LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Implementing test automation with Selenium WebDriver

Case study: MeetingPackage.com

LAHTI UNIVERSITY OF APPLIED SCIENCES
Degree programme in Business Information Technology
Bachelor's Thesis
Autumn 2018
Quan Dao Minh

Lahti University of Applied Sciences
Degree Programme in Business Information Technology

Dao, Quan:

Title: Implementing test automation with Selenium WebDriver

Case study: MeetingPackage.com

Bachelor's Thesis in Business Information Technology

61 pages, 3 pages of appendices

Autumn 2018

ABSTRACT

The objective of this research is to identify the impact of automation testing in terms of time efficiency in software development, particularly, Agile software development with Scrum methodology. To achieve that, the thesis employs the Research Design methodology with an abductive approach. In addition, data collection in this thesis is conducted by observing the artefact's behavior and results. Lastly, the thesis introduces comparison scenarios with descriptive data analysis method to assess the impact of the artefact to the case company.

In detail, the thesis discusses the current issue of the case company, then, presents the relevant theoretical framework to back up the reasonings of the selected solution. Later, the thesis introduces the implementation of the artefact Selenium Framework using Java programming language. The implementation includes planning, gathering system requirements, developing and collecting results, followed by observing data and analyzing the test results. After data evaluation, the thesis concludes about the artefact's performance in reducing time consumption for the testing effort. Finally, the finding confirms the artefact's effectiveness to the case company with statistical evidence.

Additionally, the research provides an implementation guideline for future reference with the topic of Selenium Framework. However, any usage must take into consideration the research's limitations and validity prior to any adaptation.

Keywords: Agile Software Development Methodology, Scrum, Selenium, Java, Maven, Git, POM.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

API –	Application Programming Interface
CSS –	Cascading Style Sheets
DOM –	Document Object Model
GUI –	Graphical User Interface
HTML –	Hyper Text Markup Language
HTTP –	Hypertext Transfer Protocol
IDE –	Integrated Development Environment
IE –	Internet Explorer
POM –	Page Object Model
QA –	Quality Assurance
VCS –	Version Control System
SaaS –	Software as a Service
SWOT –	Strengths, Weaknesses, Opportunities, Threats
UAT –	User Acceptance Test
UI –	User Interface
XML –	Extensive Markup Language
XPATH –	XML Path Language

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Research Background

Web development has been exploding for the past couple of years. As technology becomes more and more approachable, there is a growing vast number of users who are surfing on the Internet daily. In fact, the number tripled between 2007 and 2017 as shown in Figure 1 below:

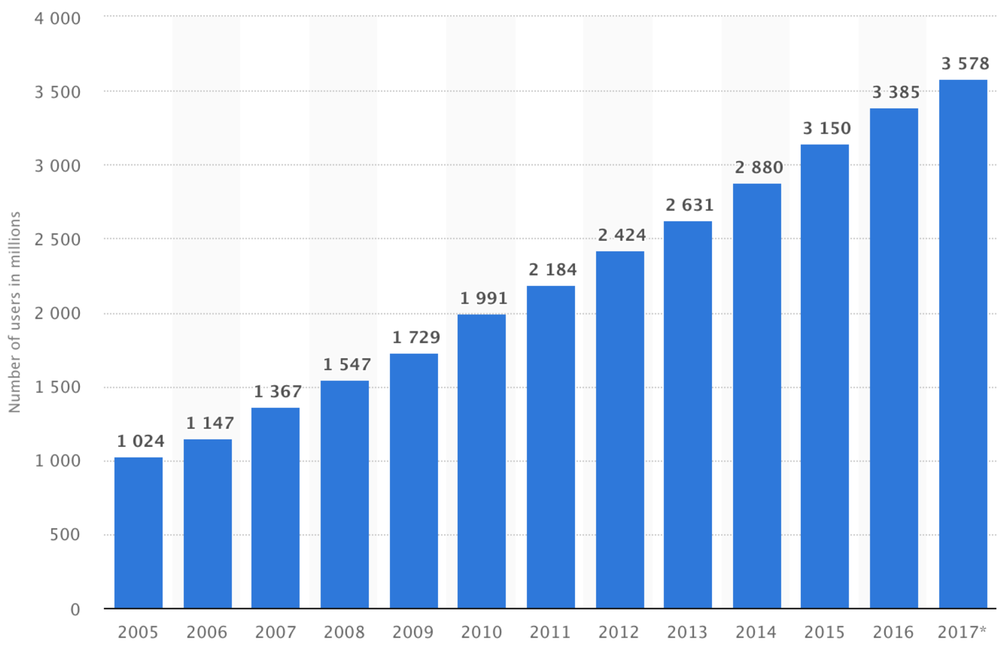


Figure 1 Number of internet users worldwide from 2005 to 2017 (in millions) (Statista 2018)

However, researches have shown that users' interest in a website drops drastically after 3-4 seconds for various reasons, for instance: bad design, too complicated or lack of information (Spilka 2016). Apparently, it is already a challenge to set up a website to represent the business. However, a slow non-responsive website is not of any favor for the business itself. In fact, sometimes, having a problematic website makes the business or the brand appear to be sketchy and not trustworthy from its potential customers (Spilka 2016).

Furthermore, for the purpose of fast constant growth, changes are required to be made daily or weekly. Therefore, it is imperative to ensure there is no development regression or down-time in the live product. As a result, the development team faces an obstacle: how to maintain a well-functioning web application with constant changes and releases in a short period of time?

To achieve this, quality assurance plays a huge role in the development cycle. By ensuring a fast, reliable and efficient QA process, the development team can speed up the feature shipment time. Traditionally, manual testing is accounted for a great part of the QA process. However, it is noticeable that automation testing is slowly replacing manual testing, due to its ability of cross-browser testing with deep penetration based on the predefined test suite. Compared to manual testing where the cost of time, money and effort remain a tremendous burden, automation testing appears to be a better solution for regression and time efficiency. (Nguyen, Hackett & Whitlock 2006, 28.)

1.2 Research Objectives and Questions

Automation testing involves manipulating the machine to perform certain tasks written by developers in a programming language. However, it is time efficient for developers to utilize certain testing framework without the need of developing all the modules, classes and methods from scratch. For example, Selenium Framework is a powerful tool to automate different browsers using web drivers in order to test web functionalities or mimic users' behaviors (Selenium 2018). This thesis demonstrates why and how Selenium Framework is applied in web development QA by finding the answer to the following question:

- Research Question: **How does Selenium Framework reduce time consumption in Quality Assurance at MeetingPackage.com?**

The objective of this research is to measure the time efficiency Selenium Framework provides when being implemented in the test suite. Later on, the data is compared to the QA time consumption before the test suite in the case company. Finally, the research concludes about the benefit of using Selenium Framework for test automation based on the statistical result. Furthermore, the thesis aims to provide a complete guideline about Selenium Framework implementation with the selected programming language (Java) for automation testing of web application.

