: Test the Google login page	

Pre-conditions: User has valid username and password

Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to login page	User= example@gmail.com	User should be able to login	User is navigated to	Pass	
2	Provide valid username	Password: 1234		dashboard with successful		
3	Provide valid password			login		
4	Click on Login button					

Post-conditions:
User is validated with database and successfully login to account. The account session details are logged in database.

Figure 2.6 Sample Test Case (Software Testing Help, 2012)

There are several techniques used in the industry today by different test automation tools for scripting purposes. The two common techniques are,

1. Capture and Replay method
2. Manual Programming

2.3.1.1 Capture and Replay method

The tools which utilize capture and replay method are used to record every action made by a user automating a test case on screen. Every mouse movement, click and keystroke are captured. These events are stored as a script for playback later. Selenium IDE (Seleniumhq.org, 2016) is a conventional tool which uses this technique in order to develop test cases. Figure 2.7 illustrates the developed script for the sample test case in Figure 2.6 using Selenium IDE.

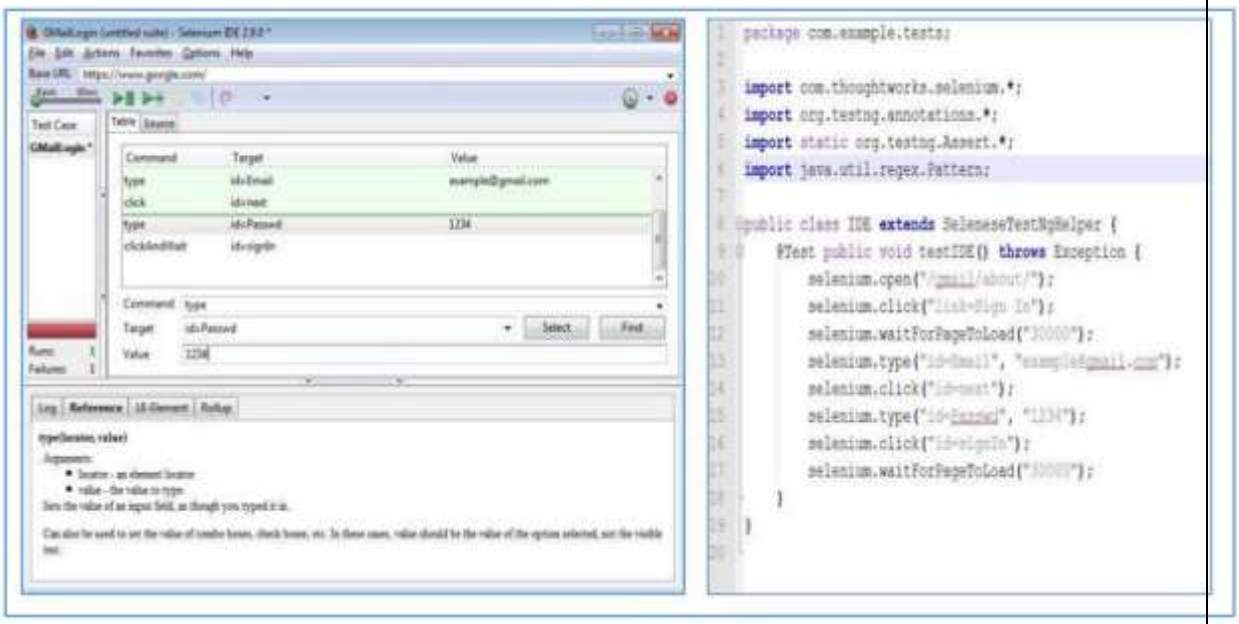


Figure 2.7 Selenium IDE Firefox plugin (Left) and the developed test case (Right)

The advantages and disadvantages of capture and replay method are discussed in Table 2.2 below.

Advantages	Disadvantages
• Easy to use	• Mistakes during recording require re-capturing the whole scenario
• No programming skills required	• Not flexible since it is not data-driven, one script applies to only one test case

• Facilitates fast script development	• Script maintenance is difficult because of low modularity
	• Scripts are strongly coupled with application interface

Table 2.2 Advantages and Disadvantages of Capture and Replay method

2.3.1.2 Manual Programming

Manual programming method is the development of test scripts by utilizing conventional programming means. This is the most widely used technique by automation tools. Laukkanen (2006) studies several different language categories used in script development.

1. System Programming Languages-Ex: Java, C++
2. Scripting Languages-Ex: Visual Basic, Python, Ruby
3. Shell Scripts-Ex: Bash
4. Vendor scripts-Ex: JRuby – A Java implementation of Ruby (Jruby.org, 2016)

Selenium Web Driver (Seleniumhq.org, 2016) is a web automation tool that uses manual programming methodology. It has several different implementations using several programming languages. Figure 2.8 presents the test script for the test case in Figure 2.6 developed using the Java implementation of Selenium Web Driver.

```
package login;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Login1 {
public static void main(String[] args) {
// Create a new instance of the Firefox driver
WebDriver driver = new FirefoxDriver();
//  Wait For Page To Load
// Put a Implicit wait, this means that any search for elements on the page
//could take the time the implicit wait is set for before throwing exception
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
// Navigate to URL
driver.get("https://mail.google.com)
// Maximize the window.
driver.manage().window().maximize();
// Enter UserName
driver.findElement(By.id("Email")).sendKeys('example@gmail.com");
// Enter Password
driver.findElement(By.id("Passwd")).sendKeys("1234");
driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
// Click on 'Sign In' button
driver.findElement(By.id("signIn")).click();
//click on Compose Mail.
driver.findElement(By.xpath("//div[@='z0']/div")).click();
// Click on the image icon present in the top right navigational Bar
driver.findElement(By.xpath("//div[@='gb_1 gb_3a gb_nc gb_e']/div/a")).click();
//Click on 'Logout' Button
driver.findElement(By.xpath("//*[@id='gb_71']")).click();
//Close the browser.
driver.close();
}
}
```

Figure 2.8 Test Script for Selenium Web Driver

The advantages and disadvantages of manual programming technique of TA are discussed in Table 2.3 below.

Advantages	Disadvantages
• High maintainability	• Requires programming skills
• Flexible scripts since test data can be parameterized for a data driven approach	• Scripting is more time consuming and requires extra effort
• Can introduce logical structures such as conditions and loops into test scripts	

Table 2.3 Advantages and Disadvantages of Manual Programming

2.3.2 Evaluation of Techniques

Capture and replay method and manual programming technique when compared are both individual mechanisms that deliver test scripts for the purpose of TA. Capture and replay method is clearly the easier technique out of the two when considering usability and adoptability. Also, it directly provides a solution to the problem addressed in this research, lack of technical expertise for TA. Laukkanen (2006) points out that capture and replay also appeals to the web domain with strict delivery timelines because scripting is much faster than programming methods. However, capture and replay method also has its challenges as expressed in Table 2.2

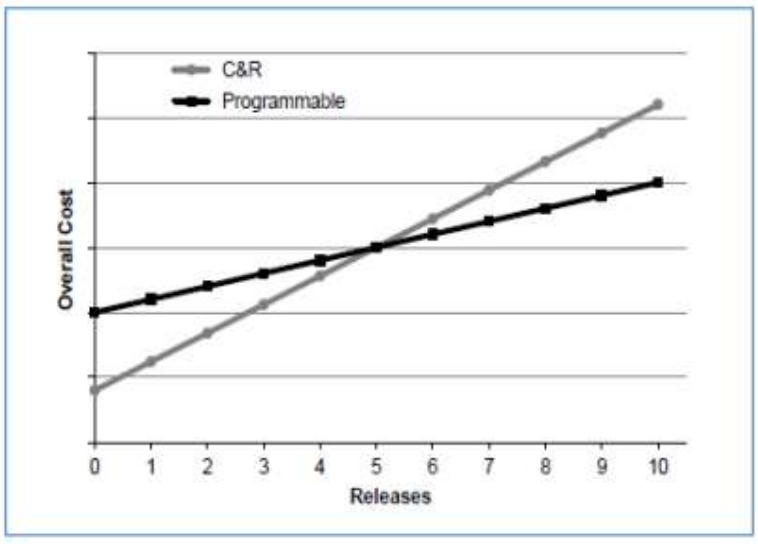


Figure 2.9 Evolution of Cost with Time of Capture and Replay and Manual Programming techniques (Leotta et al.,2013)

When considering cost and time efficiency of these two methods, (Leotta et al., 2013) presents an interesting observation as indicated in Figure 2.9. This graph was deduced through an empirical assessment carried out by Leotta in 2013 as discussed in the paper *Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution*. According to the graph, the initial cost of manual programming is higher than that of capture and replay. This is agreeable since scripting efforts, acquiring skilled programmers and initial setting up of manual programming environments would cost a considerable amount of money. However, Leotta et al. notes that over time, manual programming is more beneficial due to flexibility and high maintainability. Also, the reusability of test scripts saves a lot of time in comparison to capture and replay where each test case needs to be recorded again and again for varying test data which is very time consuming and tedious. Another benefit of manual programming is the applicability of logical structures in the code. Leotta et al. suggests that this is useful in testing complex web applications.

Therefore, arguably manual programming is more efficient than capture and replay in long term. Despite this, manual programming does not apply to the problem explored through this research since programming skills are required for TA when using this technique.

2.4 Natural Language Processing

A test automation framework which can process natural language texts describing the test case input by the user, extract test steps from that data and generate test scripts corresponding to the extracted test steps has been suggested as a solution for the research problem. Therefore, in order to process texts input to the application and extract events, a natural language processing component is essential.

Natural Language Processing (NLP) is defined by (Hirschberg and Manning, 2015) as means of utilizing computational techniques in order to understand, learn and produce content in natural language. Its' goal is to bridge the gap between human and computer interfaces.

2.4.1 Approaches in NLP

The two major approaches in NLP based on text analysis methods employed have been discussed below.

2.4.1.1 Classical NLP Approach

The classical NLP approach is also known as rule-based NLP. In Classical NLP approach, a machine executes a set of rules set by a human being or a language expert, in order to determine meaning of texts (Bhattacharyya, 2012). The rules are guided by linguistics, lexicography and

knowledge of language. Experts anticipate all possible grammatical occurrences in a language and set rules based on them.

For example, if NP = noun phrase, N = noun, ADJ = adjective, PP= prepositional phrase and P= preposition following rules can be defined by using the given test scenario below,

Test Scenario

Below test scenario is the sample test case to define the NLP rules.

Navigate to Login Page

Enter valid Username as XX

Enter valid Password as XX

Click on Login Button

a. NP => N (A noun phrase can be a noun)

Ex: Button

b. NP => ADJ + N (A noun phrase can be an adjective and a noun)

Ex: Login Button

c. NP => N + PP (A noun phrase can be a noun and a prepositional phrase)

Ex: Username as XX

d. PP => P + NP (A prepositional phrase can be a preposition and noun phrase)

Ex: as XX

2.4.1.2 Statistical NLP Approach

The statistical approach is a data-driven machine learning approach for NLP. It employs corpora to perform linguistic analysis on natural language. A corpus is a collection of textual data (Language.worldofcomputing.net, 2016) used as a source by the statistical approach. They provide extensive descriptions of the language. Some examples of English corpora available that are used in NLP are,

- Google Books Ngram Corpus
- American National Corpus
- British National Corpus

- Brown Corpus
- International Corpus of English
- Oxford English Corpus

An important concept in relation to statistical NLP is Annotations. Annotating is the labeling of texts to increase meaning and value (Bhattacharyya, 2012). The labeling of each word with information such as noun, verb, city, name and many other forms increases the meaning of the sentence. Annotation relates to similar as well as more complex levels of labeling text in order to enrich its value.

	Advantages	Disadvantages
Classical NLP	• Based on linguistic theories	• Requires rules to be pre-defined
	• Suitable for languages with limited theories	• Exceptions in human language cannot be handled
	• Computational resources are not essential	• Costly to maintain
	• Easier error analysis	• Difficult to extend
		• Disambiguation issues
Statistical NLP	• Linguistic knowledge not required	• Computational resources essential
	• Maintenance is easier	• Difficult error analysis
	• Reduce cost of human resources	• Requires large collections of textual data

Table 2.4 Comparison of advantages and disadvantages of classical and statistical NLP ((Farrús et al., 2012))

Table 2.4 presents a comparison of the two approaches of NLP (Farrús et al., 2012). Both approaches have their own benefits as well as challenges. Classical NLP appeals to a test automation framework because test cases follow a certain language pattern on which rules can be defined.

The keyword-driven approach for automation frameworks uses a similar technique as Classical NLP. It identifies that each test step consists of Action, UI element and Test Data. Accordingly, rules such as (a) A test set can be an action and a target, (b) a target can be a UI element and Test Data, can be set.

In contrast, annotations used in statistical NLP are a simpler way of defining the structure of a test step and maintenance and extension of statistical approach is easier. Therefore, statistical approach is considered best suited for an NLP based test automation framework. However classical approach will be considered when necessary in defining rules for higher accuracy.

2.4. 2 Natural Language Processing Technologies

There are many NLP libraries available with varying functionalities depending on the tasks performed by them. Two of the most common general NLP libraries are,
1. Stanford CoreNLP - A collection of NLP tools based on Java that are able to perform different tasks, developed at Stanford (Stanfordnlp.github.io, 2016)
2. Apache OpenNLP - A Java based library that uses machine learning techniques to perform common NLP tasks (Opennlp.apache.org, 2016)
Table 2.7 presents the comparison of Stanford CoreNLP and Apache OpenNLP libraries in relation to different criteria (Karlin, 2012) that are used to determine the best tool for NLP integration in the proposed solution.

Tools/Criteria	OpenNLP	CoreNLP
Programming Language	Java	Java
Approach	Statistical	Statistical
Performance	Higher performance less time and resource consumption	Lower performance relative to OpenNLP
Programming Effort	High	Low
Documentation Support	Not as much as CoreNLP	Available abundantly
NLP Tasks		
Sentence Disambiguation	Yes	Yes
Word Tokenization	Yes	Yes
P-O-S tagging	Yes	Yes

Table 2.5 Comparison of CoreNLP and OpenNLP

Upon comparison of the two NLP libraries, it is noted that both are capable of performing the required NLP tasks for the system and that the implementations of both systems are based on

Java. However, despite the fact that Karlin (2012) points out that OpenNLP has higher performance relative to CoreNLP, he also notes that the lines of code required to implement the same functionality in OpenNLP is higher than that of CoreNLP, thus the programming effort required by OpenNLP is higher. Furthermore, documentation for CoreNLP is readily available therefore it is easier to adopt and implement. In contrast however, documentation regarding training data and models in available mostly for OpenNLP. Therefore, the author confirms that both libraries are best suited for separate tasks and therefore both would be considered during implementation.

2.4.3 Natural Language Processing Tasks

There are various NLP tasks are used for analyzing and understanding natural language texts. The tasks pertaining to the suggested solution are discussed below.

2.4.3.1 Sentence Boundary Disambiguation

Sentence Boundary Disambiguation (SBD) is the task of separating natural language texts into separate sentences. In order to determine the end of a sentence a binary classifier technique is used with two Boolean outputs true and false stating if it is the end of the sentence or not. These binary classifiers can be implemented using rules, regular expressions or machine learning. The simplest classifier used for this is a Decision Tree. Figure 2.10 illustrates a simple decision tree to determine end of sentence (EOS).

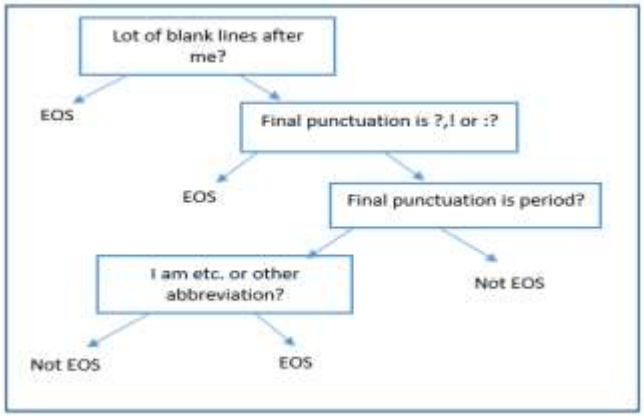


Figure 2.10 Decision Tree to determine EOS

Decision trees are often implemented using a statistical approach where the structure is learned through machine learning from a training corpus.

2.4.3.2 Word Tokenization

Tokenization is the isolation of a sentence into parts known as tokens. A token is a collection of characters that are grouped in order to form a meaningful unit (Manning et al., 2008). A type is an element from the vocabulary. A type is distinct whereas tokens are stated as the occurrences of types in a text or corpus.

2.4.3.3 Part of Speech Tagging

Part of Speech (POS) tagging is a task where each word, or rather token, is assigned a tag which indicates its part of speech in the sentence or semantic information (Language.worldofcomputing.net, 2016). The same word can be a noun, verb or adjective given the context in which it is used. POS tagging is used to distinguish this information. This task is carried out by programs known as POS taggers. Figure 2.11 depicts the classification of POS tagger models based on approach.

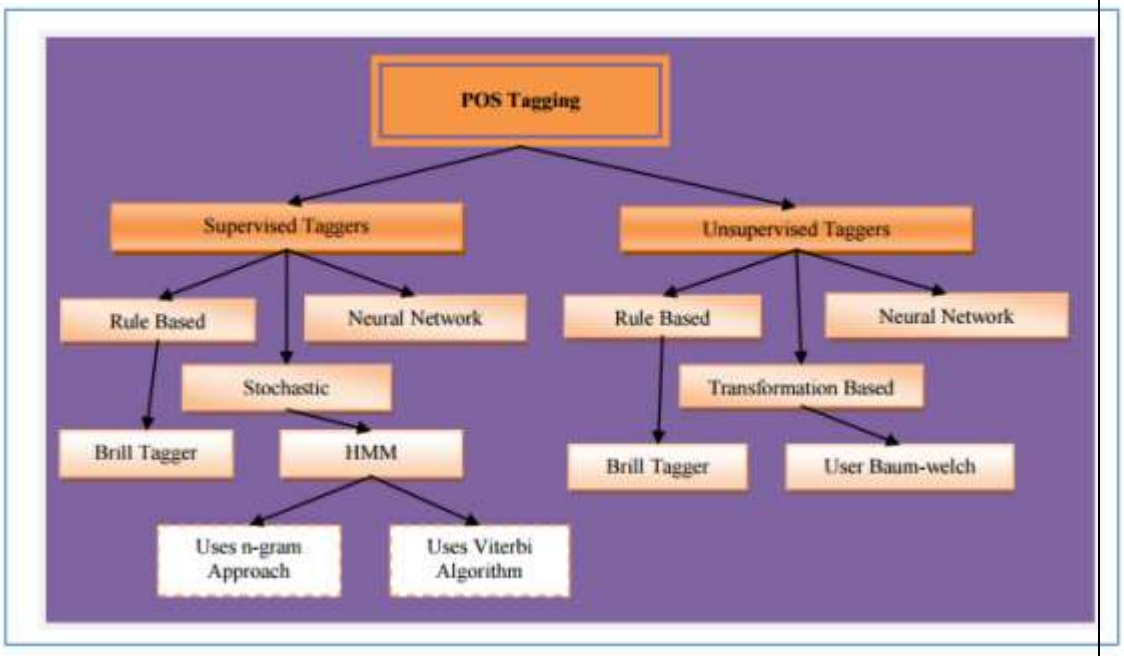


Figure 2.11 Classification of POS tagging models (Kumawat,2015)

2.4.4 Evaluation

Upon reviewing the two main approaches in NLP, it was clear that the statistical approach has shown to be more beneficial in addressing the research problem as the linguistic knowledge is not required. However, if the need arises for specific rule definitions, classical NLP approach will be considered.