2 RESEARCH DESIGN

2.1 Research Approach

The second chapter explains the reason behind the research approach and method selection for this thesis. According to Saunders (2012, 144-145), there are three approaches in research design:

- Deductive approach: researcher based on an existing theory to test some hypotheses and come up with a conclusion.
- Inductive approach: as opposed to deductive, research starts with gathering data or legitimate benchmarks, study the patterns and similarities or dissimilarities, then develop theories and concepts.
- Abductive approach: the purpose of abductive approach is to generate or correct an existing idea with partial information so that further data can be evaluated in further research.

Figure 2 illustrates the difference between each approach:

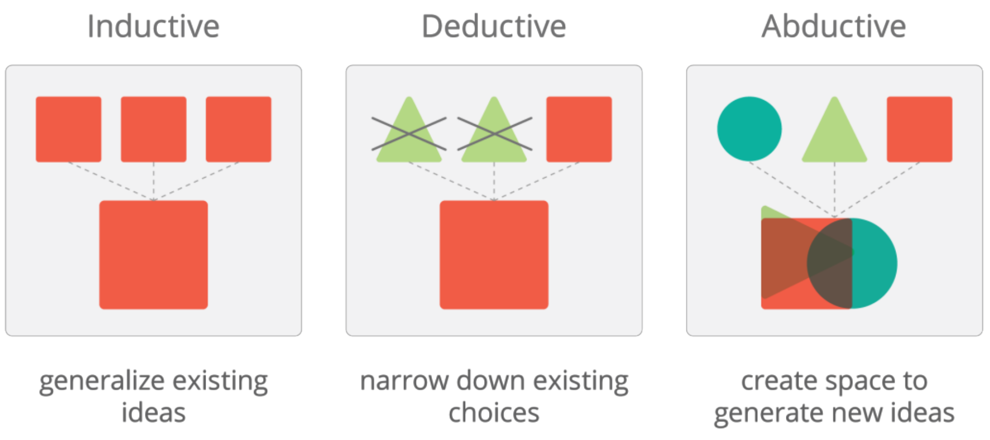

Inductive
Deductive
Abductive
generalize existing ideas
narrow down existing choices
create space to generate new ideas

Figure 2 Three research approaches (AssignmentPoint 2018)

This thesis suggests an explanation for the aforementioned problem (the need for test automation), however, it is not certain that it is the only factor contributing to the problem. From there, the author conducts research to identify the true reason behind the problem with an open-minded to new hypotheses or ideas (Shuttleworth 2008). With all that factors taken into consideration, the thesis employs an abductive research approach.

2.2 Research Method

The thesis author chooses Design Science to be the thesis research method due to the following reasons. First of all, it is a research method that involves a problem-solving approach, posing the need for change toward a better future (Barab & Squire 2014.). Secondly, Design Science consists of a research practice to create an artefact to resolve the declared issue, analyzing the result and demonstrate the finding to the audience (Peffers, Tuunanen, Rothenberger & Chatterjee 2008, 6). As Figure 3 below shows, the process starts with an awareness of the problem, then the researcher comes up with a suggestion to develop the artefact. After that, a thorough evaluation is conducted to bring up a conclusion of the results.

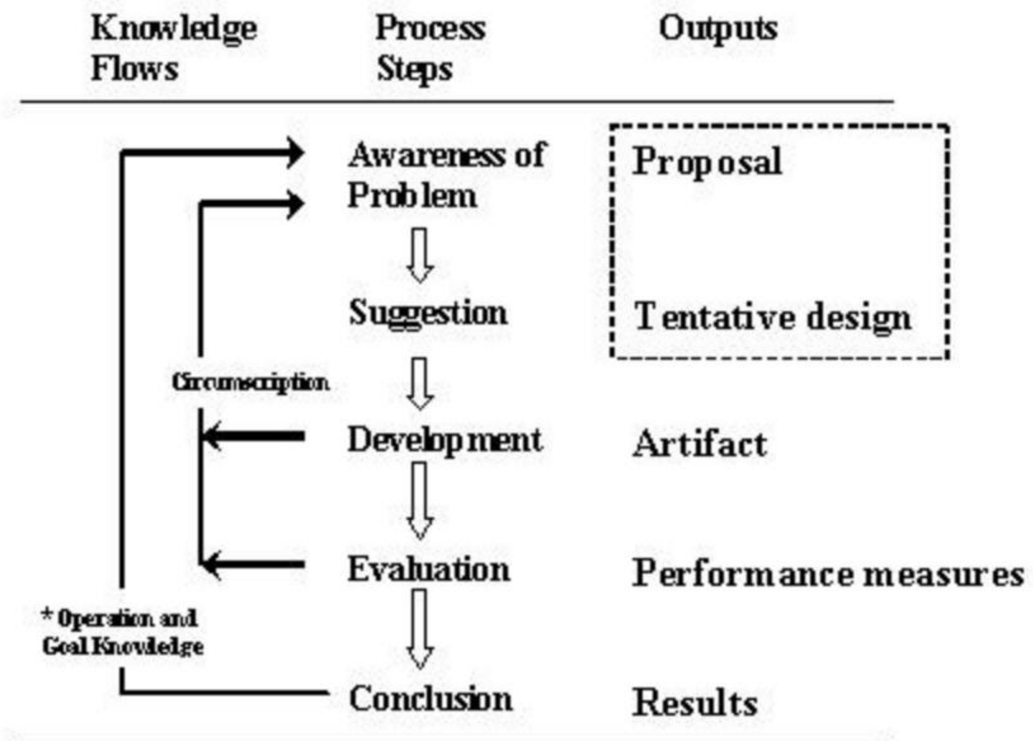


Figure 3 Reasoning in the Design Cycle (Vaishnavi & Kuechler 2012)

Practically, this thesis follows closely these six steps in Design Science method (Geerts 2011):

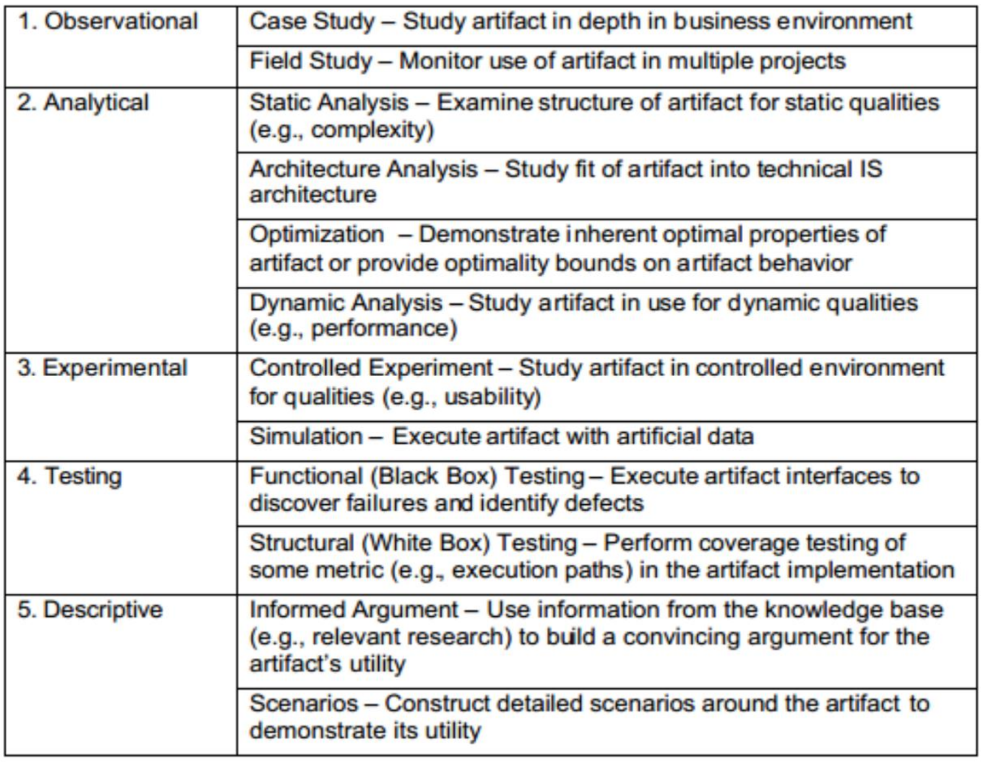
- Problem identification and motivation: the main question is to recognize and identify the problem, its relevance, and the current situation (Geerts 2011).
- Define the solution goal: finding the answer for "How to solve the problem" by suggesting a specific objective to achieve. This requires the knowledge of the feasibility and the possibility of the solution as well as the theories and method behind that (Geerts 2011).
- Design and development: build the artefact that resolves the aforementioned issue. The researcher needs to combine the required knowledge to create the artefact (Geerts 2011).
- Demonstration: a proof of how the artefact works and solve the problem needs to be generated in this phase (Geerts 2011).
- Evaluation: based on the demonstration, the researcher observes and measures its performance by comparing the actual results with the set objectives (Geerts 2011).
- Communication: researcher must present a detailed conclusion of the effectiveness of the artefact, its scope, and limitation (Geerts 2011).

2.3 Data Collection and Data Analysis

Data collection plays an important role in any research. As imperative as selecting the research design and approach, the choice of particular data collection methods determines the result of the research. In fact, in order to have a convincing data collection, the researcher needs to identify the proper objective. Once it is defined, the format of the data collection and the way the data is handled is determined.

According to Hevner (2004), there are five different methods to evaluate the data in Design Science (Table 1). For the purpose of this thesis, an observational method to collect data and descriptive method with scenarios to analyze its performance are utilized. Throughout the development progress, the author keeps track of field notes, memos, behavior records or activates during the observations. Later, the data is analyzed under a descriptive scenario and compared to the previous behavior before the artefact is developed to measure the artefact's effectiveness and limitation.

Table 1 Data Evaluation Methods (Herver 2004)



1. Observational	Case Study – Study artifact in depth in business environment
	Field Study – Monitor use of artifact in multiple projects
2. Analytical	Static Analysis – Examine structure of artifact for static qualities (e.g., complexity)
	Architecture Analysis – Study fit of artifact into technical IS architecture
	Optimization – Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
	Dynamic Analysis – Study artifact in use for dynamic qualities (e.g., performance)
3. Experimental	Controlled Experiment – Study artifact in controlled environment for qualities (e.g., usability)
	Simulation – Execute artifact with artificial data
4. Testing	Functional (Black Box) Testing – Execute artifact interfaces to discover failures and identify defects
	Structural (White Box) Testing – Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Descriptive	Informed Argument – Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility
	Scenarios – Construct detailed scenarios around the artifact to demonstrate its utility

2.4 Thesis structure

This thesis contains five sections which then divided into seven chapters. Figure 4 illustrates the overview structure of the thesis:

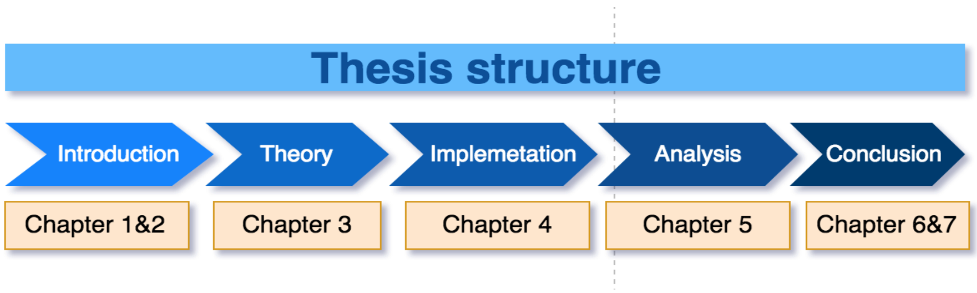


Figure 4 Thesis structure

The five main sections are the following: introduction, theoretical framework, artefact implementation, analysis, and conclusion. The following summarizes the content of each of the five sections:

Introduction: This section aims to deliver a brief understanding of the thesis, its objective and research methods used in research. The research question and data collection methods are also introduced in this section.

Theory: This section provides all the necessary theoretical framework for the thesis. The key concepts which are defined include Agile software development methods, testing principles, automation testing with Selenium WebDriver, Git and GitHub, Java and Maven.

Artefact implementation: the third part of the thesis is the development of the artefact. In this section, the thesis author demonstrates with examples how he developed and maintained the artefact.

Analysis of the test results: in this part, the author gathers the data and calculates the cost and benefit of the artefact to find out whether or not the artefact provides any benefits to the case company.

Conclusion: The last section presents the comparison of data before and after the artefact together with the author’s conclusion of the artefact’s performance. Lastly, the disadvantages and limitations of the artefact are also pinpointed.

3 THEORETICAL FRAMEWORK

3.1 Testing in Agile software development

3.1.1 Agile Software Development Overview

The most well-known and traditional method when developing a software is the waterfall methodology – a collection of separate phases, each phase follows each other from the planning to the deployment. (Ransome & Misra 2014, 50). However, it becomes clear that this method faces an issue – there is no room for errors whether they are big or small.