Several tasks of NLP will contribute to extracting actions from text input by users in natural language. Sentence segmentation will be used in separating individual test steps and word

tokenization and POS tagging will be applied to separate each word as a token and determine the target, UI element and test data from each test step. The stochastic approach of POS tagging is favored over rule-based approach because defining rules is time consuming and the time constraint on this research should be considered. However, the author observes that the available text corpora will not be sufficient in processing test steps through POS tagging therefore a text corpus with more suited tags for the TA domain should be trained to a tagger. The technologies most suitable for this will be discussed in the next section.

2.5 Technologies

2.5.1 Web Automation Frameworks and Tools

When considering web automation there are several tools and frameworks available in the software market today. Table 2.6 presents a comparison between three popular web automation technologies ((Kaur and Gupta, 2013); (Gupta et al., 2015); (Monier and El-mahdy, 2015)). Table 2.5 presents the parameters using which the tools will be compared.

Parameter	Explanation
Cost	Initial cost of tool
Language	Scripting languages supported by the tool
Platform Support	Operating systems supported by the tool
Browser Support	Browsers that can be automated using the tool
Data-Driven Framework	Data sources supported by tool as a data-driven framework
Technical Expertise	Programming skills required for scripting or not
Ease of Use	Previous experience required in utilizing the tool

Table 2.5 Explanation of comparison parameters

Tools/Parameter	Selenium	Quick Test Professional (QTP)	TestComplete
Cost	Open Source	Commercial Tool	Commercial Tool
Language	Ruby, Java, Python, Php, JavaScript	VBScript	VBScript, C#, JavaScript
Platform Support	Windows, Mac	Windows Only	Windows 7 and Higher

Browser Support	Google Chrome, Mozilla Firefox, IE, Opera	Google Chrome, Mozilla Firefox, IE	Google Chrome, Mozilla Firefox, IE, Opera
Data-Driven Framework	Excel, CSV	Excel, Text files, XML, DB files	Excel, CSV, SQL
Technical Expertise	Required	Partially	Required
Ease of Use	Experience Required	Required	Experience Required

Table 2.6 Comparison of Selenium, QTP and TestComplete

Considering the three web automation tools compared in Table 2.6 above, Selenium, QTP and TestComplete, Selenium is the only open source framework available. Therefore, acquiring Selenium for development purposes would be easier. Also compared with the other two tools, Selenium additionally supports Mac OS as a platform. All three tools support the major browsers however QTP is not supported in Opera browser. None of the tools support natural language scripting. When ease of use is considered QTP seems a better option than the other two. However, since Selenium is more beneficial in every other way and also considering the fact that Selenium is an easily extendable framework because it is open source, Selenium is ideal as the foundation automation framework for web applications in the suggested solution for execution of generated test scripts.

2.6 Related Work

Studies related to using natural language for programming have been carried out by many researchers in the domain. Thummalapenta et.al (2011) presents a conceptual system that can analyze stylized English test steps to generate test scripts. They have used the Stanford NLP POS tagger to identify separate tuples containing action, UI element and test data to generate code. Adding to that research Madhavan (2014) presents a similar approach however the POS tagging in his system is done through a custom trained model and therefore does not require stylized English unlike the previous system by Thummalapenta. Madhavan's system is capable of processing natural language test steps and generate test scripts. A constraint noted in his system however is the object mapping module. A user should input an additional set of data mapping each object with an action. This is extra effort in developing scripts and can be time consuming.

More recently, Testsigma Inc presented Testsigma which is a SaaS, AI-Driven test automation software for Web and Mobile applications to achieve continuous testing with Shift-left

approach. Testsigma helps the web and mobile dependent businesses to reduce the cost of software quality and to continuously release their great quality software products faster.
Testsigma uses NLP to build stable and reliable tests faster and speed-up the execution and maintenance of automated tests. Testsigma is built to address some of the problems with existing automation testing tools, like huge initial time and cost, slow test development, high execution time and costs, high maintenance efforts, less automation coverage and longer payback time.

As in the customer review report 2021, customers have faced some issues while using the Testsigma tool such as their in-built custom commands breaks, sometimes application is slow and freezes and users cannot copy the test steps from one scenario and paste in another scenario. Testsigma is not provided any view to see the Generated Script from the test scenario and directly focus the test execution. However, TA tools should facilitate the QA engineers to understand the view and generated script before the test execution. This view will display the executable code that has been generated by the model which corresponds to the test steps that have been entered by the user.

Chapter 3 – **METHODOLOGY**

3.1 Design

3.1.1 Development Methodology

For the selection of a proper development methodology, existing methodologies like waterfall, prototyping, agile and rapid application development (RAD) have been taken into consideration. Although waterfall model is commonly used as a software development methodology it has the disadvantage of not being able to return to a previous phase in the Software Development Life Cycle (SDLC) once the development process surpasses that particular phase. In this research project it is important to have the capability of traversing back and forth the SDLC in order to integrate new features for the framework as required to optimize efficiency. Therefore, waterfall model is inconvenient for this project. Moreover, when using the waterfall model, a fully functioning prototype can be obtained only at the end of the SDLC. Rapid Application Methodology is not applicable since a strong design and development team is unavailable for this project. In agile, development takes place as iterations of short time frames typically a week or two known as sprints. At the end of each sprint a few fully functioning features are added to the prototype. Combined sprints produce the end result. Agile is capable of adapting to change which in this project can be feedback received regarding the product.

However, time allocated for this research is insufficient to schedule sprints for development. Therefore, agile is not considered as the best option for a development methodology for this project. In contrast however, prototyping is a suitable development methodology since several prototypes can be obtained during the SDLC which is convenient for this research since prototypes are necessary to obtain expert feedback on the product. The author views this methodology as the most suitable development methodology since it adds flexibility to traverse the SDLC as desired in order to optimize the product while delivering multiple functional prototypes for evaluation purposes. Figure 3.1 outlines the progression of prototype model.

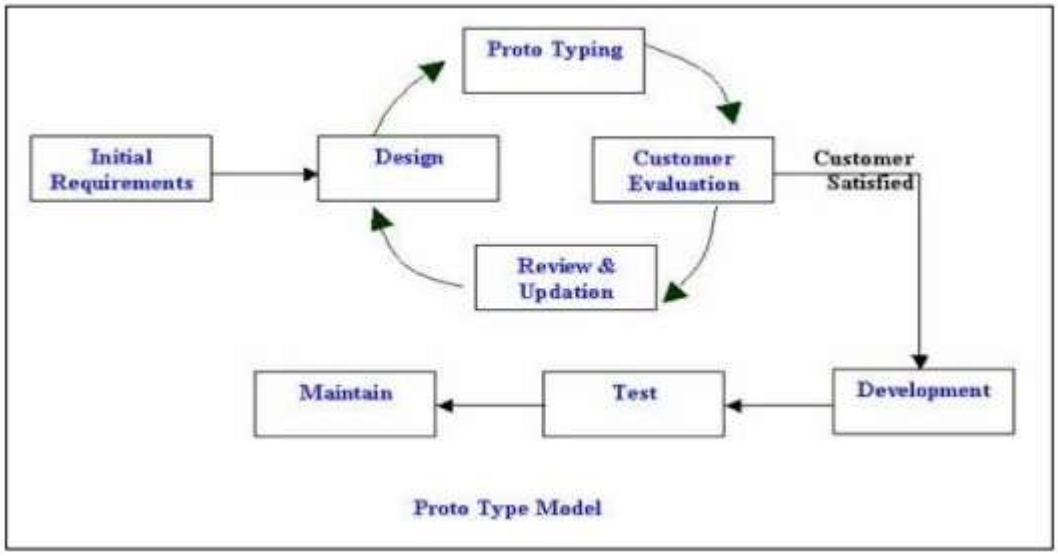


Figure 3.1 : Prototype Model (Learnwithkamal.wordpress.com, 2017)

3.2 High Level Design

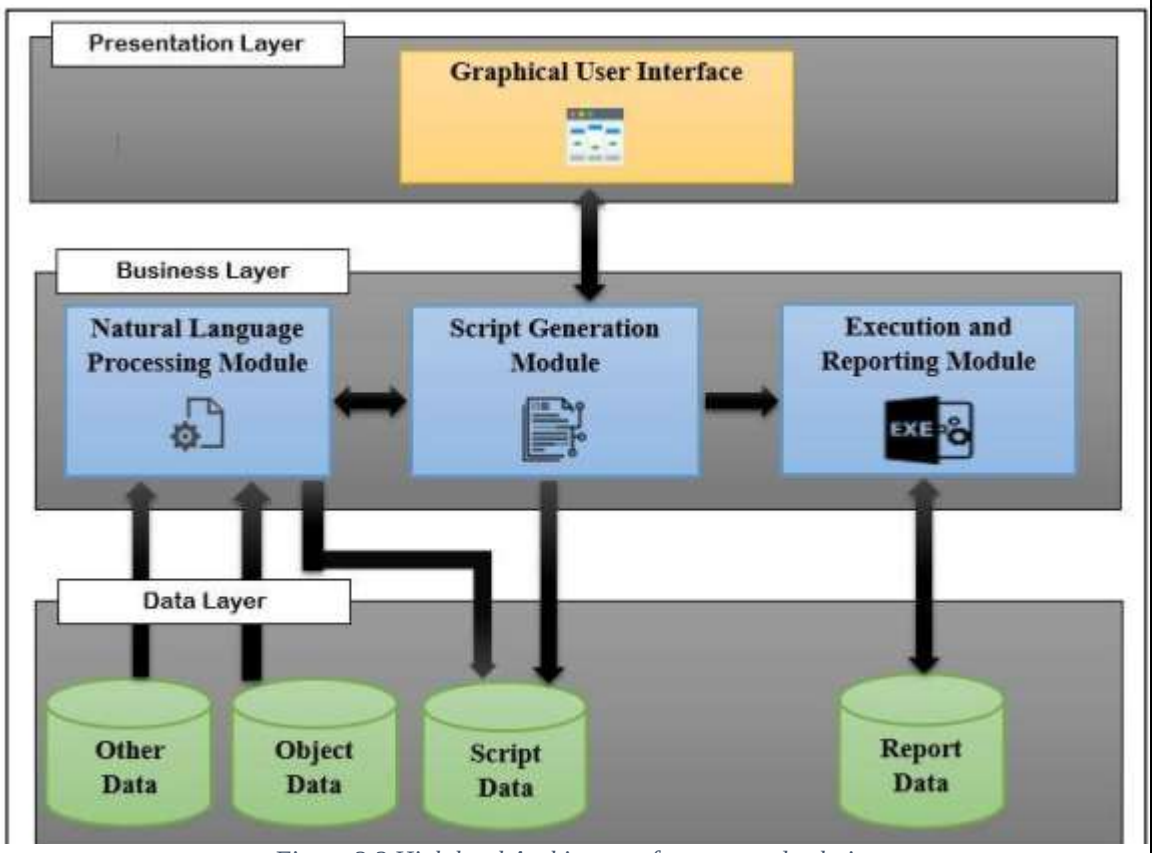


Figure 3.2 High level Architecture for proposed solution

Figure 3.2 illustrates the high-level design which is an overall overview of the proposed solution outlining the relation between individual modules, data structures and data flow. The solution has been designed based on one tier architecture.

One tier architecture consists of all its elements, namely the presentation layer, business layer and data layer installed on the same location. This architecture was chosen for the solution since it is easier to implement and therefore the cost of maintenance and deployment is low. Issues related to complexity and compatibility are also minimized through this approach. Such applications are known as fat or thick client applications.

1. Presentation Layer

The presentation layer comprises of the Graphical User Interface. It will be used by a QA Engineer to interact with the system. It should provide the user with means of creating, editing test scripts, generating and executing automation scripts and viewing execution result reports. The graphical user interface needs to conform to the functional requirements that follow, A user should be able to:

- generate a test script by entering test steps and saving them
- execute any generated automation script in the execution environment
- view and filter reports based on date and time
- upload UI object data which contains UI locators into a new or existing object repository (object repository is excel sheet which consists the UI locators to relevant objects)
- edit an existing test script
- delete an existing test script

The above requirements have been taken into consideration when designing the GUI of the solution.

Additionally, usability aspects namely how the system is easy to use for a user have been considered and incorporated as a non-functional requirement of the GUI.

The main UI of the system consists of three components,

a) Scenario Tab

This component provides the text editor for the user to enter test steps in English. There is list of predefined commands. The list consists of commands such as Open, Type and Click. Open

command is similar to “GoToURL” command in selenium. Usually in UI automation the respective url needs to be launched. And other main events in UI automation are clicking elements and entering the texts. Click event consists of buttons, checkboxes, radio buttons and links. Predefined ‘Click’ command will cover all of these and ‘Type’ command will cover the text entering.

Sample test scenario example

Open url https://www.linkedin.com/home
Type email as abc@email.com
Type password as 12345
Click button signIn

The “Upload Object Data” button will allow a user to upload an excel sheet which conforms to a specific format consisting of object data such as xpath locators and locator names. Object Data component in figure3.2 uploads the object data into Natural Language Processing module. The format of the excel sheet will be decided during implementation. In order to promote ease of use, the uploaded UI information will be displayed beside the editor for the user to refer to when entering test steps. The “Generate” button is placed for a user to convert the plain text input into executable automation steps.

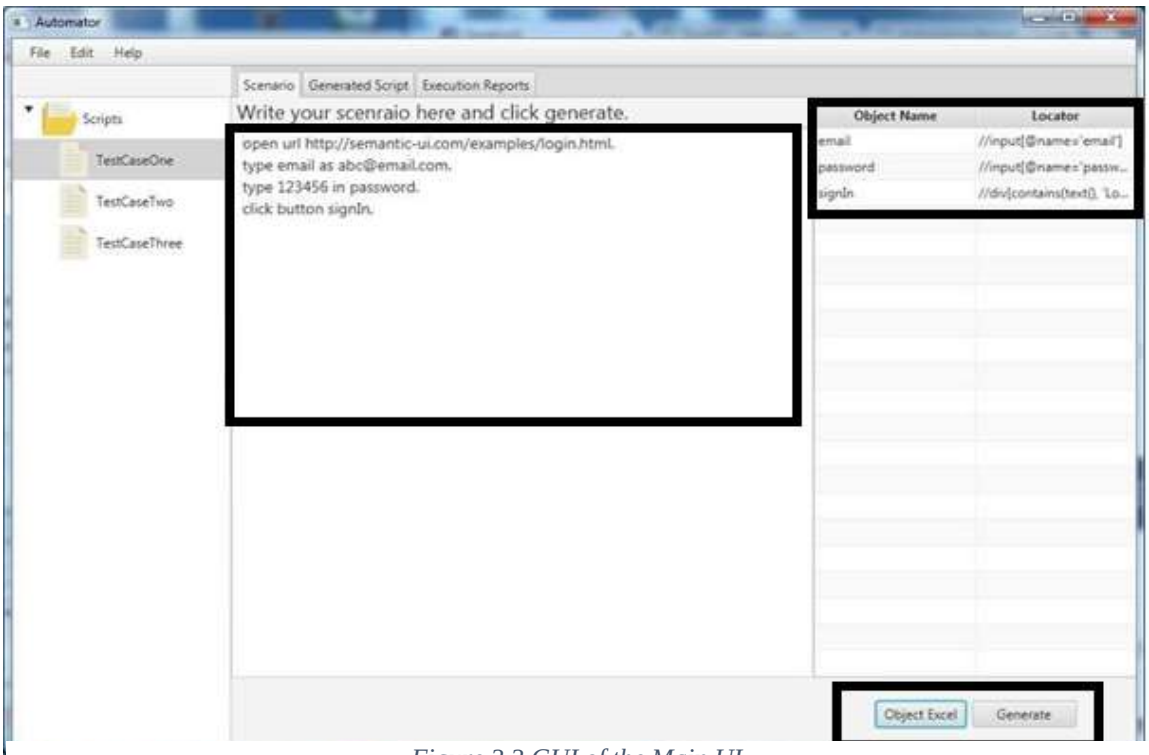


Figure 3.3 GUI of the Main UI

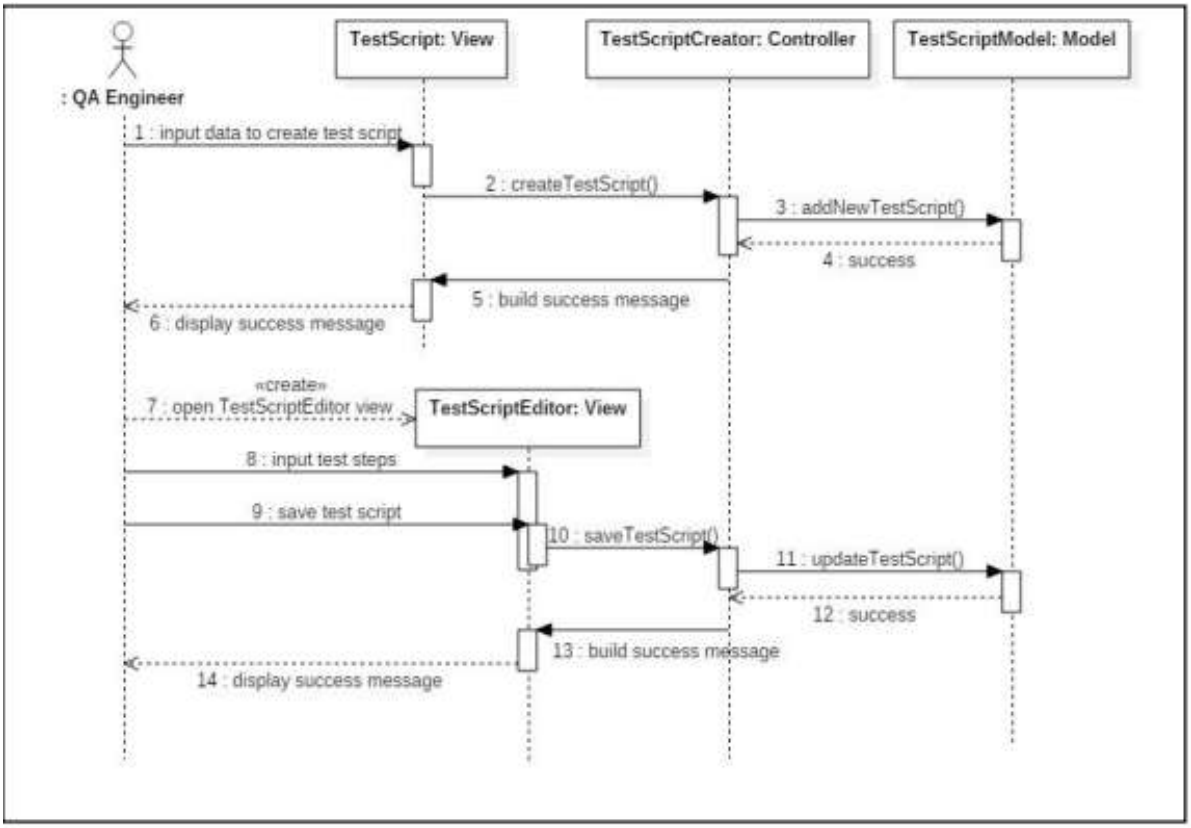


Figure 3.4 Sequence Diagram for Create/Update Test Script

The created test scripts and locator information will be saved in JSON format in a text file. Since the solution is a rich client application a database has not been integrated to avoid complexity.

b) Generated Script Tab

This tab will display the executable code that has been generated by the model which corresponds to the test steps that have been entered by the user. “Execute” button will be added to execute the code to generate a result report.

c) Execution Reports Tab

The execution reports tab will consist of a list of a list of reports corresponding to each generated script. The reports will be generated in HTML format, which can be viewed using any browser, so that more graphical content can be included and the report can be shared among the QA team.