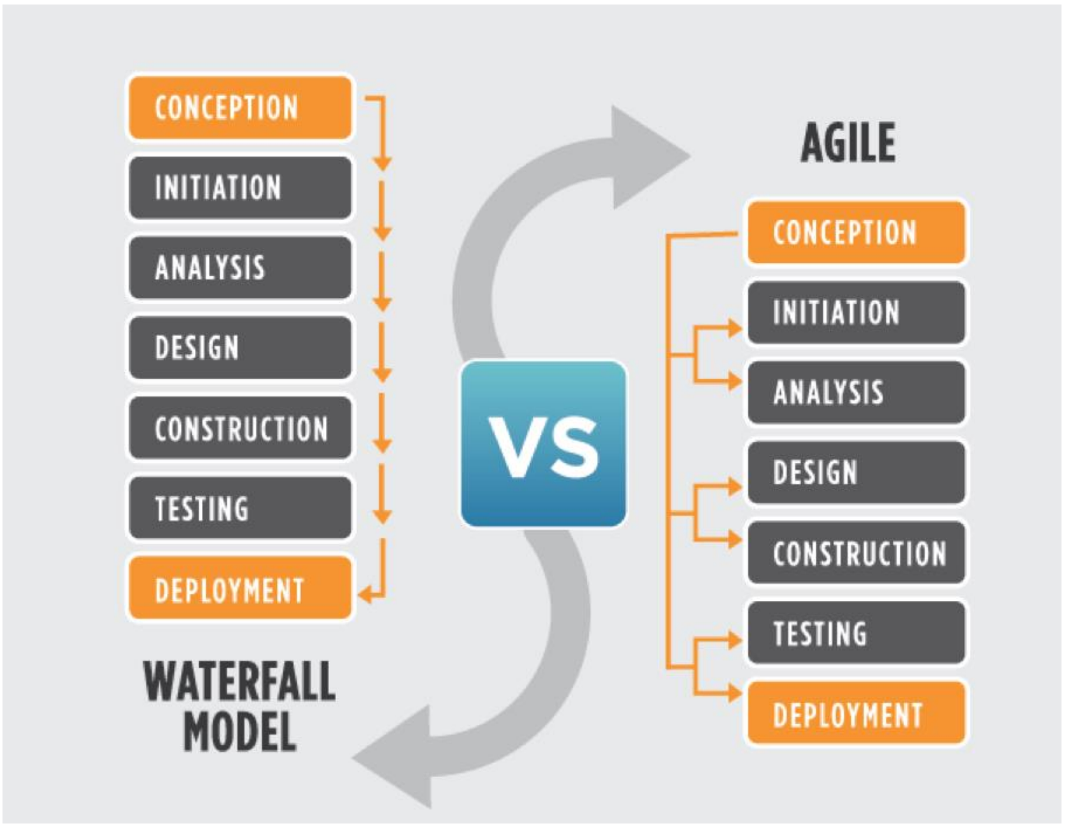


Figure 5 Waterfall VS Agile (Lotz 2013)

Figure 5 above shows the reality of waterfall methodology. With a circumstance that every stage starts after the previous one finished, the development effort becomes tremendous with hidden failures and even poses a failed status (Lotz 2013). For example, even when the product concept is agreed by both parties (development team and client), there is no guarantee that the final product which will be presented only at the end of the development is what the clients want. As the testing only happens at the late stage of the development, developers and testers are struggling with errors and bugs, not to mention if the client suddenly wants to change any requirements of the product. In other words, most of the development efforts could be in vain, and the loss of time, money and resources is unmeasurable (Lotz 2013).

Agile methodology comes forth as an ideal solution for the aforementioned issue. Developed in the 1970s and 1980s, the Agile methodology was widely welcomed and highly adopted by development teams in the effort of minimizing project errors, faster development time and reducing resources (Ekas & Will 2013). Agile method prides itself on its ability to raise communication between the team and clients, a collaboration between team members and improvement on code quality (Ekas & Will 2013). Each sprint (a time-boxed duration) lasts from one to four weeks with a specific plan and objective. Were there any unfinished tasks, the project manager would re-prioritize and design a new plan for the next sprint. In 2001, several software professionals came up with The Agile Manifesto with four core values (Ashmore & Runyan 2014, 2-9):

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

This manifesto set the four concrete principles of software development nowadays. In Agile, testing is involved at any point of the development process, even at the very start. Therefore, the testing quality of the product remains managed and cost-effective (Schwaber & Beedle 2002).

3.1.2 Scrum Methodology Overview

Scrum is a guideline or a development framework to which development team follows in Agile projects. With Scrum, time-to-market is shortened with high customer satisfaction due to the fact that bugs or errors are continuously identified before the big deployment, which means the quality of the product increases. In short, Scrum serves as an Agile framework to raise the team collaboration, enhance the control of the project and constant product deliver every two or four weeks. (Ashmore & Runyan 2014, 53.). In reality, there are three factors required in a Scrum methodology (Resnick & Bjork 2011):

- Product Owner or IT Business Analyst: they are the bridge between the development team and the customers. Their role in the team is to understand the business requirements and convey that into development requirements. They ensure the developers fully acknowledge and follow the right path of product development. To the customers, they are required to inform any changes happened.
- Scrum master: Taking charge of the team communication, planning and actions. For a small team, the project manager is the Scrum master, but in a big corporation, lead developers or a separate Scrum master are assigned to be.
- Development team: a cross-functional team in which every member can be responsible for different tasks (developing, testing, deploying, etc.). They work on the product backlog and deliver the potentially shippable increment at the end of the sprint.

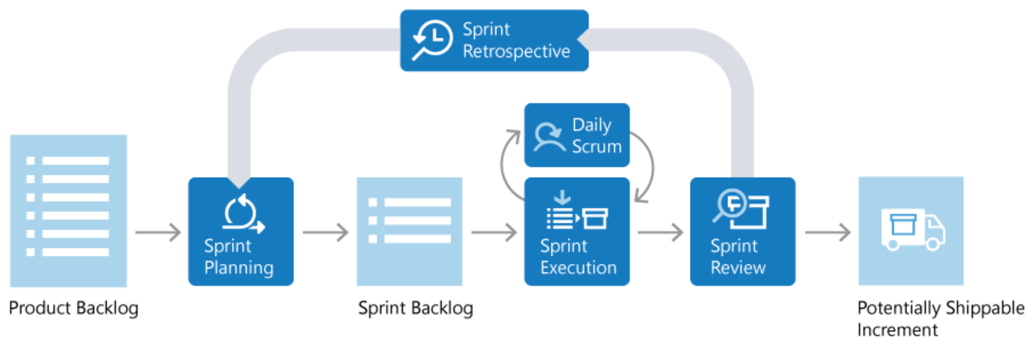


Figure 6 The Scrum Lifecycle (Boer 2017)

As can be seen from Figure 6, a sprint includes five actions:

- Sprint planning: scrum master and the team discuss and move the product backlog items into the sprint backlog based on each item priority and team’s capabilities. For each item in sprint backlog, the team divides into smaller tasks and assign to each team members (Boer 2017).
- Sprint Execution and Daily Scrum: the sprint execution starts right after the backlog is defined. Daily scrum is a practice of which team members and scrum master report their progress, their plan, and blockers daily to each other (Boer 2017).
- Sprint Review and Sprint Retrospective: after each sprint, the team demos the increment (the product of the sprint) as well as assess on remaining tasks, room for improvement (Boer 2017).