2. Business Layer

The business layer consists of three separate modules,

- Natural Language Processing Module

This module will process the test steps entered by the QA Engineer in English and identify the actions, keywords and UI elements in each statement by isolating nouns and verbs using a natural language processing library.

Processing of text for this framework consists of two stages, separating sentences as individual test steps and then identifying the pre-specified keywords which are verbs would be considered as actions. Nouns which would be considered as UI elements and test data. Figure 3.5 presents an overview of the text processing algorithm in the NLP module.

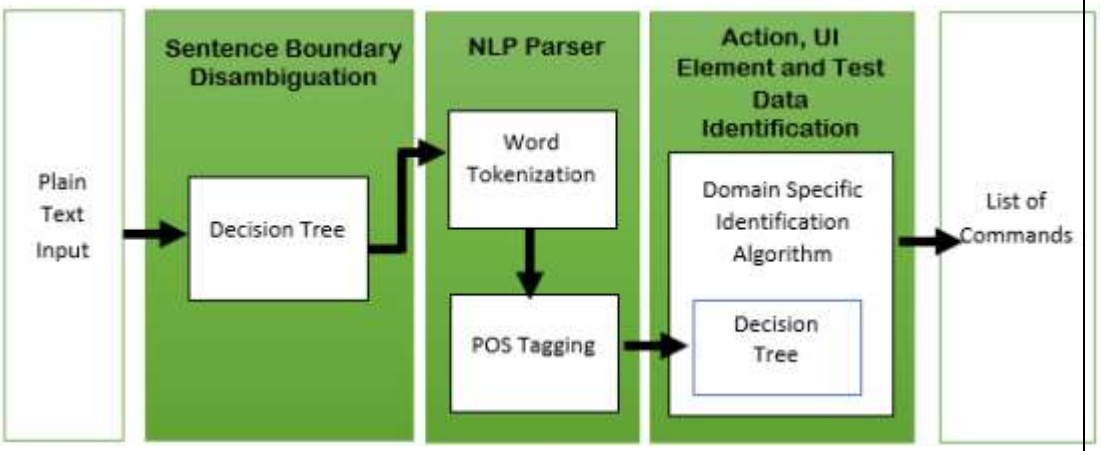


Figure 3.5 Text Processing Algorithm in NLP Module

The text input is primarily divided into separate sentences using sentence boundary disambiguation technique in natural language processing. The decision tree involved in separation of sentences has been previously discussed under section 2.4.2.1. The separated sentences are then separated into words using word tokenization technique. Subsequently, part of speech (POS) tags will be assigned to the tokens. The verbs in each sentence will be extracted to determine the action and the corresponding automation command for the action. Once the automation command has been decided, disambiguating UI elements and test data from the nouns in the sentence is achieved using a domain specific implementation of a decision tree algorithm.

The design of the Statement entity (user input) has been based on the template design pattern as shown in Figure 3.6. The build method which will be overridden in each subclass that implements the Statement super class will consist of the statement specific identification algorithm to extract data from the tokenized text.

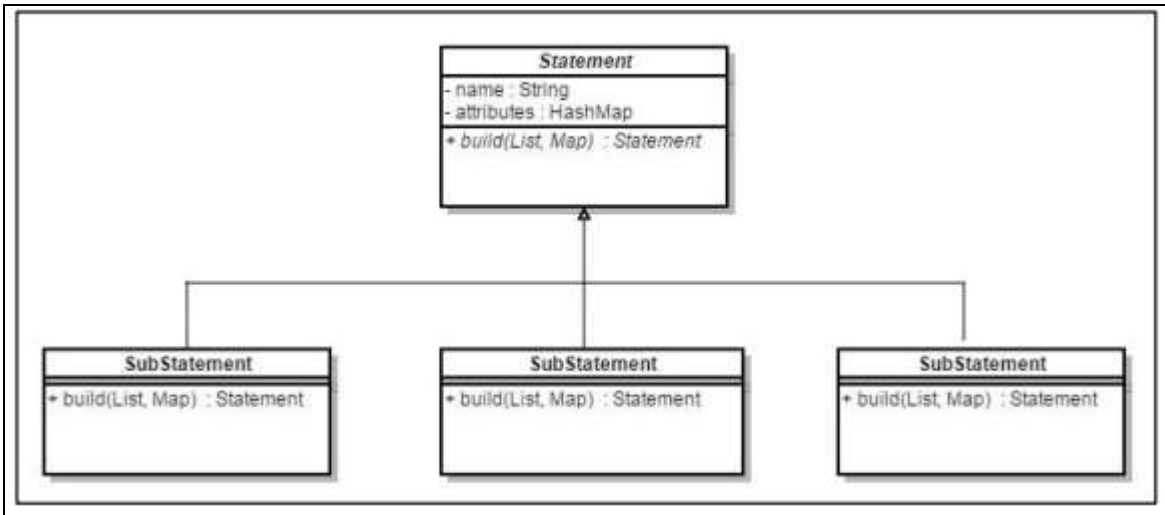


Figure 3.6 Template Design Pattern for Statement class

The final output of the NLP module is a list of statement objects containing the extracted data.

- Script Generation Module

Script Generation module consists of the logic to generate test commands based on the keywords and actions identified by the NLP module to produce an executable automation script. Test commands are Selenium wrapped and can be executed using a Selenium framework.

The list of statements output by the NLP module is taken in as input by the script generation module.

This list can consist of statements such as Open, Type and Click. These statements are then converted into executable commands by mapping them with a predefined template. The UI object data uploaded by the user is separately mapped to another template and generated as an object page. The format of these templates will be decided during implementation. The generated script is then integrated into an automation framework which acts as the runtime for generated automation scripts. Sequence diagram corresponding to generate automation script is shown in Figure 3.7;

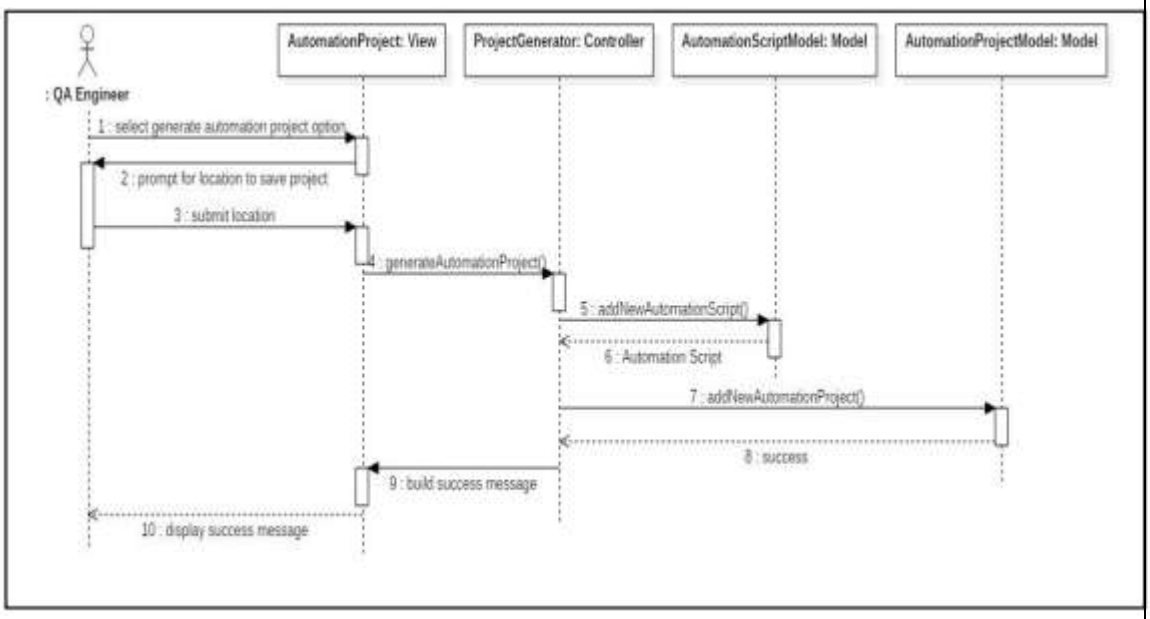


Figure 3.7 Sequence Diagram for Generate Automation Project

- Execution and Reporting Module

This module handles automation script execution using Selenium test automation framework for web applications. A result report will also be generated for each execution by this module. Selenium, as discussed in 2.4.1 is a web automation model that can easily be extended. Therefore, the execution and reporting functionalities will be handled by an automation framework that will be built on top of Selenium. This automation model will act as the core of the proposed solution. The following features have been taken into consideration when designing the model,

- Execution of automation scripts

During execution, the model has to extract data from generated automation commands and pass it to the Selenium framework by creating a Selenium command object. Dynamic creation of web elements and resolution of xpath locators should also be handled during execution.

- Error handling during execution

Errors during execution such as xpath locator mismatches, time out errors and element not found errors need to be handled.

- Report generation

Report should be generated during execution containing details of every test step executed and the corresponding pass or fail status.

The proposed architecture for the automation framework developed as the execution and reporting module is presented below in Figure 3.8

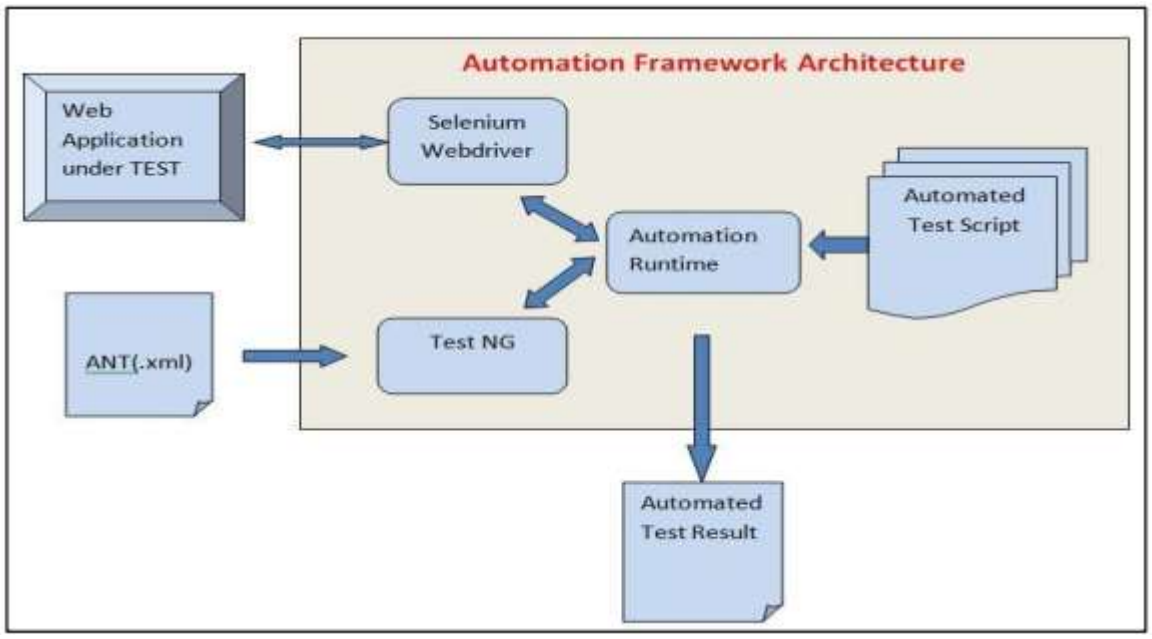


Figure 3.8 Proposed Automation Framework Architecture of Executing and Reporting Module

The technologies used in the automation framework will be justified during implementation. The sequence diagram pertaining to automation script execution is presented in Figure 3.9.

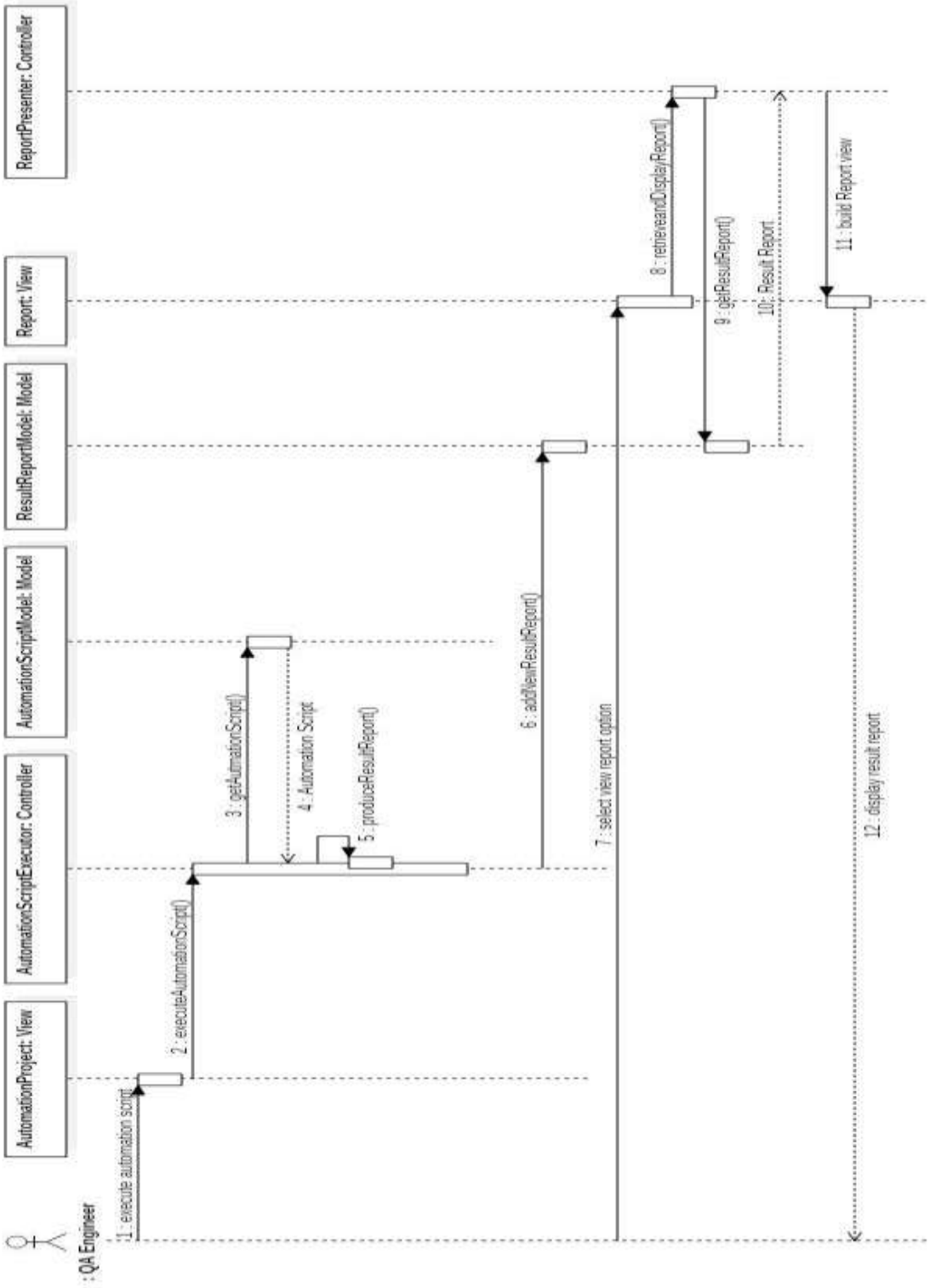


Figure 3.9 Sequence Diagram for Execute Automation Script

3. Data Layer

Data	Description
Script Data	Script data consists of test scripts created by users and automation scripts generated by the model.
Object Data	The UI element data such as xpath locators and locator names are saved as Object data.
Report Data	The execution time, name of script, results of each test step for each execution is saved as a report under Report data.
Other Data	Other data consists of static raw data required for the generation of automation scripts such as the text corpus used to process user input test steps.

Table 3.1 Data Sources in the Data layer

3.3 Component and Deployment Diagrams

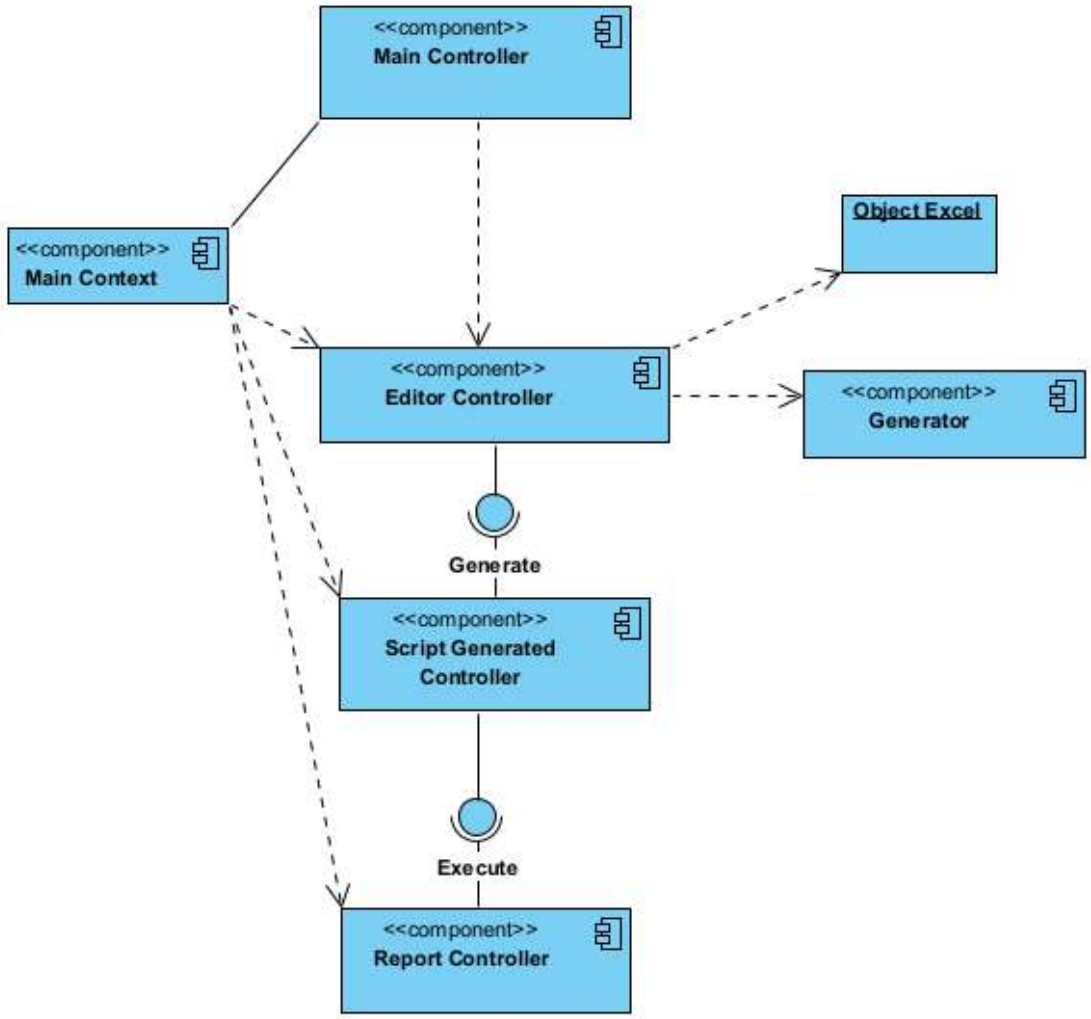


Figure 3.10 Component Diagram of proposed test automation model for web application

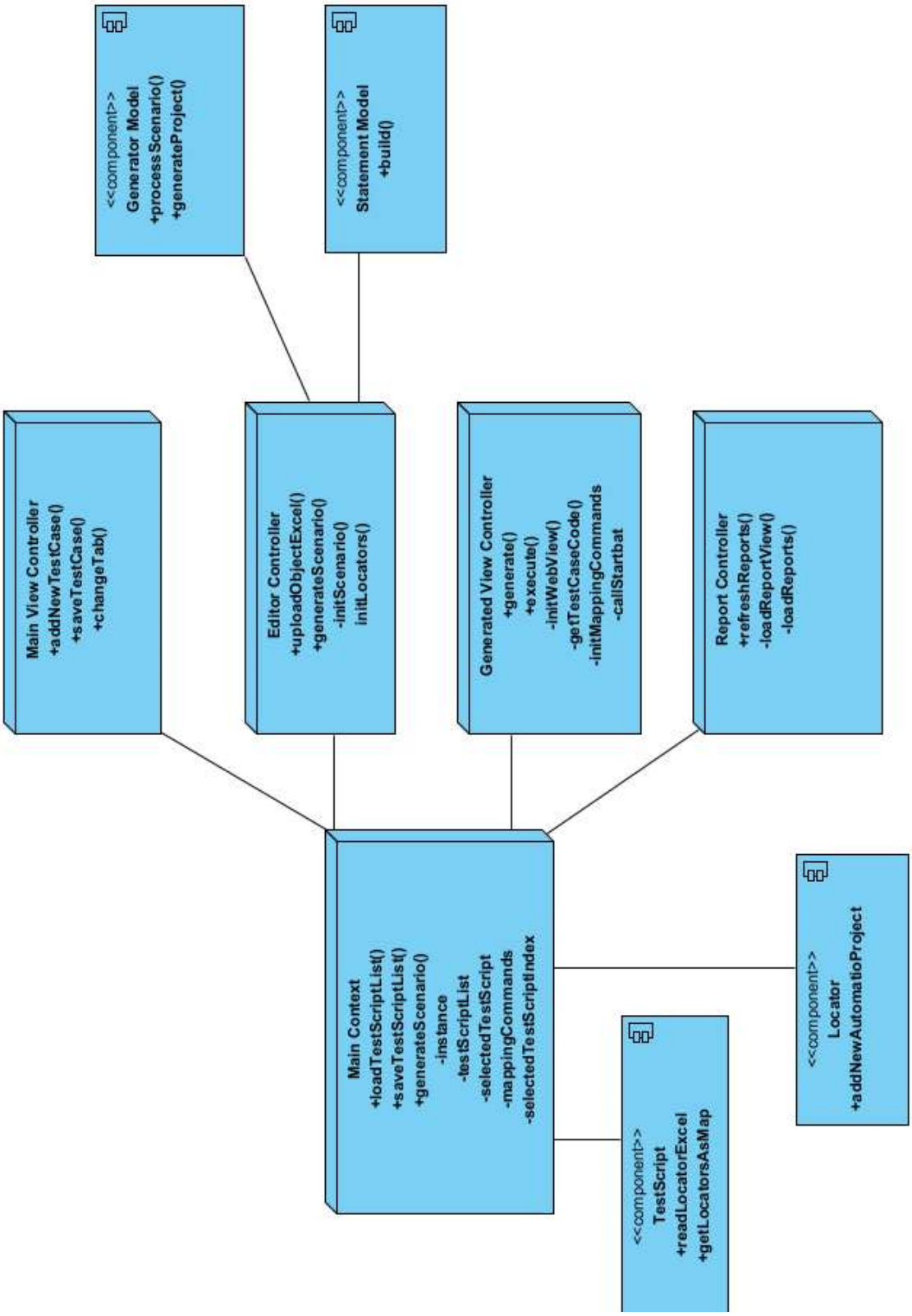


Figure 3.11 Deployment Diagram of proposed test automation model for web application

MVC architecture has been used in order to separate business logic from the views facilitating the re-use of logic throughout the application.

Since the framework is required to be modular and flexible as pointed out in the requirement specification, usage of MVC ensures inter-relation between components in the code base are minimized. Models, views and controllers are independent of each other and therefore new classes as well as new operations to the existing classes can be introduced as future enhancements with minimum impact on the existing system.

Table 3.2 provides details as to how each class contributes to the system as a whole.

Class Name	Description
Models	
Context	This class inserts and retrieves test scripts to and from the data source
TestScript	TestScript class represents the test script entity. It will retrieve locator information to create a TestScript object.
Locator	Locator represents the UI element locator entity. It consists of name and locator attributes.
Views	
Main	Main view contains the file menu, test case tree and three tabbed views. By default, Editor view is selected.
GeneratedView	Generated view consists of the generated automation script to be viewed by a user.
Reporter	Reports generated after execution will be displayed in the Reporter view.
Editor	The Editor view consists of a text area for the user to enter test steps. It also consists of a table which displays object locator data uploaded by the user.
Controllers	
Editor	Editor controller handles the building of object locator table using the Excel sheet uploaded by the user.

Generator	Generator controller will handle extraction of test data and locator information from the plain text scenario input by the user to generate automation script.
MainViewController	Main View Controller handles the loading if the test case tree and loading and saving of test cases.
Reporter	Reporter controller loads reports from the Context model and builds Reporter view.
GeneratedView	Generated View Controller handles the execution of generated automation scripts and reporting functions.

Table 3.2 Description of class diagram of proposed solution

3.4 Implementation

3.4.1 Development Environment

A discussion of the development environment within which the implementation of the solution was carried out follows,

3.4.1.1 Eclipse IDE

Programming of the solution was mainly done in Java. Existing Java development environments have been taken into consideration with the aim of determining a suitable Integrated Development Environment (IDE). The selected IDE should at a minimum support Java, JavaFX and Maven for the development purposes of the proposed solution. Eclipse, Netbeans and IntelliJ IDEA are the three most commonly used Java IDEs. All three IDEs provide an editor for coding and debugging and an IntelliSense feature for code completion. Code refactoring to obtain quality code is also possible using any of the three IDEs mentioned. In comparison with the other two, Eclipse has a large number of plugins readily available making it very versatile and customizable (Lifewire, 2017). Netbeans has better database support owing to the fact that it has multiple database drivers integrated within the IDE. However, since the solution has no database feature, that advantage of Netbeans is not applicable. Features of IntelliJ such as smart code completion and code refactoring are appealing to a developer. Nevertheless, Eclipse has been chosen as the IDE that would be used due to its extendibility via plugins and also since the developer is more conversant in Eclipse than the other two IDEs stated.

3.4.1.2 JavaFX Scene Builder

JavaFX has been chosen over Swing to implement the user interface of the application due to its many advantages. JavaFX has many improvements such as the ability to develop markup interfaces using FXML and beautify them by applying CSS. It also supports MVC pattern so that the views can be separated from models and controllers as indicated in the design of the proposed solution. Scene Builder by Oracle is a tool that allows quick and easy implementation of JavaFX application UIs with minimum amount of coding (Oracle.com, 2017).

3.4.2 Implementation of Graphical User Interface

The GUI of the application has been developed according to its proposed design. Requirements of the GUI were taken into consideration during the design phase and a mockup of the main UI was drafted. The implemented GUI is presented in Figure 3.11.

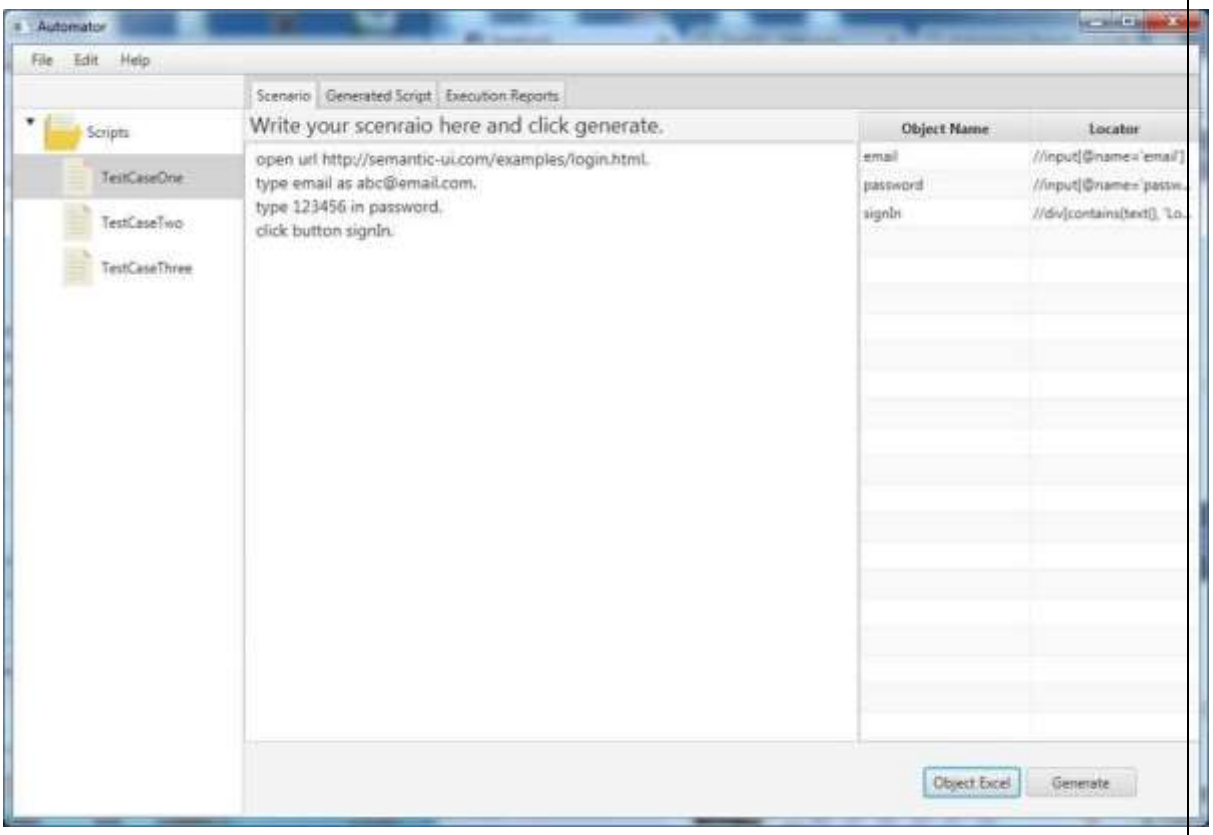


Figure 3.12 Main view of the application

The UI elements were placed for easier navigation and on-screen instructions were added to increase usability of application. The "Scenario" tab is open by default. A text area has been provided for the user for scripting purposes. The "Generated Script" tab view is presented in Figure 3.12

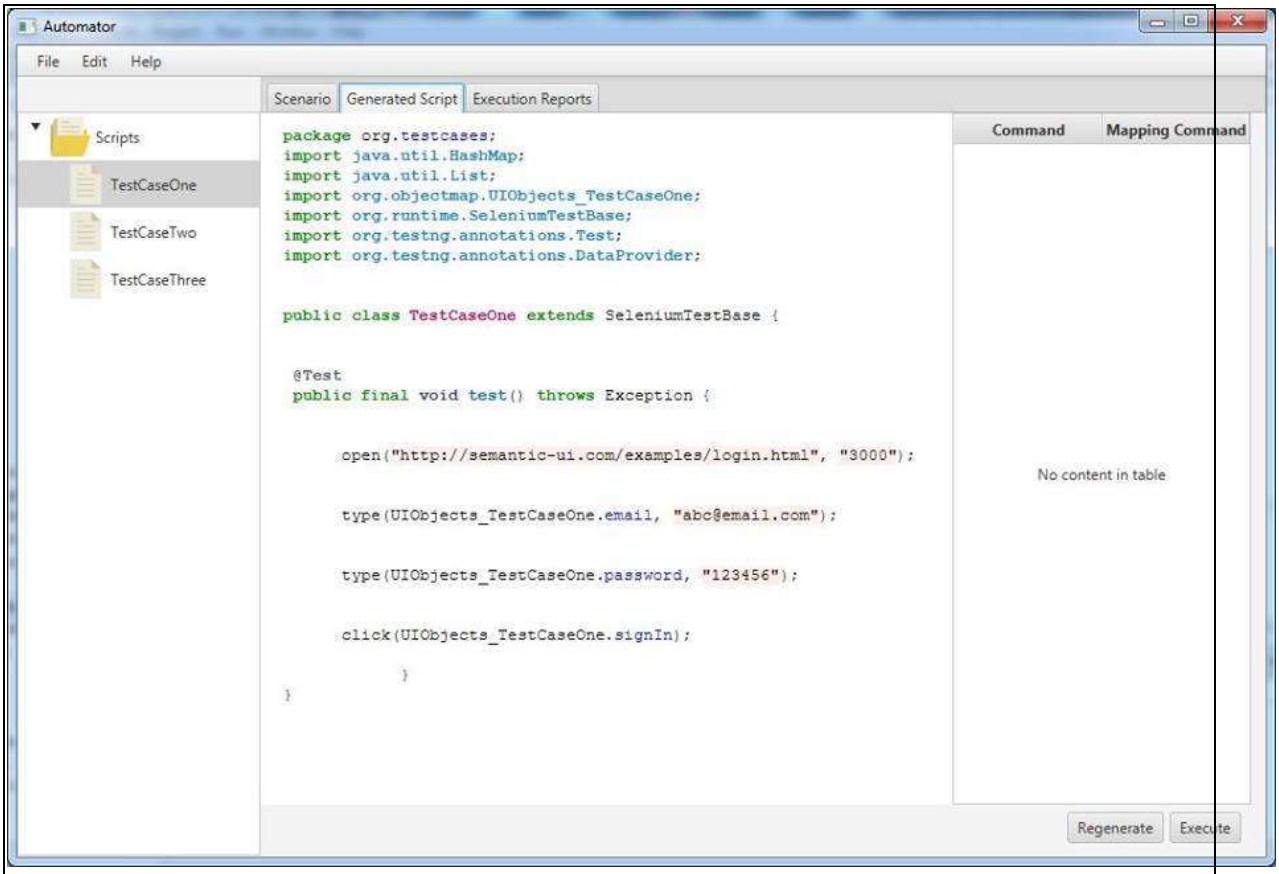


Figure 3.13 Generated Script View

The generated code has been beautified using hilite.me API for display purposes. Incorrect mappings can be re-mapped using the table provided next to the editor and regenerated. "Execution Reports" tab view consists of the execution reports. The HTML reports have been displayed using a JavaFX web view.

Data Storage

Data such as user input test steps, generated scripts, UI element locator information and report details need to be saved in a suitable format. The following figure illustrates a sample JSON file that contains a test script,

```
[
  {
    "name": "TestCaseOne",
    "scenario": "open url http://semantic-ui.com/examples/login.html.\ntype
      email as abc@email.com.\ntype 123456 in password.\nclick button signIn."
      ,
    "reports": [
    ],
    "locators": [
      {
        "name": "email",
        "locator": "//input[@name='email']"
      },
      {
        "name": "password",
        "locator": "//input[@name='password']"
      },
      {
        "name": "signIn",
        "locator": "//div[contains(text(), 'Login')]"
      }
    ]
  },
```


Figure 3.14 JSON of Test Script

3.4.3 Implementation of the Natural Processing Module

The proposed solution processes test steps input by the user in natural language and convert them into commands in order to generate a script which is executable. As the final step a project containing one or more scripts will be generated as output. The following Figure 3.14 illustrates the steps involved in this process.

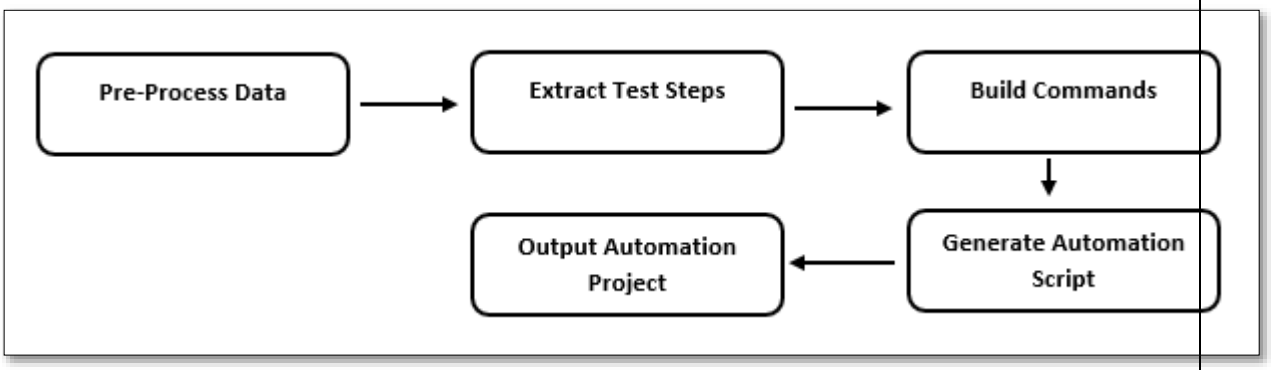


Figure 3.15 Steps involved in the project generation

The implementation of the natural language processing module which carries out the initial text processing task is discussed below.