Quality Assurance in Scrum or in Agile, in general, is not a particular phase (Larman 2004, 113). The testing effort starts from the very beginning of the development, in fact, in each sprint there is always a testing task. Thanks to this, as the project expands and gets complicated, the team can confidently develop knowing that bugs and errors of the product are under control (Larman 2004, 113). With automation testing, the testing effort may seem to be quite large at first (building the test suite) but it benefits later on against regression.

3.2 Software Testing Fundamentals

The aim of testing is to find defects or errors in the product under certain condition and environment (Myers 1979). In software testing, there is an array of fundamental principles to follow. These principles account for the make-or-break of any development, especially in Agile software development practice. However, prior to the planning, the company needs to identify the test policy (regulations for testing) and the test strategy (overall testing approach). These rules and regulations are the backbones of the project testing. (Graham, Veenendaal, Evans & Black 2014.)

First and foremost, it is unquestionable that developers or testers need to do the research and design a test plan. The plan includes the test objective and the scope of the test. In reality, every project has its own scope to which the test plan needs to follow strictly in order to save time and resources. At this point, the project manager or test lead of the team determine the necessary hardware and software as well as human resource. In addition to that, risk assessment needs to be taken into consideration to avoid any unexpecting factors or unwanted difficulties. In the next step, the team decides on which testing approach is suitable for the project, what is the exit condition of the testing effort and the testing policy. (Grahrai 2017.) However, there is a need for quality control in case the plan follows an unexpected route. For example, if the testing effort requires more time than expected, testers need to report it to the project manager in time to modify the initial test plan. The project manager or test lead assesses the importance of the issue and makes the necessary changes and reports it to the client. These activities require a constant update after each sprint so that the team and the client keep track of the progress. Needless to say, to have a successful test plan, the test coverage is required to be followed at any time of the project. (Graham, Veenendaal, Evans & Black 2014.)

Secondly, followed the test plan is test analysis and design phase. With thorough observation of the project’s requirements and conditions, testers determine their test conditions and scenarios. For example, if users input string value to an integer field, there should be a warning from the software to indicate the problem. These problems seem to be small and harmless but in fact, they can do some serious severity to the software. Most of the time, they require a deeper knowledge and experience from the testers, as they don’t usually appear in the project’s requirements. Every requirement needs to be as specific as possible so that there is no uncertainty or confusion between the team and the clients. After a thorough analysis, testers move onto the design phase where they deliver the testing environment, database, licenses, hardware, and software, etc.

The third phase of the testing development is execution. In this phase, testers conduct detailed test scenarios and test cases. Test scenarios include all the test cases that are related to a function or a requirement of the project. Test cases define the expected behavior after a certain input or changes (Bath & McKay 2008). For example:

Test condition: Users input the wrong password five times or more, a warning should appear.

- Test Scenario #1: Users input the wrong password five times or more, a warning should appear.
  - Test case #1: Users input the wrong password five times, a warning should appear.
  - Test case #2: Users input the wrong password six times, a warning should appear
- Test Scenario #2: Users input wrong password up to four times, a warning should not appear.
  - Test case #1: Users input the wrong password three times, a warning should not appear.
  - Test case #2: Users input the wrong password four times, a warning should not appear.

All the test cases are collected and organized into a test suite. Then, the test suite is executed in a testing environment in the execution phase. The testing environment needs to strictly imitate the production environment so that it can mimic the actual behaviors once the software is deployed. The results by then are verified and collected by testers to identify the bugs or errors. A test with bugs makes the test case failed, but were there any system failure or shortage, the test case would return an error due to its inability to perform the test. Therefore, testers are required to have a proper explanation of all the bugs and errors so that developers can understand and modify their codes. (Graham, Veenendaal, Evans & Black 2014.) However, it is imperative to ensure the reliability of the test suite. The testing effort becomes redundant if the test suite misbehaves and returns false results.

The last principles or software testing is to gather and evaluate the test report. In this phase, the project manager or test lead defines whether or not the testing effort is enough, which means the exit condition is met. Generally, they base their decision on the test report and the testers' documents (what has been tested, what hasn't been tested, what is the known issues, etc.). The test report shows the percentage of successful test cases, failed test cases, and errors. These statistics are recorded after every test effort has been made so that the team can keep track of their performance and the status of the project. (Graham, Veenendaal, Evans & Black 2014.)

3.3 Automated Testing

3.3.1 Definition of Automation Testing

Manual testing refers to the concept of performing test cases using human interaction (Itkonen, Mäntylä & Lassenius 2009). In contrast, by utilizing machine or software, developers can execute a certain amount of test cases and compare the results with the expected outcome (Dustin, Garrett & Gauf 2009). In other words, test automation includes a collection of the test script to be executed without human obstruction (Henry 2008). There are 3 levels of automated testing according to Figure 7:

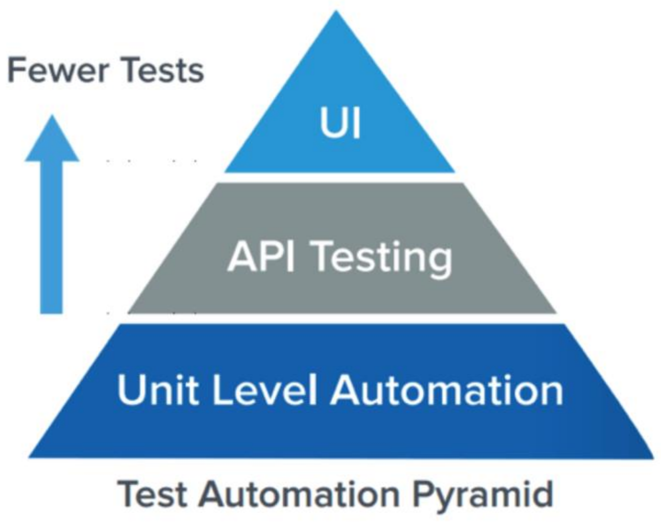


Figure 7 Test Automation Pyramid (Smartbear 2018)