Pre-Process Data

As discussed in Literature Review chapter both Stanford CoreNLP and Apache OpenNLP have their own advantages. However, considering the time constraints CoreNLP was chosen as the NLP library that would be usedfor the implementation of the NLP module due to the comparatively low programming effort that needs to be put in for its implementation. Primarily, the user input text is separated into sentences and the sentences are separated into words using Stanford NLP Parser. Consequently, the part of speech value for each token in the sentence is assigned using the Stanford POS tagger. Figure 3.15 shows the code snippet corresponding to this task.

```
//this will read scenario from UI and tag word in all sentences using POS Tagger
nlpTagger.buildTaggedSentences(Context.getInstance().getSelectedTestScript().getScenario());
Map<String, String> objectMap = Context.getInstance().getSelectedTestScript().getLocatorsAsMap();
for (int statmentCnt = 0; statmentCnt < nlpTagger
        .getTaggedSenetenceListSize(); statmentCnt++) {
    List<String> verbList = nlpTagger.getVerbFromStatement(statmentCnt);
    for (int verbCnt = 0; verbCnt < verbList.size(); verbCnt++) {
        String verb = verbList.get(verbCnt);
        if(CommandList.vaidateCommand(verb)){
            Statement newStatement = CommandList.getStatment(verb);
```

Figure 3.16 Code Snippet of Pre-Processing Data

The xpath data related to UI object locators are to be provided as a separate excel sheet by the user.

This excel is also read in as raw data. A sample excel sheet containing UI object locators is provided in Figure 3.15. The raw data is stored in an object map with the locator and xpath as key value pairs.

	A	B	C
1	locator	xpath	
2	email	//input[@name='email']	
3	password	//input[@name='password']	
4	signIn	//div[contains(text(), 'Login')]	
5			
6			

Figure 3.17 Sample Excel Sheet containing location information

Extract Test Steps and Build Commands

A custom domain specific identification algorithm has been designed to extract actions, UI elements and test data separately. Actions are used to determine events such as Click and Type in UI automation. The algorithm extracts the verbs from the tokenized output returned by the POS tagger, matches the verb with a corresponding automation command and creates an

instance of the command using the remaining data. Following the determination of the automation command which corresponds to a single sentence, the disambiguation and identification of UI elements and test data is handled by the 'build' method overridden in each subclass of the 'Statement' super class. Only three types of commands have been implemented as subclasses for the prototype due to time and resource constraints. The implementation of the extracting algorithms in these classes have been implemented by taking their application in the test automation domain into consideration.

• ***Open Statement***

The Open Statement performs the UI action of launching a given web page. It can consist of only
one UI element which provides the address of the web page. The pseudo code to extract the UI element from an Open Statement is shown in Figure 3.17,

```
START
    IF wordtype == "verb" && word == "Open" THEN
        FOR (0 TO remaining_words.count)
            IF remaining_word instanceOf noun THEN
                IF Objectmap has remaining_word THEN
                    URL = remaining_word
                ELSE
                    IF remaining_word startsWith "http://" THEN
                    URL = remaining_word
                ENDIF
            ENDIF
        RETURN URL
    ENDIF
END
```

Figure 3.18 Pseudo Code for Open Statement

The Java implementation of the above pseudo code is included within the “build” method of Open Statement as follows,

```
package commands;

import java.util.List;
import java.util.Map;

import edu.stanford.nlp.ling.TaggedWord;

public class OpenStatement extends Statement {

    @Override
    public Statement build(List<TaggedWord> restTags, Map<String, String> ObjectMap) {

        this.setName("Open");
        for (TaggedWord tw : restTags) {
            if (tw.tag().startsWith("NN")) {
                String key = tw.word();
                if (ObjectMap.containsKey(key)) {
                    this.getAttaributes().put("URL", ObjectMap.get(key));
                    break;
                } else if (key.contains("http://")) {
                    this.getAttaributes().put("URL", key);
                    break;
                }
            }
        }
        return this;
    }
}
```

Figure 3.19 Build method implementation of Open Statement

- ***Type Statement***

The Type Statement performs the UI action of typing text in a text field or text area in the web interface. It can consist of an UI element which provides the xpath locator of the text field and test data which is the content that needs to be typed in the given text field. The pseudo code to extract this information from a tokenized sentence is shown in Figure 3.20

```
START
    IF wordtype == "verb" && word == "Type" THEN
        FOR (0 TO remaining_words.count)
            IF Objectmap has remaining_word THEN
                element = remaining_word.locator
                IF other_remaining_word instanceOf preposition THEN
                    SPLIT sentence BY other_remaining_word
                    FOR (0 TO split_half_without_element.count)
                        IF word instanceOf noun THEN
                            test_data = word
                        ENDIF
                    END
                ENDIF
            ENDIF
        END
    ENDIF
END
```

Figure 3.20 Pseudo Code for Type Statement

In the above pseudo code, first the token list is traversed to find the UI element by matching each word with the Object map till a match is found. Once the UI element is determined, the token list is split into two using a preposition. The split half without the UI element is then traversed to find a noun which is considered as test data. A partial code snippet of the implementation of the above pseudo code in the "build" method of the Type Statement is given below,

```java
package commands;

import java.util.ArrayList;

public class TypeStatement extends Statement{
    private static final List<String> prepositions = new ArrayList<String>(Arrays.asList("IN", "AS"));
    @Override
    public Statement build(List<TaggedWord> restTags, Map<String, String> ObjectMap) {
        this.setName("Type");
        String foundOne = null;
        int foundPos = 0;
        // check whether tag contains;
        for(int count =0; count < restTags.size(); count++){
            TaggedWord tag = restTags.get(count);
            if(tag.tag().startsWith("IN") && prepositions.contains(tag.word().toUpperCase() )){
                foundOne = tag.word();
                foundPos = count;
                break;
            }
        }
        this.getAttaributes().put("uiObject", getMappingValue(restTags,foundPos,foundOne,ObjectMap));
        this.getAttaributes().put("value", getTextWord(restTags,foundPos, foundOne));
        return this;
```

Figure 3.21 Build method implementation of the Type Statement

- ***Click Statement***

The Click Statement fires a click on a given UI element. It consists of one UI element. The pseudo code of the Click Statement is depicted in Figure 3.22,

```
START
    IF wordtype == "verb" && word == "Click" THEN
        FOR (0 TO remaining_words.count)
            IF remaining_word instanceOf noun THEN
                IF Objectmap has remaining_word THEN
                    element = remaining_word
                ENDIF
            ENDIF
        RETURN element
    ENDIF
END
```

Figure 3.22 Pseudo Code for Click Statement

The pseudo code of the Click Statement is similar to the one implemented in the Open Statement.

Instead of the URL, the noun identified in this token list is considered as the UI element. The code snippet corresponding to the implementation of the above pseudo code follows,

```
package commands;

import java.util.List;
import java.util.Map;

import edu.stanford.nlp.ling.TaggedWord;

public class ClickStatement extends Statement {
    @Override
    public Statement build(List<TaggedWord> restTags, Map<String, String> ObjectMap) {
        this.setName("Click");
        for (TaggedWord tw : restTags) {
            String key = tw.word();
            if (tw.tag().startsWith("NN") && ObjectMap.containsKey(key)) {
                this.getAttaributes().put("uiObject", key);
                break;
            }
        }
        return this;
    }
}
```

Figure 3.23 Build method implementation of the Click Statement

3.4.4 Implementation of Script Generation Module

Two templates have been designed for script generation and object page generation using Apache Velocity template standards. Apache Velocity is a Java based template engine which allows referencing of Java objects within the template itself. The template used for script generation is shown below in Figure 3.23

```
package org.testcases;

import java.util.HashMap;
import java.util.List;

import org.objectmap.UIObjects;
import org.runtime.SeleniumTestBase;
import org.testng.annotations.Test;
import org.testng.annotations.DataProvider;


public class tc_login extends SeleniumTestBase {


    @Test
    public final void tc_001() throws Exception {

    #foreach( $statment in $statments )

        #if( $statment.getName() == 'Open' )
            open("$statment.getAttaributes().get("URL")", "3000");
        #elseif( $statment.getName() == 'Type' )
            type(UIObjects.$statment.getAttaributes().get("uiObject"), "$statment.getAttaributes().get("value")");
        #elseif( $statment.getName() == 'Click' )
            click(UIObjects.$statment.getAttaributes().get("uiObject"));
        #else

        #end

    #end


    }

}
```

Figure 3.24 Script template designed using Apache velocity templates

The list of statements output from the NLP module is passed as a parameter to the template engine which will traverse the list and generate the line corresponding to the name of each statement in the list using the “attributes” property of the “Statement” object. The code snippet presented in Figure 3.25 contains the implementation of the initialization of Velocity engine and automation script generation using the above template.

```
private void createTestCaseFile(String fileLoc, List<Statement> statments) {
    /* first, get and initialize an engine */
    VelocityEngine ve = new VelocityEngine();
    ve.init();
    /* next, get the Template */
    Template t = ve.getTemplate(Util.TEMPLATE_TESTCASE);
    /* create a context and add data */
    FileWriter writer = null;
    try {
        writer = new FileWriter(new File(fileLoc + ".java"));
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    VelocityContext context = new VelocityContext();
    context.put("statments", statments);
    t.merge(context, writer);
    try {
        writer.flush();
        writer.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Figure 3.25 Code Snippet for velocity engine initialization

Similarly, the following template in Figure 3.26 has been used for the generation of object data page which belongs to a test script.

```
package org.objectmap;

public class UIObjects {

#foreach( $locatorKey in $locators.keySet() )

#set($tag = '##')
public static final String $locatorKey = "$locatorKey$tag$locators.get($locatorKey)";

#end


}
```

Figure 3.26 Object page template designed using Apache Velocity templates

This page is generated using the object map which contains object data uploaded by the user. The generated scripts and object pages are integrated into the automation framework which exists as a project within the application folder.

3.4.5 Implementation of Executing and Reporting Module

The automation framework above contains the implementation of the execution and reporting module of the solution. It has been developed according to the proposed architecture in 3.4.4. Primarily, Apache Ant have been used for the purpose of building and compiling the "Automation" project which contains the generated scripts. The core of this module is contained within the "ExecutionHelper" class which extends "SeleniumTestBase" in the "org.runtime" package. The "ExecutionHelper" class initializes the web driver and reporting engine in preparation for execution using TestNG annotations (Figure 3.27). TestNG has been used for this purpose due to its many benefits. It provides annotations to specify tasks that needed to be completed before and after test execution. Additionally, it generates HTML reports of the execution (Toolsqa.com, 2017).

```
@BeforeTest
public final void setUp() throws Exception {


    this.configWebDriver("chrome");
    this.startBrowserSession("chrome");
};

@BeforeMethod
public final void setTestContext(final Method method) {
    this.initReport(method.getDeclaringClass().getSimpleName());

}
```

Figure 3.27 Web driver configuration and report initialization using TestNG

Subsequent to the initialization, the content annotated with "@Test" annotation, which in this case is the content of the generated script, is executed by TestNG. The generated script contains Selenium wrapped command methods which will fire UI events on the web interface. The purpose of the wrapper method is optimized error handling and result reporting. If an exception occurs, a retry country has been specified to define the retry attempts before failing a command. The wrapper methods of the three commands that have been implemented in this prototype, Open, Type and Click are included in the "SeleniumTestBase" class. Figure 3.28 provides a code snippet of the implementation of the wrapper method for Type command.

```
public final void doType(final ObjectLocator locator, final Object objValue) {
    int counter = getRetryCount();
    int retryInterval = RETRY_INTERVAL;
    // Getting the actual object identification from the object map
    String objectID = locator.getActualLocator();
    try {
        // Check whether the element present
        WebElement element = checkElementPresence(objectID);
        /* Following for loop was added to make the command more consistent try the command
         * for give amount of time (can be configured through class variable RETRY_INTERVAL)
         * command will be tried for "RETRY" amount of times or until command works. Any exception
         * thrown within the tries will be handled internally.
         * can be exited from the loop under 2 conditions 1. if the command
         * succeeded 2. if the RETRY count is exceeded
         */
        try {
            element.clear();
        } catch (Exception e) {
            e.printStackTrace();
        }
        while (counter > 0) {
            try {
                counter--;
                // Calling the actual command
                element.sendKeys(objValue.toString());
                break;
            } catch (StaleElementReferenceException staleElementException) {
                element = checkElementPresence(objectID);
            } catch (Exception e) {
                sleep(retryInterval);
                if (!(counter > 0)) {
                    e.printStackTrace();
```

Figure 3.28 Code snippet for Type command wrapper method

3.5 System Testing

3.5.1 Functional Testing

Table 3.3 provides the list of functional requirements along with their implementation and test status.

Requirement	Implementation Status	Test Status
Create test script	Implemented	Tested
Generate automation project from test scripts	Implemented	Tested
Execute automation script	Implemented	Tested
Generate result report	Implemented	Tested
View result report of executed script	Implemented	Tested
Upload object data to repository	Implemented	Tested
Edit test script	Implemented	Tested

Delete test script	Implemented	Tested

Table 3.3 Implementation and Test Status of Functional

3.5.2 Non-Functional Testing

The non-functional testing of the system was carried out using either quantitative testing or qualitative testing approach as required.

Usability Testing (Qualitative)

Requirement	**Implementation Status**	**Test Status**
Create test script in natural language.	Implemented	Tested
Display execution results clearly.	Implemented	Tested
Should give clear indication of errors and other notifications.	Implemented	Tested

Table 3.4 Implementation and test status of Usability Testing

As indicated in table 3.4, three non-functional requirements pertaining to the usability of the solution have been implemented and tested. Usability testing was carried out qualitatively since it is a measure of quality therefore quantitative values cannot be obtained.

Accuracy Testing (Quantitative)

Accuracy is an important requirement since the solution depends greatly on its accuracy level. Once a user has input plain text test steps, it is important that the solution is capable of mapping it to an appropriate automation command. In order to determine the accuracy of the natural language test step conversion to executable statements, test steps that conform to each of the three commands that have been implemented were tested separately since they have separate statement specific data extraction algorithms implemented as presented in Table 3.5,

Statement	**No of Test cases**	**No of Test cases passed**	**Accuracy (%)**
Open Statement	30	27	90%
Type Statement	30	25	83%
Click Statement	30	24	80%
Overall Accuracy			84.33%

Table 3.5 Accuracy of the test results

Chapter 4 - EVALUATION AND RESULTS

4.1 Evaluation Criteria

The evaluation of the developed solution will be carried out based on pre-determined criteria as

presented in below table.

Criteria	Description
Approach	The efficiency of addressing the problem and relevance of solution needs to be discussed with the aim of evaluating the approach taken to resolving the research problem.
Design and Architecture	The design and architecture of the solution needs to be evaluated to determine if it is correct and suitable.
Impact	The impact of the provided solution on the problem domain and its effect on existing solutions has to be evaluated.
Usability	The usability of the provided solution by users with no technical background as well as experienced users needs to be evaluated.
Enhancements	Enhancements that can be made in order to improve the efficiency, impact and usability of the solution provided needs to be identified through evaluation.

Table 4.1 Evaluation criteria of developed model

4.2 Evaluation Methodology

The model was presented to stakeholders and domain experts for direct hands on experience and feedback was obtained regarding the model provided with relevance to the research problem, its effectiveness and drawbacks. The feedback received contributes to determining the impact, efficiency and usability of the provided solution.

4.3 Evaluation by Users

The prototype was presented to 30 evaluators who were associated with the QA domain having different amounts of experience in the industry and different levels of technical expertise for evaluation. Also, Testsigma tool was given to evaluators to compare the prototype with the tool. An Online Questionnaire based on the evaluation criteria that was pre-determined was conducted following the prototype demonstration.

Online Questionnaire

Question	**Purpose of including the question in the survey**
How many years of experience do you have related to web application testing?	To get an understanding about the subjects of the survey and estimate the reliability and accuracy of their answers based on experience.
Which testing approaches are you familiar with?	Adding to reliability to ensure that the answers provided are unbiased and valid.
If you are familiar with automated testing, what would you say is the greatest challenge/s in conventional test automation?	Focusing on collecting the respondents' opinion regarding the challenges of test automation that have been previously identified and any additional challenges he/she would like to specify
What is your preferred web application test automation solution?	To clarify the current validity of Selenium in the test automation domain and to gather information regarding better frameworks that may exist and are currently being used in the industry.
What is the scripting technique used in the solution mentioned above?	To identify the current level of technical experience expected in test script creation.
Would you prefer a framework which interprets manual test steps in natural language (English) to generate test scripts for web application test automation?	To estimate the respondents' willingness to adopt the proposed solution.
Do you know of any tool or framework which provides the above functionality?	To identify existing solutions that the subjects of the survey consider as similar solutions in addition to the ones that have already been identified.
If 'Yes' Please list here	
Have you tried Testsigma (NLP based automation tool) before?	What is the most user-friendly application?

In which tool can you do the scripting easily?	To identify that which tool allows to script easily.
What is the most user friendly application?	To clarify from which tool user can do the functions easily or what is the most usability tool
Is it good to show the generated automation script to user?	To clarify the new feature is important to users.
What is the best tool based on the performance?	To estimate that which tool is performed well.
Any enhancements needed?	To get the enhancements for future work

Table 4.2 Online Questionnaire

4.4 Quantitative Evaluation

In order to evaluate the solution quantitatively, three QA engineers with approximately the same level of technical expertise and experience in the industry were chosen at random. One QA engineer was provided with the prototype. The second QA engineer was asked to use their general approach in web application test automation, using Selenium framework. Third QA engineer was provided the Testsigma automation tool. All were provided with the same manual scenario as follows,

```
1  Open ebay.com
2  Click on Sign In link
3  Type username as "john.doe"
4  Type password as "qwerty"
5  Click Sign In button
6
```

Figure 4.1 Manual Automation Scenario

The time taken by the QA Engineer was less than 5 minutes. All he had to do was copy the scenario provided into the editor, capture the relevant UI object to compose the excel sheet that needs to be uploaded and click “Generate” followed by “Execute”. The QA Engineer who used the Testsigma took 7-8 minutes since the tool is not providing the copy and paste the test steps from other tests he had to type the scenario and capture the UI elements. The QA Engineer on

the other hand took between 12-15 minutes for the automation of this simple scenario using Selenium since he had to write the code for automation in addition to capturing UI elements. This test was repeated for two other scenarios which yielded similar results. Also, it is very user friendly. The above scenario was automated with only four steps. Therefore, it can be quantitatively proved that the provided solution has a positive effect in test automation.

4.5 Critical Evaluation of the System

The proposed solution was to address the research problem of requiring specifically trained human resources with a considerable amount of technical expertise in the domain of test automation in web applications. The intention of the solution is to provide a less technical framework to act as the middle man between a QA Engineer and the automation platform. This way the cost and time spent on training resources in test automation is reduced and automation is encouraged as a beneficial mean of testing web applications.

The requirements for the solution were gathered directly from the stakeholders of the system themselves and also by studying a number of literatures works related to the problem domain. This paved way to developing solution which directly addresses the problem. The design was based on the research which was originally carried out regarding tools and techniques that are to be applied to increase the efficiency and effect. The noted drawback in the architecture was making it a thick client application instead of a web application. As observed in latter stages it would have been better if the solution was a web application which would have been even more beneficial with regards to time and resource saving when automating web applications.

The prototype has been developed as proof of concept that the proposed solution will efficiently address the problem in question. The relevance of the solution was validated by experts who agreed the solution was an interesting approach to handle the research problem. The core functionalities of the prototype include creation of test scripts in English, generation of automation scripts and the execution of automation scripts to obtain result reports. This framework is easier to use in comparison with the conventional method of test automation in web applications which requires a certain amount of coding on the user's behalf whereas this framework only requires plain English text as input. English is highly opted over code an input by any user of the system and therefore the issue of technical knowledge has been directly addressed. Also, the GUI of the framework is clear and simple therefore usability is guaranteed. Generated automation scripts can be executed repeatedly by making slight changes to data and therefore requires less scripting effort by a QA Engineer which is very convenient in saving

time. Furthermore, the performance of the prototype decided based on the time taken to generate an automation script has yielded satisfactory results. Accuracy has room for improvement if the POS tagger is trained with larger text corpuses containing more sample data. Overall however, as a proof of concept, this approach can be further explored as a potential solution to address the research problem by integrating the suggested improvements.

4.6 Results

Developed prototype and Testsigma tool were given to 30 QA Engineers and collected the responses via the Online Questionnaire.

Participants of the survey were coming from different levels in the industry such as QA Engineer, Senior QA Engineer, Associate QA Lead, QA Lead, Senior QA Lead, Senior Test Automation Engineer and Senior Test Automation Lead. More coding, maintenance the automation code, Reliability of the automation code and cost of the time are common issues of test automation identified through the survey. Since the problems they were facing in the test automation, 66.3 % of the participants agreed that they prefer to have a framework which interprets the manual test steps in natural language (English) to generate test scripts for web application test automation.

Through the survey identified that NLP based test automation platforms are not very much popular as 76.7 of the participants % didn't know about such kind of tools or platforms.

Script Generated View is the new module added to prototype that is not exists in Testsigma tool. 100% of the participants are agreed that it is very important to have in the model. Then we can thoroughly evaluate that the new feature is more effective.

83.3% of are told that it is very easy to script with new model rather than the Testsigma because editor allows to copy and paste the scenarios.

Survey has been included some nonfunctional related scenarios such user – friendliness, performance. 56.7% of participants are told that proposed model is more user friendly than Testsigma and 62.1% of are determined that model is performed well than the Testsigma.

Chapter 5 - CONCLUSION AND FUTURE WORK

5.1 Problems and Challenges Encountered

Since the domain of test automation has been a recent topic of interest in the software industry researching on the problem domain was not very difficult. However, finding the optimum solution for the research problem which would adequately address the research problem was a challenge. The solution provided had to be efficient, effective with a high level of usability. To factor in all those to a single solution was the initial challenge faced during this research project. Another problem faced was how plain English would be processed to generate Selenium code. NLP was considered for this purpose however; an NLP library alone was not sufficient. Therefore, an algorithm was designed to convert test steps extracted by the NLP library into Selenium wrapped automation commands. Lack of a domain specific text corpus was another problem faced.

When using the POS tagging, Stanford POS tagger was used for the purpose of assigning part of speech tags to each word or token input by the user. However, the English tag set used by the POS tagger by default, which is provided by the Penn Treebank site, is unable to assign the correct part of speech values for statements that are generally used in the test automation domain. As an example, in the statement "Type username in textbox", if properly tokenized, 'type' should be a verb, however the general POS tagger assigns NN which stands for noun to 'type'. Therefore, as a solution, the POS tagger model was specifically trained using a custom corpus defined sssby the developer which contains a few sample statements which are generally used in test automation.

A user may type in synonyms of the verbs as actions. For example, "launch", "navigate" are all synonyms of the verb "open". These words need to be correctly mapped to the corresponding command. For this purpose, a dictionary synonym has been compiled. It consists of a command which represents the action and the list of synonyms. If an immediate match is not found, the synonym dictionary will be traversed to find the action that corresponds to the noun. If a match is not found in the synonym map, then the user is prompted to perform the matching task. Once the user maps a word to the corresponding command the word is added to the synonym dictionary as a synonym.

5.2 Learning Outcomes

Both technical and soft skills were improved as outcomes of the research project. They can be listed as follows,

- Soft skills such as critical thinking, problem solving, time management and creativity was improved.
- Programming skills using technologies such as Java, Java FX, Selenium and TestNG.
- Techniques used in text processing such as POS tagging using Stanford NLP library was studied.
- Training of a POS tagger using specified text corpus was practiced.
- Extensive knowledge regarding test automation in general and more specifically in the web
  application domain was obtained.
- Documentation skills regarding report writing of a research project was developed
- Algorithm design and implementation skills were sharpened.
- Critical evaluation based on qualitative and quantitative measures was studied.

5.3 Limitations of the Solution

This solution has been designed as a thick client application. Therefore, it has to be individually installed to each user's system. This is a limitation of the system. Since the prototype was developed for proof of concept purposes, a database has not been integrated. Database integration leads to added complexity. However, if the solution is to be converted to a web application, a database needs to be integrated in place of the JSON files that currently store data. Also, the scope of the solution was initially limited to test automation in web applications. It cannot be applied to mobile web and mobile native applications.

5.4 Future Enhancements

The feedback received by external evaluators and the critical evaluation of the system lead to the
determination of improvements that can be made to the provided solution for better outcomes. Accordingly, the following enhancements have been suggested for this solution,

- Integration of an Object Spy
- Integrate more selenium commands

5.4.1 Integration of an Object Spy

An object spy is an object capturing tool used in test automation for the purpose of extracting XPaths and another locator information from an UI element through a single click. Although there are many object capturing tools available in the software market today, it would be easier if the object spy feature is integrated into the application. In that manner a user would be able to populate the object data table in the application by using the object spy feature itself instead of using a separate tool to capture XPaths and create an excel to be uploaded to the system.

The object spy module will be added to the high-level design as shown below,

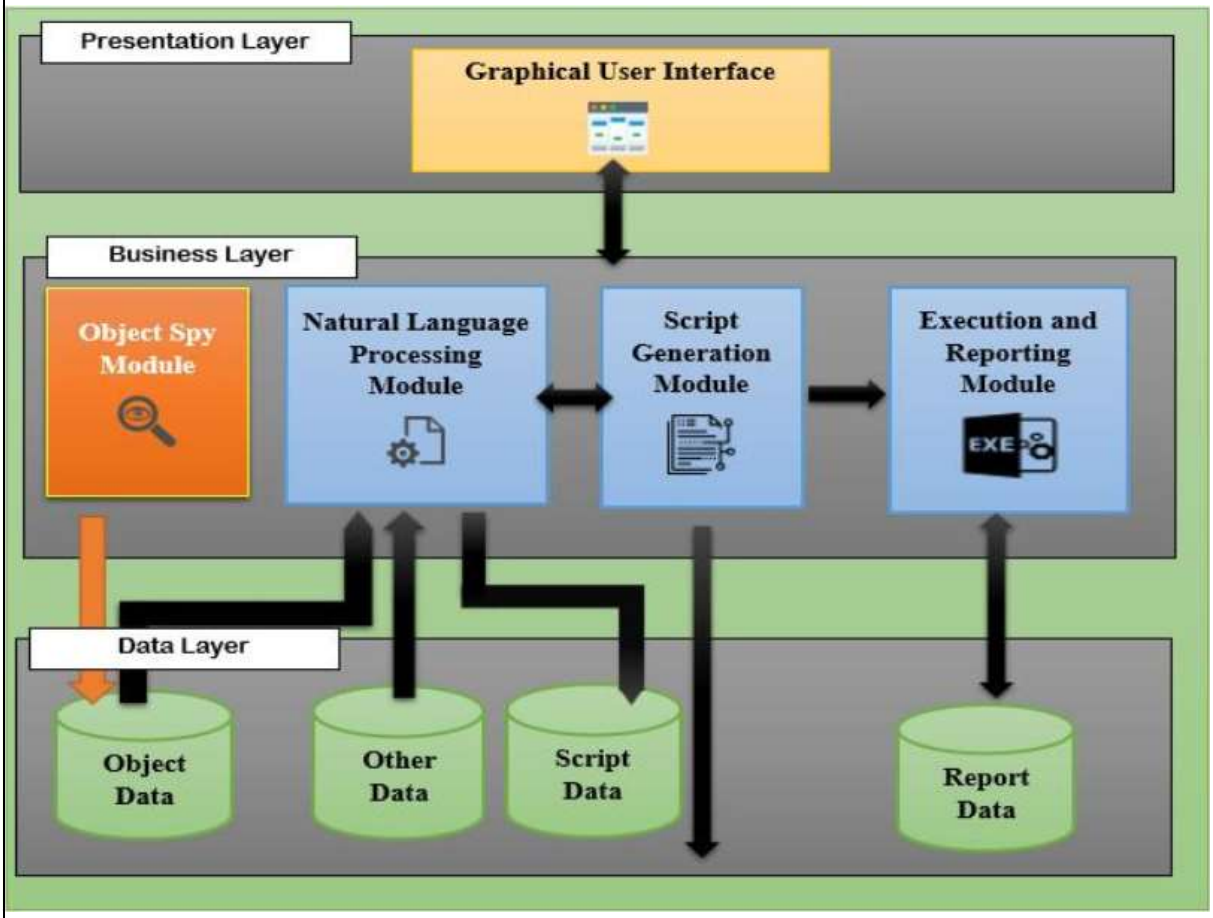


Figure 5.1 Updated High-Level Architecture Diagram

5.4.2 Integrate more selenium commands

Selenium framework supports for multiple commands to interact with the elements. Open, Type, Click commands have been already implemented in the prototype and most of the element events can be captured from these commands. Other than that, there are several commands in selenium framework such as Wait, Element Disabled, Element Enabled, Element Displayed. By adding those commands user can develop more advanced scenarios

Chapter 6 **APPENDICES**

Appendix A: Summary of Online Questionnaire Responses

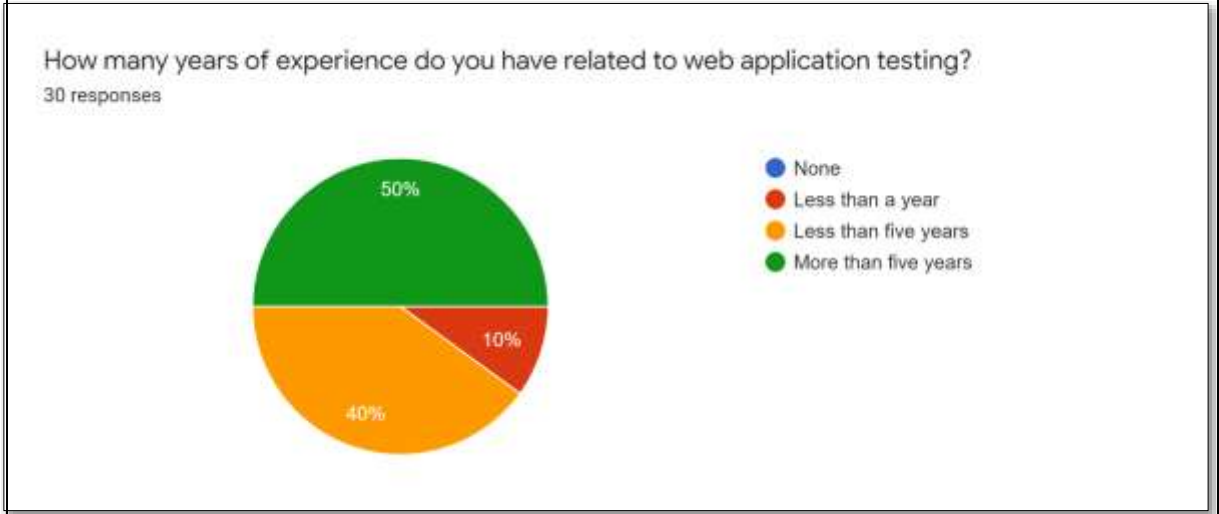


Figure 6.1 Results of Q02 in Questionnaire

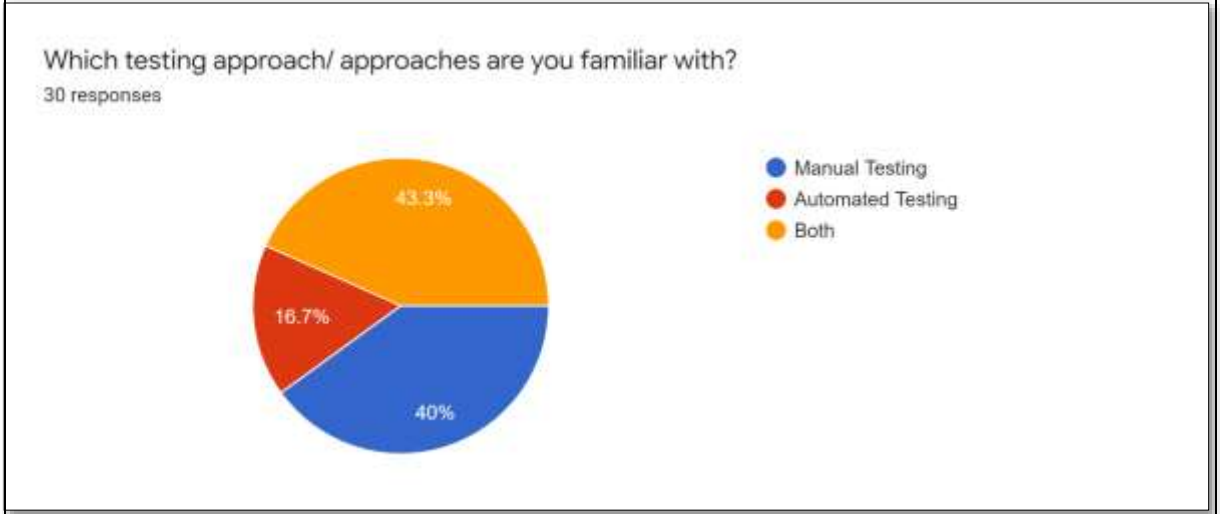


Figure 6.2 Results of Q03 in Questionnaire

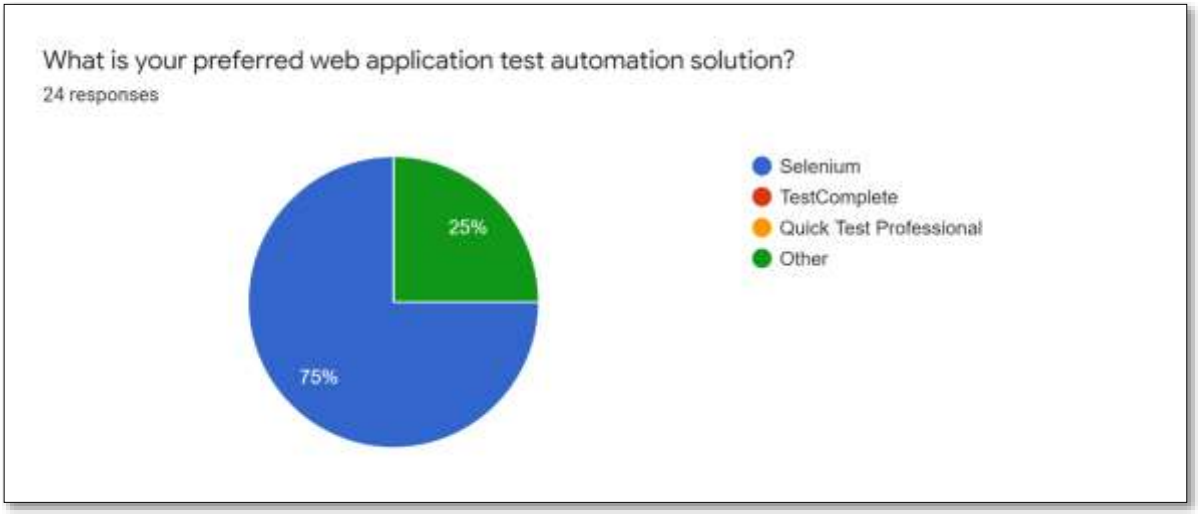


Figure 6.3 Results of Q05 in Questionnaire

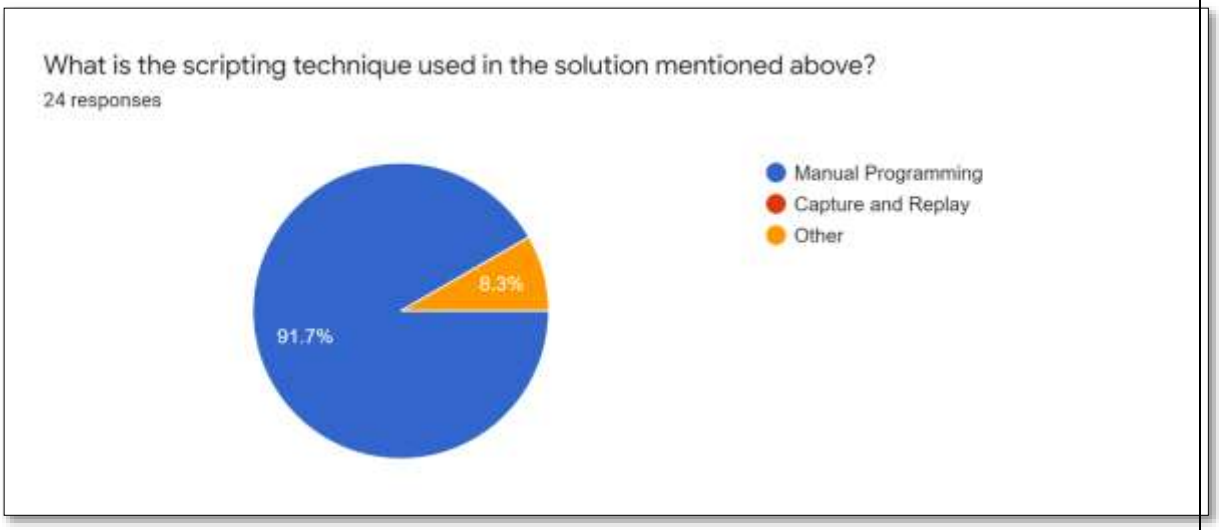


Figure 6.4 Results of Q06 in Questionnaire

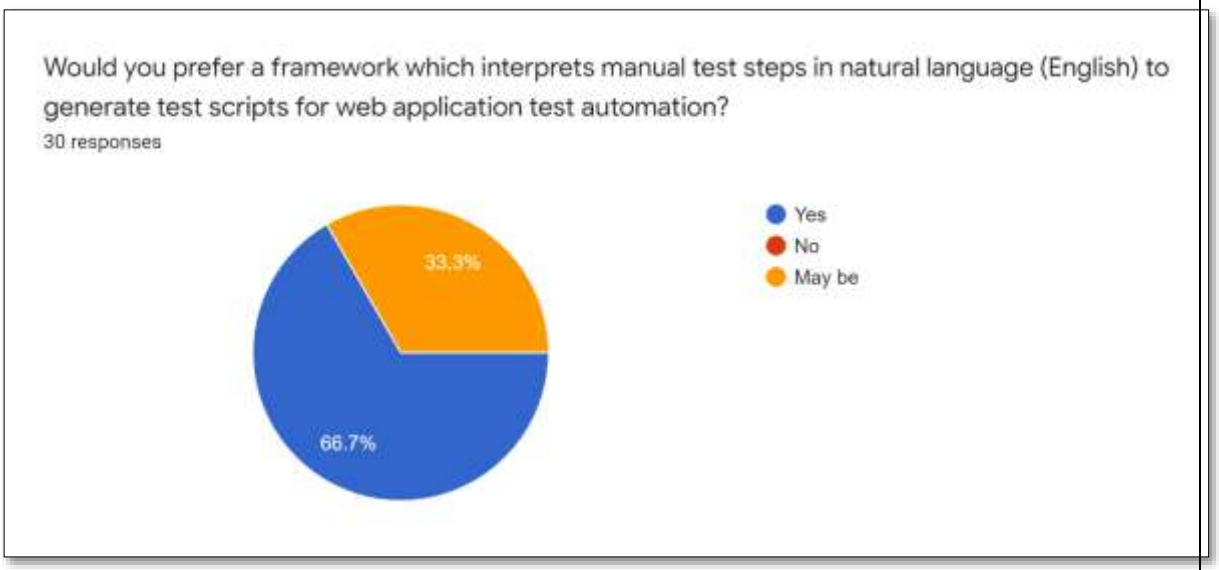


Figure 6.5 Results of Q07 in Questionnaire

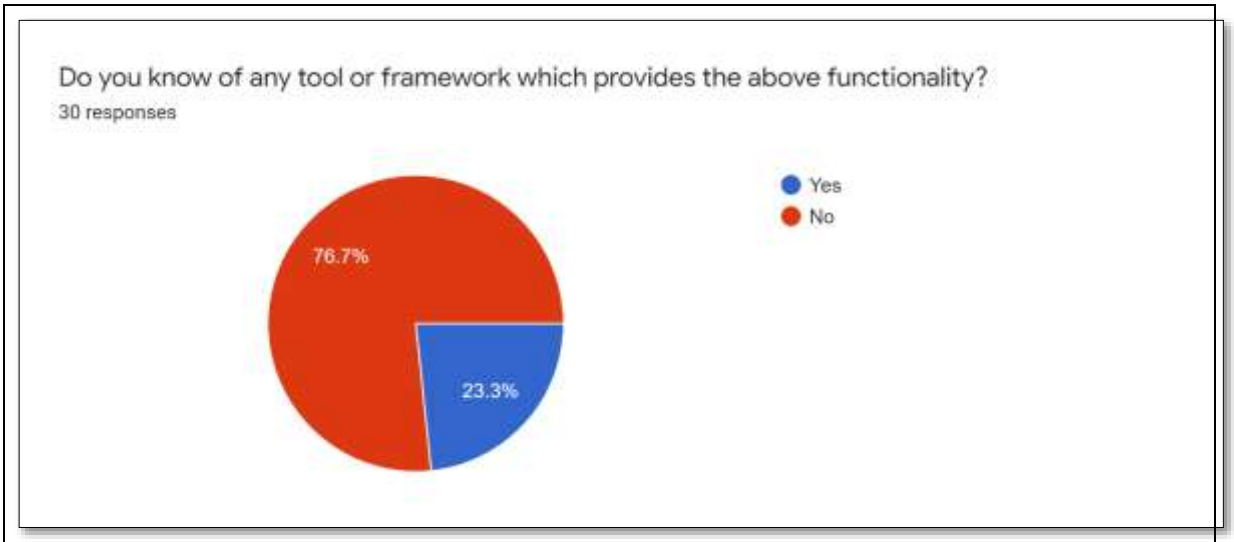


Figure 6.6 Results of Q08 in Questionnaire

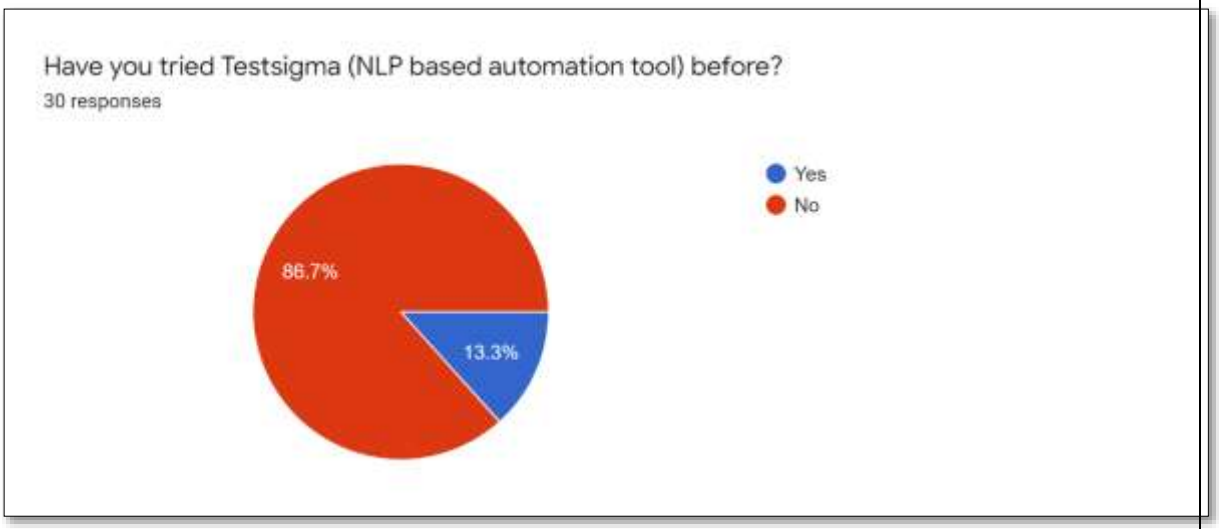


Figure 6.7Results of Q09 in Questionnaire

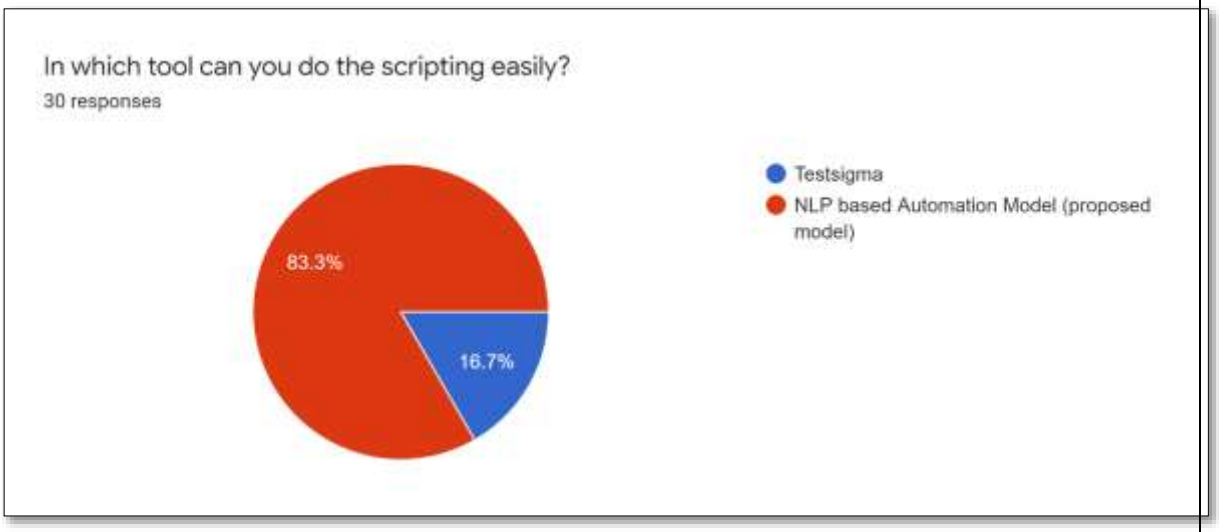


Figure 6.8 Results of Q10 in Questionnaire

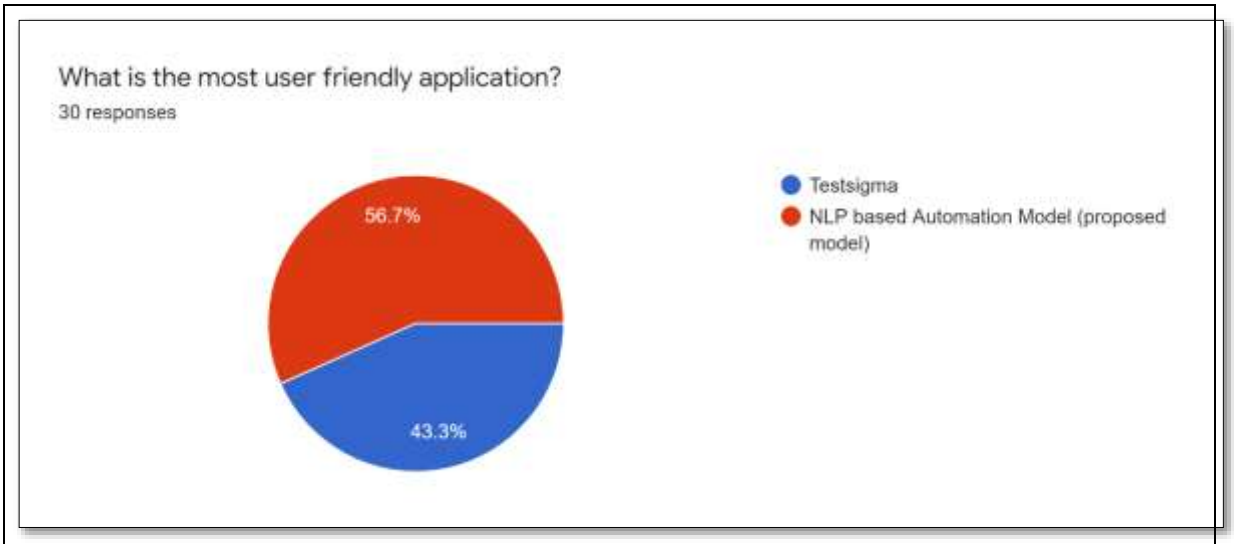


Figure 6.9 Results of Q11 in Questionnaire

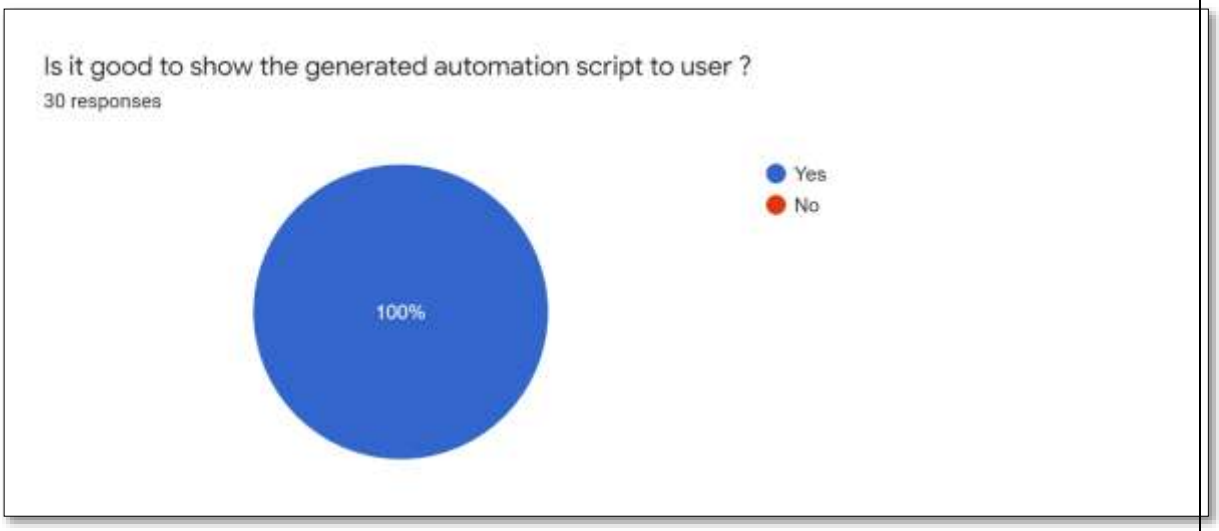


Figure 6.10 Results of Q12 in Questionnaire

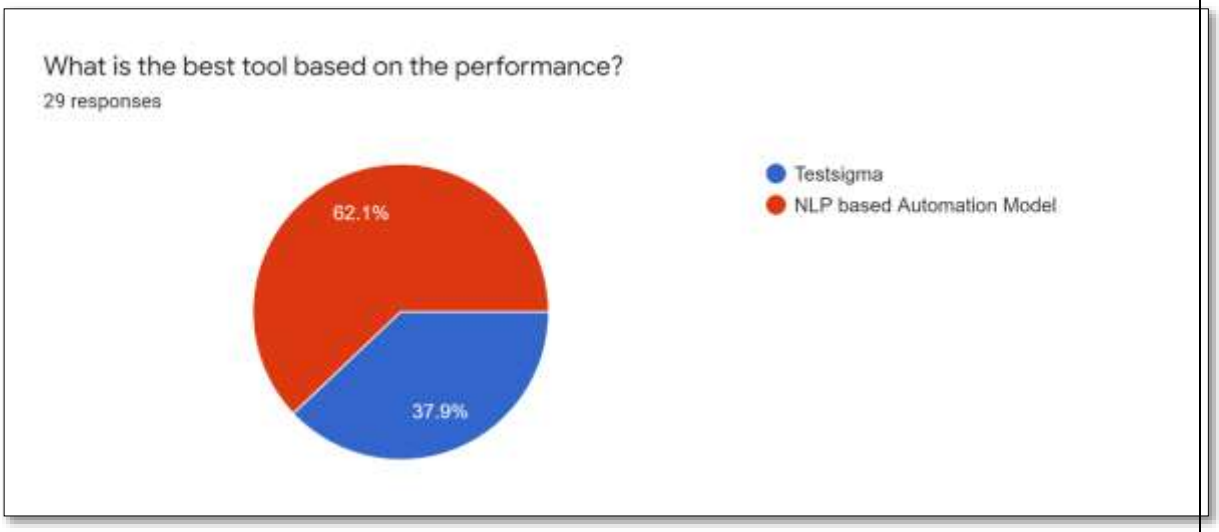


Figure 6.11 Results of Q13 in Questionnaire

Appendix B: Interviewee Background Status

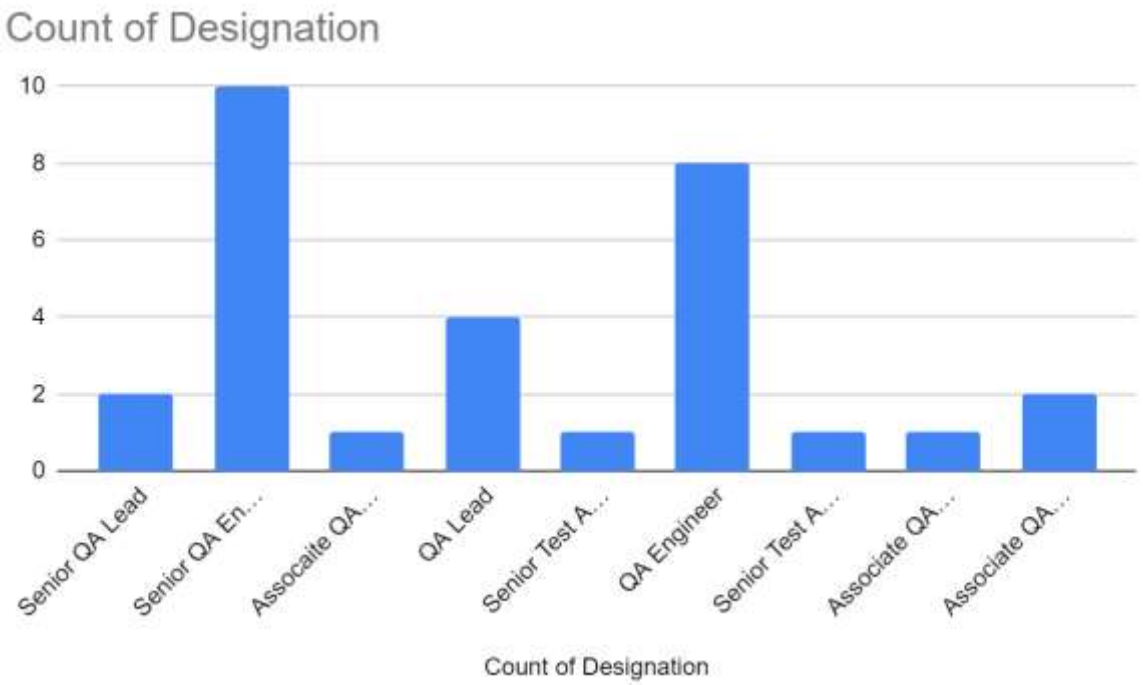
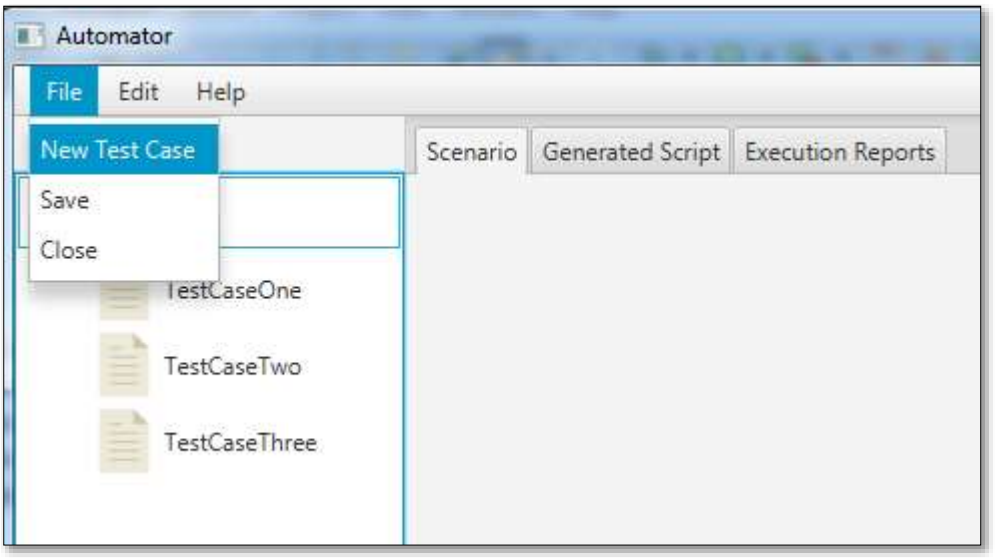


Figure 6.12 Interviewee Background Status

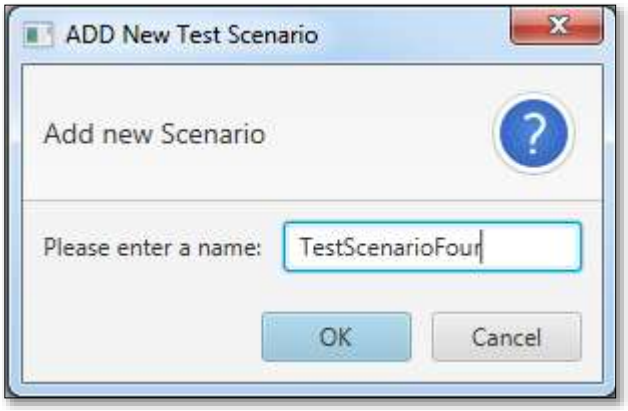
Appendix C: User Manual

C.1 Create New Test Script

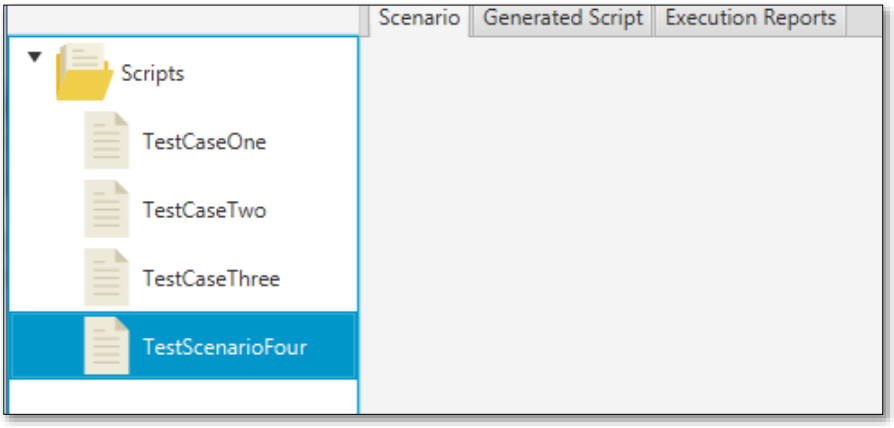
a. Click file menu and click New Test Case.



b. Enter new name and click OK (make sure name is unique)

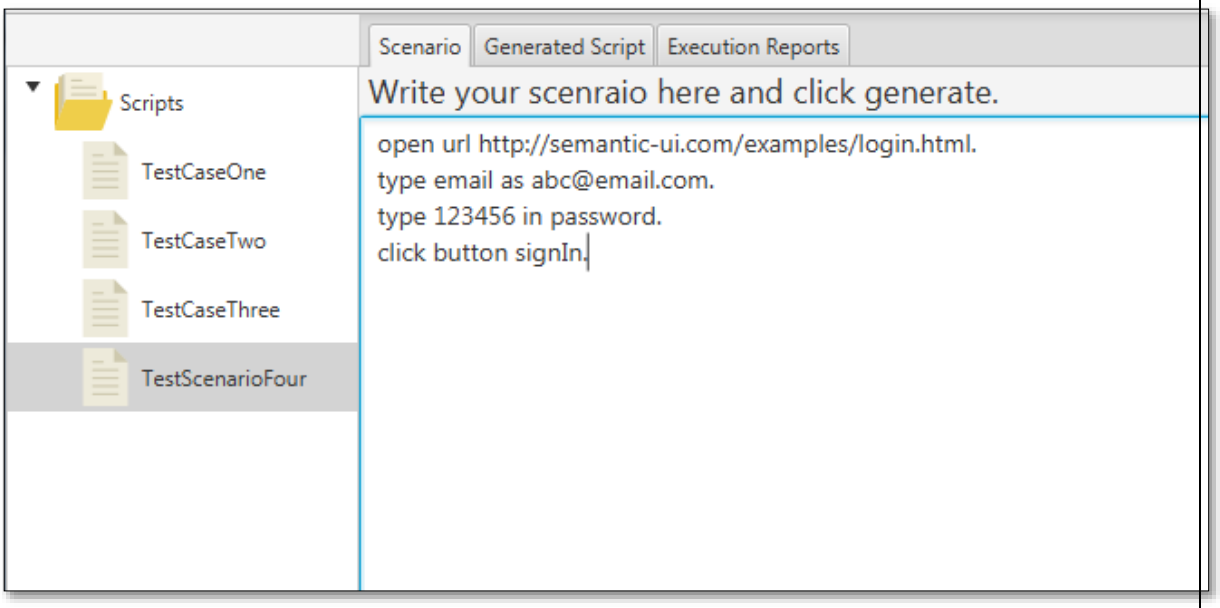


c. A new test script will be added as follows,



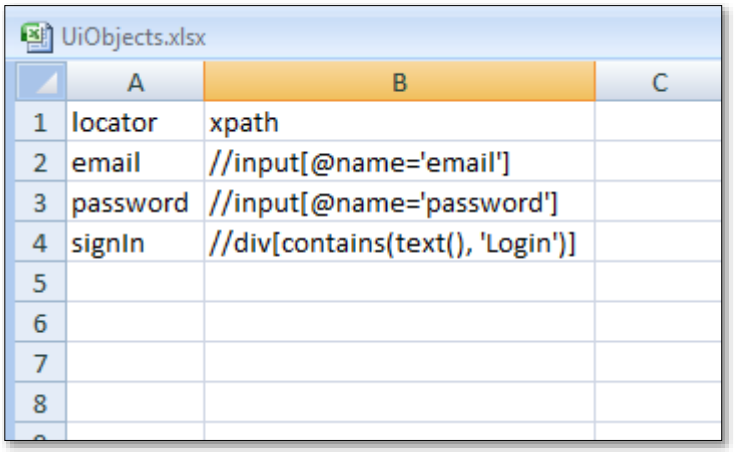
C.2 Write Scenario and upload Object Excel file.

a. Write/Script your scenario in text area.



b. Create an excel file

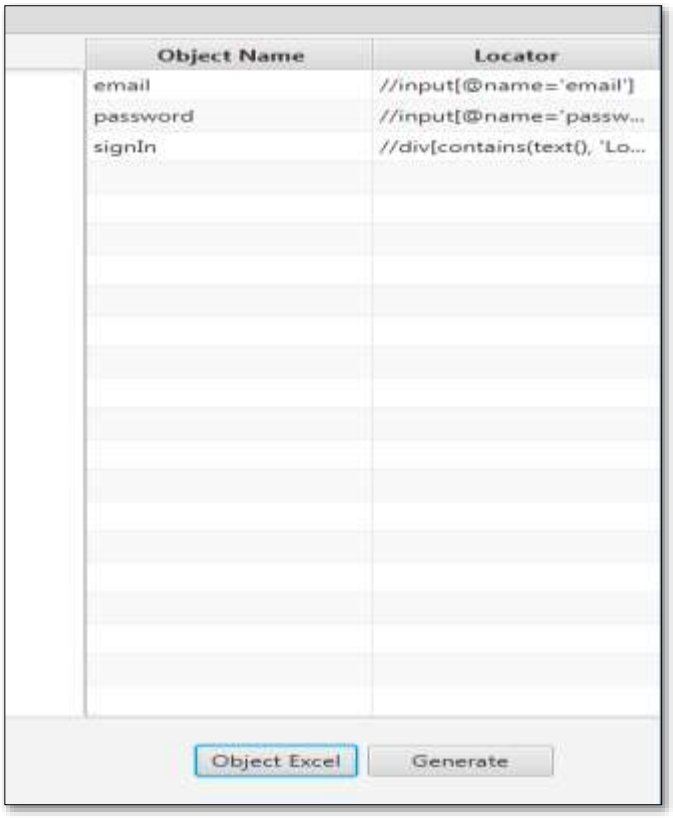
Note: The first row should have two columns (locator, XPath). Fill your objects logical name in locator column and xpath of object in xpath column. **Note**: make sure logical name is not included whitespaces



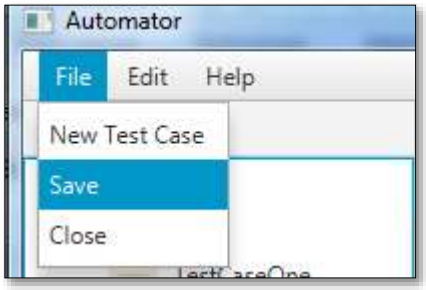
	A	B	C
1	locator	xpath	
2	email	//input[@name='email']	
3	password	//input[@name='password']	
4	signIn	//div[contains(text(), 'Login')]	
5			
6			
7			
8			

c. Upload excel file to editor using button Object Excel. Uploaded Object Data will be displayed in table.

Note: locator names included in the scenario should match the locator names uploaded in the excel file.

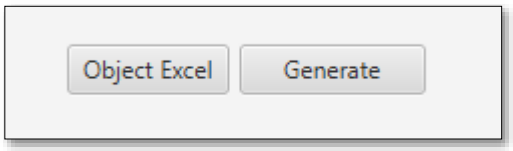


d. Click File menu and click Save to save the selected test script



C.3 Script Generation and Execution

a. After writing scenario and uploading object excel file click generate button in Scenario tab and user will be redirected to Generated Script tab.

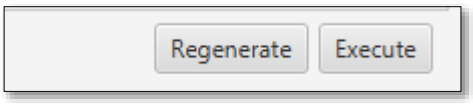


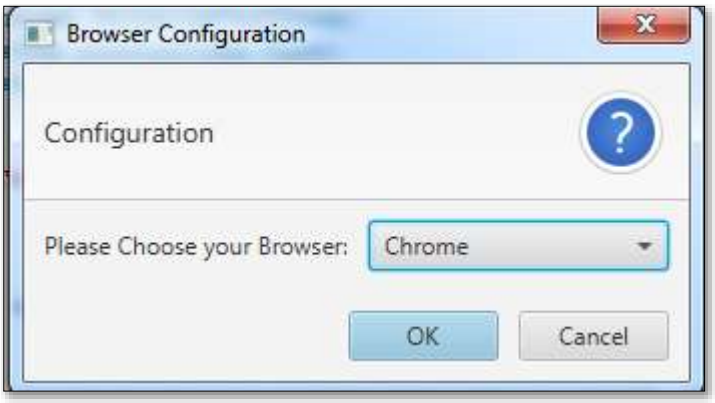
b. Generated code will be displayed in Generated Script tab.

Scenario | Generated Script | Execution Reports

```
package org.testcases;
import java.util.HashMap;
import java.util.List;
import org.objectmap.UIObjects_TestScenarioFour;
import org.runtime.SeleniumTestBase;
import org.testng.annotations.Test;
import org.testng.annotations.DataProvider;


public class TestScenarioFour extends SeleniumTestBase {


 @Test
 public final void test() throws Exception {


      open("http://semantic-ui.com/examples/login.html", "3000");


      type(UIObjects_TestScenarioFour.email, "abc@email.com");


      type(UIObjects_TestScenarioFour.password, "123456");


      click(UIObjects_TestScenarioFour.signIn);

            }
}
```

c. After mapping verb with commands click regenerate otherwise click button execute and choose the browser (default will be chrome browser) to start the execution

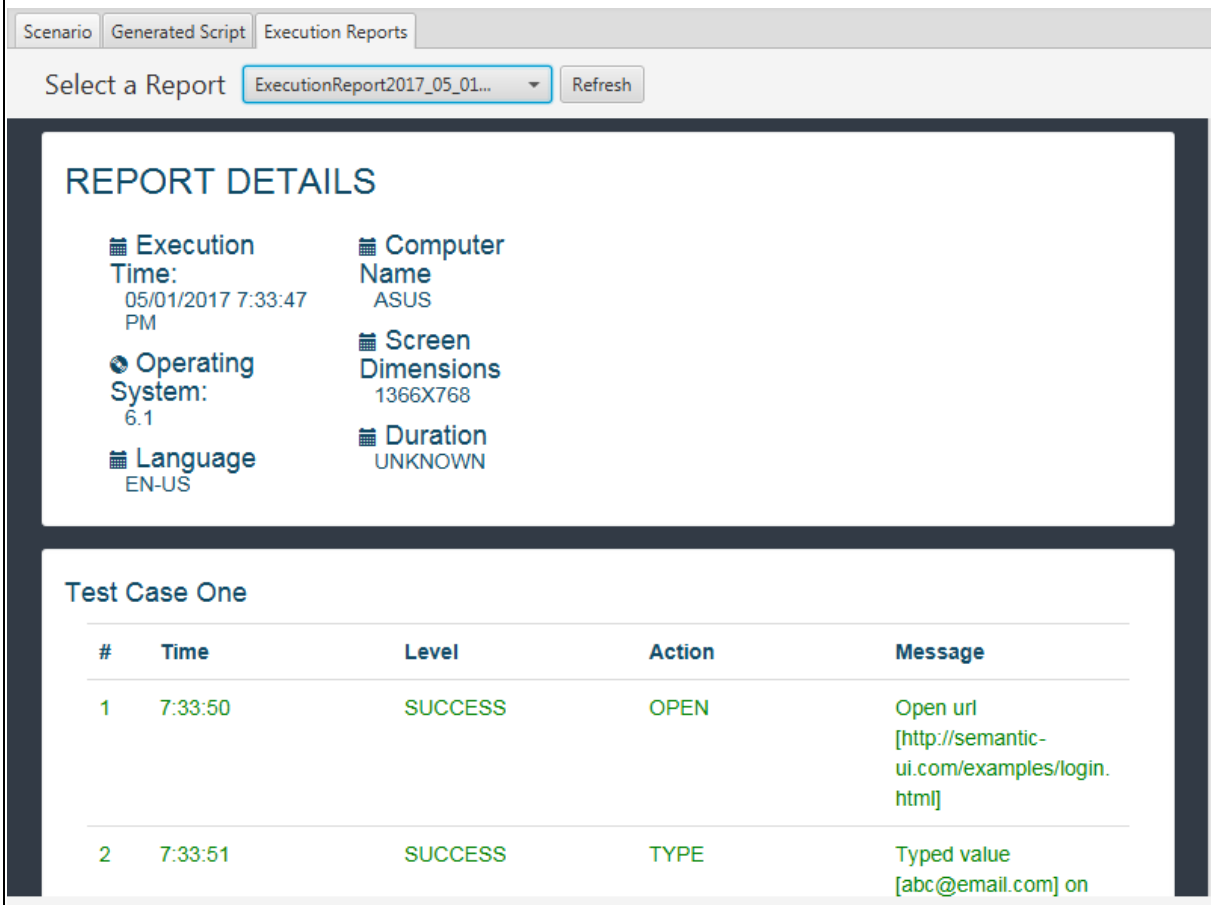




C.4 View Reports

sa. Go to the Execution Reports tab to view reports for selected test scenario.

b. Select a report from drop down. Refresh will update the report list to latest.



REFERENCES

[1] Bhattacharyya, P., 2012. Natural Language Processing [Accessed 22 Oct. 2020].

[2]. Burnstein, I., 2003. Practical software testing. N. Y. Springer [Accessed 09 Sep. 2020].

[3] Farrús, M., Costa-jussà, M., Mariño, J., Fonollosa, J., 2012. STUDY AND COMPARISON OF RULE-BASED AND STATISTICAL CATALAN-SPANISH MACHINE TRANSLATION SYSTEMS [Accessed 23 Feb. 2021].

[4] Fecko, M., Lott, C., 2002. Lessons learned from automating tests for operations support system. Software—Practice Exp. Arch [Accessed 20 Dec. 2020].

[5] Fewster, M., Graham, D., 1999. Software test automation.

[6] Gupta, S., Kumar, S., Saxena, C., 2015. Review Paper on Comparison of Automation Testing Tools Selenium and QTP. MIT Int. J. Comput. Sci. Inf. Technol.

[7] Hirschberg, J., Manning, C., 2015. Advances in natural language processing.

[8] Javvaji, N., Sathiyaseelan, A., Selvan, U.M., 2011. Data Driven Automation Testing of Web Application using Selenium [Accessed 22 Dec. 2020].

[9] Karlin, I., 2012. An Evaluation of NLP Toolkits for Information Quality Assessment. Master Linnaeus Univ [Accessed 15 Sep. 2020].

[10] Kaur, H., Gupta, D., 2013. Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and TestComplete. Int. J. Eng. Res. Appl [Accessed 20 Dec. 2020].

[11] Kelly, M., 2003. Choosing a test automation framework [Accessed 08 Feb. 2021].

[12] Language.worldofcomputing.net, 2016.

[13] Laukkanen, P., 2006. Data-Driven and Keyword-Driven Test Automation Frameworks. Data-Driven Keyword-Driven Test Autom. Framew [Accessed 22 Oct. 2020].

[14] Leotta, M., Clerissi, D., Ricca, F., Tonella, P., 2013. Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. 20th Work. Conf. Reverse Eng. WCRE.

[15] Manning, C., Raghavan, P., Schü tze, H., 2008. Introduction to information retrieval. N. Y. Camb. Univ. Press [Accessed 20 Dec. 2020].

[16] Monier, M., El-mahdy, M., 2015. Evaluation of automated web testing tools. Int. J. Comput. Appl. Technol. Res [Accessed 08 Feb. 2021].

[17] Nagle, C., 2000. Design for Test Automation. URL http://safsdev.sourceforge.net/DataDrivenTestAutomationFrameworks.htm

[18] Opennlp.apache.org, 2016. Apache OpenNLP - Welcome to Apache OpenNLP.

[19] Pettichord, B., 2003. Deconstructing GUI Test Automation. URL https://www.prismnet.com/~wazmo/papers/deconstructing_gui_test_automation.pdf [Accessed 22 Dec. 2020].

[20] Pradhan, L., 2011. User Interface Test Automation and its Challenges in an Industrial Scenario [Accessed 22 Oct. 2020].

[21] Rankin, C, 2002. The Software Testing Automation Framework. IBM Syst. J. p.126 [Accessed 15 Jan. 2021].

[22] Sangave, V., Nandedkar, V., 2014. A review on Automating Test Automation. Int. J. Adv. Res. Comput. Sci. Manag. Stud. pp.79-86.

[23] Seleniumhq.org, 2016. Selenium Documentation — Selenium Documentation [Accessed 25 Dec. 2020].

[24] Stanfordnlp.github.io, 2016. Stanford CoreNLP – a suite of core NLP tools | Stanford CoreNLP [Accessed 27 Sep. 2020].

[25] Torkar, R., 2006. Towards automated software Testing Techniques, classifications and frame works. Blekinge Inst. Technol. Swed.

[26] Zambelich, K., 1998. Totally data-driven automated testing. URL http://www.oio.de/public/softwaretest/Totally-Data-Driven-Automated-Testing.pdf [Accessed 14 Jan. 2021].