- Level 1 – Unit test: A unit test is considered as a module test due to the fact that it tests a singular component in the application (Bentley 2004). xUnit (JUnit or NUnit) supports testers by making sure there is no error in one section of the source code, which means with a certain input, the function should return the expected outcome. In agile software development, developers can build unit tests for the functionality before the actual codes are written (Henry 2008).
- Level 2 – API test: API (Application Programming Interface – a set of functions that returns a collection of values after being called or interacted with) testing refers to the concept of directly testing APIs to determine its reliability, performance, security and the most important of all – functionality (Crispin & Gregory 2008). Nowadays, in agile software development, API testing helps developers and testers in keeping up with short sprint and fast pace of development (Henry 2008).
- Level 3 – GUI test: GUI (Graphical User Interface) testing is the lowest level of automation testing. As long as the application has a GUI, testers can control the machine to mimic any repetitive tasks which can be performed by end-user (manual testing). GUI test stresses more on the end-user side, as it can detect not just failed functionality but also false layout or missing/hidden components. Such problems cannot be detected with Unit or API test. However, as small changes in GUI can make the test case fail, it is imperative to record and have a fallback when developing GUI test cases (Henry 2008).

3.3.2 When to perform Automation Testing

It is a good practice to evaluate and plan the testing method before any sprint or development. Test automation is a promising candidate when it comes to regression testing (re-test the preexisted product and functions that are being changed or developed with new ones). Regression testing helps tremendously to ensure the newly introduced version or changes don't affect the old ones. In addition, any repetitive tests which are not likely to change throughout the development sprint or cycle should be optimized to be automated. Generally, these tests are high data-driven, which means the same function can be entered with large different data or input. Those data can be imported from a database to the test case to run in the different environment. (Fernandes & Fonzo 2018.)

3.4 Selenium WebDriver

3.4.1 Selenium Overview

Selenium is an open source software testing framework developed in 2004 by Jason Huggins under Apache 2.0 (Selenium 2018). It serves for the purpose of mimicking the end-users' interaction on a web application by controlling the browser web driver. Selenium supports all most all of the popular web browser nowadays and the test suite can be written with many languages (Selenium 2018). Up to this point, there are four components of Selenium:

- Selenium IDE: a Firefox add-on allows testers to record and playback interaction steps on a web page with the integration of Selenium Core. It contains a context menu with a list of verification points and assertions (Selenium 2018).
- Selenium Core: a Javascript-based tool to run the test script written in Selenese (a collection of commands). It required installation on the testing server to where testers have no access (Selenium 2018).
- Selenium WebDriver:

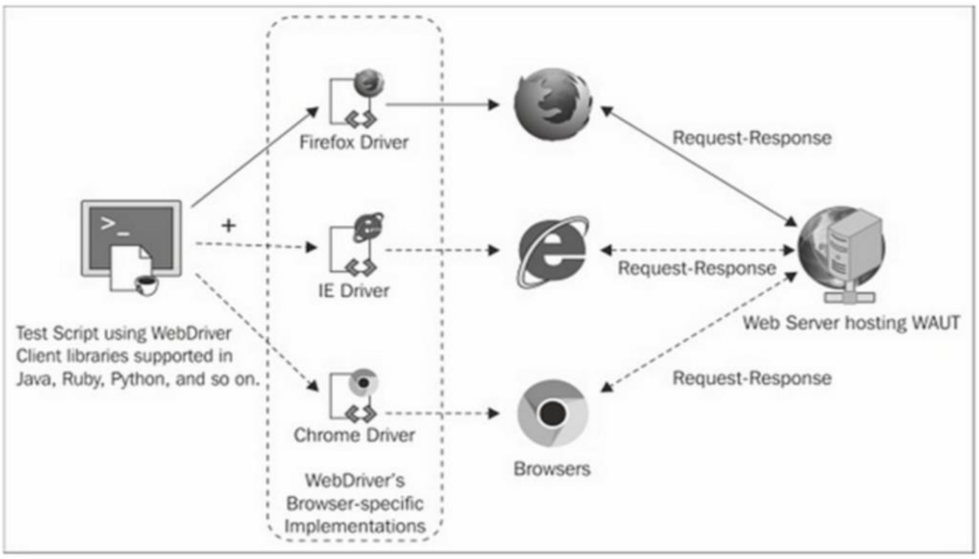

Firefox Driver
IE Driver
Chrome Driver
WebDriver's Browser-specific Implementations
Test Script using WebDriver Client libraries supported in Java, Ruby, Python, and so on.
Request-Response
Request-Response
Request-Response
Web Server hosting WAUT
Browsers

Figure 8 Interaction of WebDriver (Avasarala 2014, 13)

Is the most popular tool in test automation for web application. Figure 8 indicates that Selenium WebDrivers enables developers to compile and run an end-to-end test suite with its API. Based on different test cases, the tool passes the Selenese test script to the Selenium Core and controls the browser without the need to connect to a remote server unless it is not the native machine. As a result, this reduces the connection time, thus, fastens the test run-time (Avasarala 2014, 13).

- Selenium Grid: provides a possibility to run tests concurrently on multiple machines with various browsers and operating systems built by WebDriver (Selenium 2018).

3.4.2 Selenium WebDriver Behavior

A complete break-down of the WebDriver behavior can be found below in Figure 9:

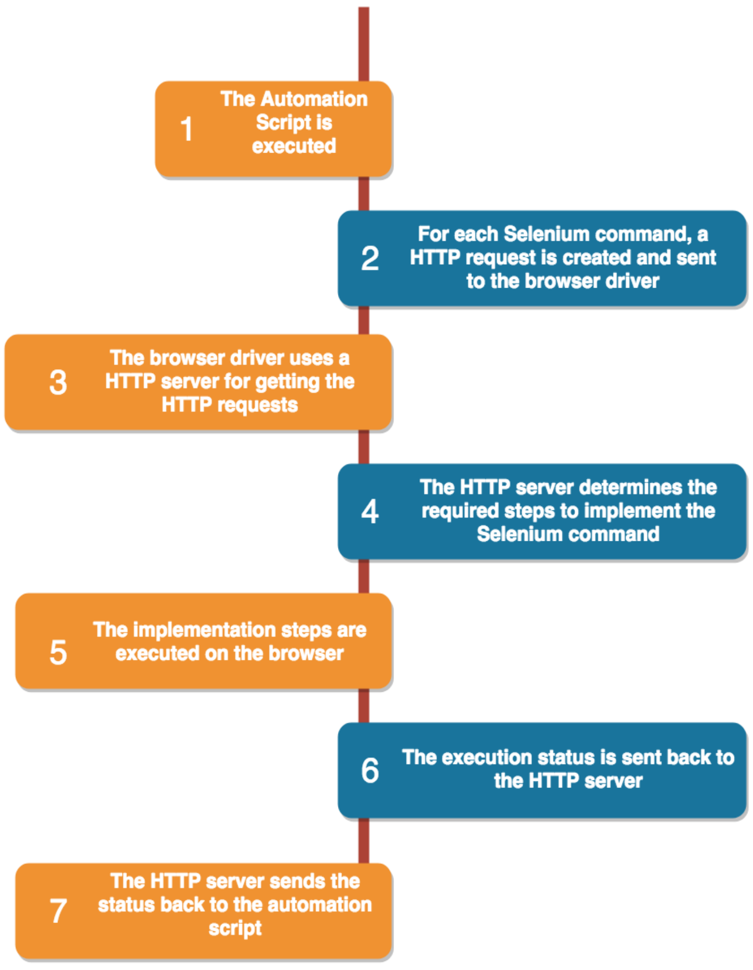

1
The Automation Script is executed
2
For each Selenium command, a HTTP request is created and sent to the browser driver
3
The browser driver uses a HTTP server for getting the HTTP requests
4
The HTTP server determines the required steps to implement the Selenium command
5
The implementation steps are executed on the browser
6
The execution status is sent back to the HTTP server
7
The HTTP server sends the status back to the automation script

Figure 9 Demonstration of Selenium WebDriver operation (Quora 2018)

Selenium WebDriver supports a wide range of client library, for example, PHP, C# (NUnit), Java (TestNG, JUnit), Perl, Ruby (RSpec) and Python (Robot, PyUnit, Unittest) (Selenium 2018). Furthermore, it also supports current web browsers: IE, Chrome, FireFox, Safari, PhantomJS (headless browser) (Selenium 2018).

3.4.3 UI Locator Overview

A webpage contains a number of HTML elements (body, input, iframe, etc.) and in Selenium WebDriver, they are named WebElements. By locating these elements, WebDriver can easily perform task and interact with (Selenium 2018). Therefore, it is imperative that the testers need to identify correctly the element involving in each test case, if not, the test case will fail due to its inability to locate (Selenium 2018). Thanks to resourceful WebDriver API, there are various techniques to locate the UI locators based on the different attributes of the web page (tagname, classname, cssSelector, Xpath, ID, Name, LinkText) – the DOM (Document Object Model) elements (Selenium 2018). After being queried in the DOM, if more than one element is matched and found, the system throws an error unless it is stored in an array (Selenium 2018). For example (Selenium 2018):

- Tag Name: By.tagName(“iframe”);

  <iframe src="..."**></iframe>**

- Class Name: By.className(“fruit”);

  <div **class="fruit"**><span>Tomato</span></div>

- ID: By.id(“bear”);

  <div **id="bear"**>...</div>

- Name: By.name(“apple”);

  <input **name="apple"** type="text"/>

- LinkText and partial LinkText: By.linkText(“bear and fruit”); or By.partialLinkText(“fruit”);

  <a href="http://www.google.com/search?q=bear+and+fruit&ie=utf-8&oe=utf-8&aq=t">**bear and fruit**</a>

- CSS: By.cssSelector("#fruit span.red.yellow");

  <div **id="fruit"**><span **class="red"**>tomato</span><span **class="red yellow"**>bell peppers</span></div>

- XPath (describe information on an XML document, not the fastest but the most accurate locator). XPath provides the ability to search

forward and backward to identify child or parent element on the DOM.
For example: By.xpath("//input");

```
<input type="text" name="example" />
```

3.4.4 Common Methods in Selenium

The most common method in Selenium WebDriver can be found in the Table 2 below:

Table 2 Common methods and commands in Selenium (Selenium 2018)

Method	Command
init webdriver	WebDriver driver = new ChromeDriver();
open url	driver.get(“google.com”);
init webElement	WebElement button1 = driver.findElement(By.id(“button”);
click an element	driver.findElement(By.id(“button”)).click();
enter text	driver.findElement(By.id(“textbox”)).sendkey(“Hello”);
refresh the page	driver.navigate().refresh():
navigate	driver.navigate().forward(); or driver.navigate().back();

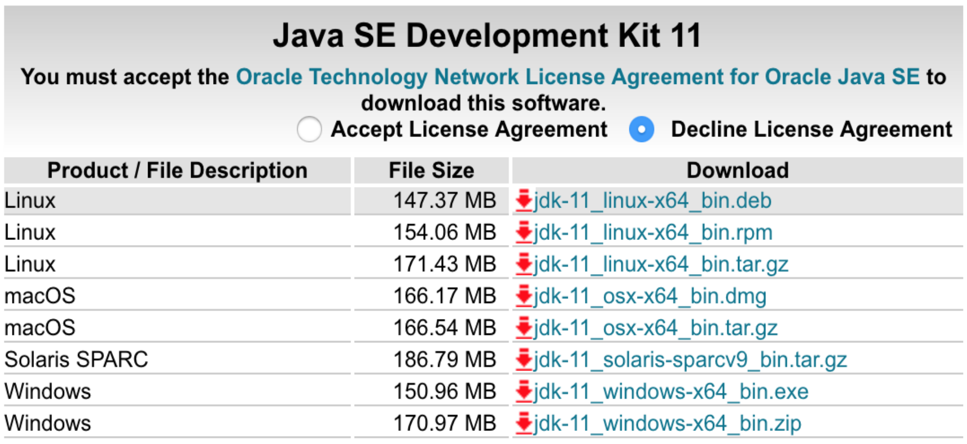
drag and drop	WebElement element = driver.findElement(By.name("origin"));
	 WebElement target = driver.findElement(By.name("new"));
	 (new Actions(driver)).dragAndDrop(element, new).perform();

get text	driver.findElement(By.id("textbox")).getText();

3.4.5 Selenium Installation System Requirements

To have a running Java environment ready to develop a Selenium test suite, there are some required system requirements (Selenium 2018):

- Java Runtime Environment (JRE) on the native computer: Java Software Development Kit (JDK) comes with JRE integration. Download and install JDK at Oracle official website for Java (Oracle 2018), users can select which operating system they are using in Figure 10:



Java SE Development Kit 11

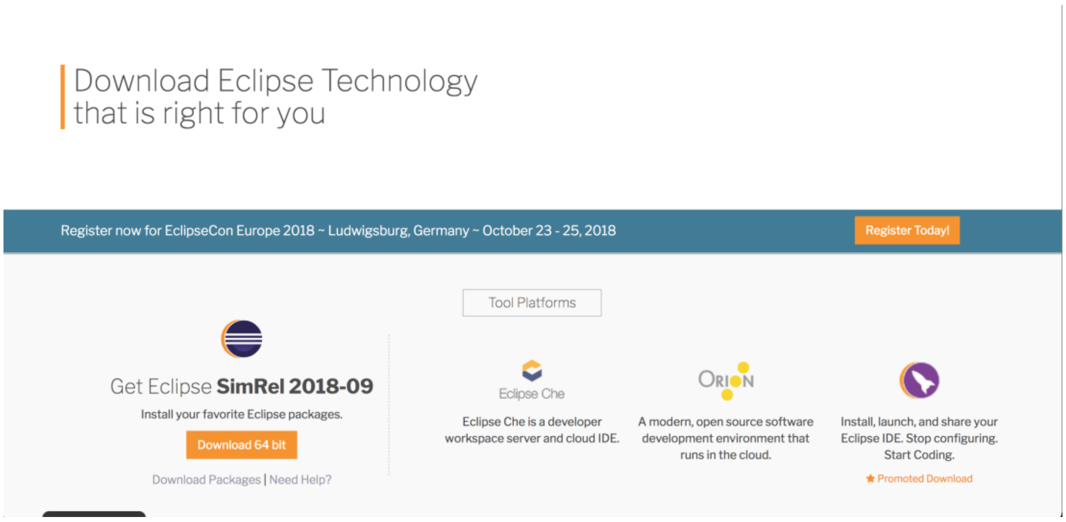
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download
Linux	147.37 MB	jdk-11_linux-x64_bin.deb
Linux	154.06 MB	jdk-11_linux-x64_bin.rpm
Linux	171.43 MB	jdk-11_linux-x64_bin.tar.gz
macOS	166.17 MB	jdk-11_osx-x64_bin.dmg
macOS	166.54 MB	jdk-11_osx-x64_bin.tar.gz
Solaris SPARC	186.79 MB	jdk-11_solaris-sparcv9_bin.tar.gz
Windows	150.96 MB	jdk-11_windows-x64_bin.exe
Windows	170.97 MB	jdk-11_windows-x64_bin.zip

Figure 10 List of supported JDK (Oracle 2018)

- Install Eclipse IDE (Integrated Development Environment) at Eclipse official website (Eclipse 2018) with Figure 11:


Download Eclipse Technology that is right for you

Register now for EclipseCon Europe 2018 – Ludwigsburg, Germany – October 23 - 25, 2018

Get Eclipse **SimRel 2018-09**

Install your favorite Eclipse packages.

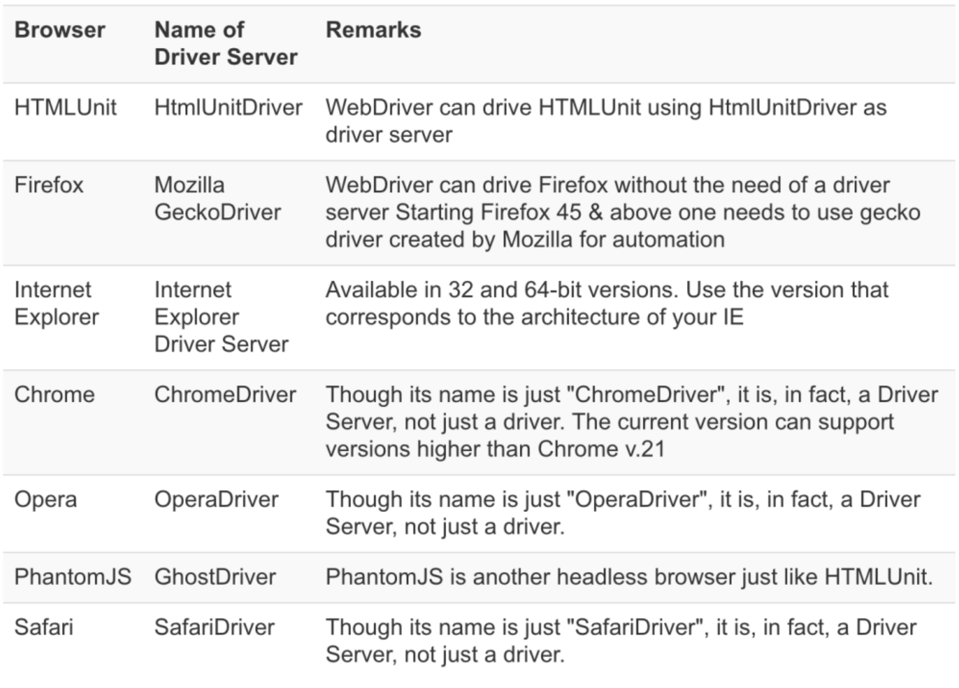
Eclipse Che is a developer workspace server and cloud IDE.

A modern, open source software development environment that runs in the cloud.

Install, launch, and share your Eclipse IDE. Stop configuring. Start Coding.

Figure 11 Eclipse Download Page (Eclipse 2018)

- Browser Driver: in order to run test scripts on a specific browser, developers need to install the corresponding browser driver and set up the path linked to the drivers' location. The following Figure 11 lists all the common browser and its corresponding drivers:



Browser	Name of Driver Server	Remarks
HTMLUnit	HtmlUnitDriver	WebDriver can drive HTMLUnit using HtmlUnitDriver as driver server
Firefox	Mozilla GeckoDriver	WebDriver can drive Firefox without the need of a driver server Starting Firefox 45 & above one needs to use gecko driver created by Mozilla for automation
Internet Explorer	Internet Explorer Driver Server	Available in 32 and 64-bit versions. Use the version that corresponds to the architecture of your IE
Chrome	ChromeDriver	Though its name is just "ChromeDriver", it is, in fact, a Driver Server, not just a driver. The current version can support versions higher than Chrome v.21
Opera	OperaDriver	Though its name is just "OperaDriver", it is, in fact, a Driver Server, not just a driver.
PhantomJS	GhostDriver	PhantomJS is another headless browser just like HTMLUnit.
Safari	SafariDriver	Though its name is just "SafariDriver", it is, in fact, a Driver Server, not just a driver.

Figure 12 Browsers and the corresponding drivers (guru99 2018)

After all the required components are installed, a complete guide to set up a Maven project can be found in Appendix 1.

3.4.6 POM (Page Object Model) Overview

According to Selenium (Selenium 2018), POM has been developed as a Design Pattern for the purpose of better maintenance and code recycling. In POM, each page is implemented as an Object-oriented class so that the tests can utilize the methods of this class as a technique to communicate with the page (Figure 13). Obviously, when there are any changes in the page, the test itself remains unchanged, only the class content needs to be updated. As all the content of the page is in one file, it is convenient to find and update new attributes at any time of the development. Moreover, the page class is independent of the tests, which mean there can be multiple types of test sharing the same page class (acceptance test, behavior-driven test, data-driven test). Lastly, methods are named realistically make them easier to read and maintain. (Selenium 2018.)

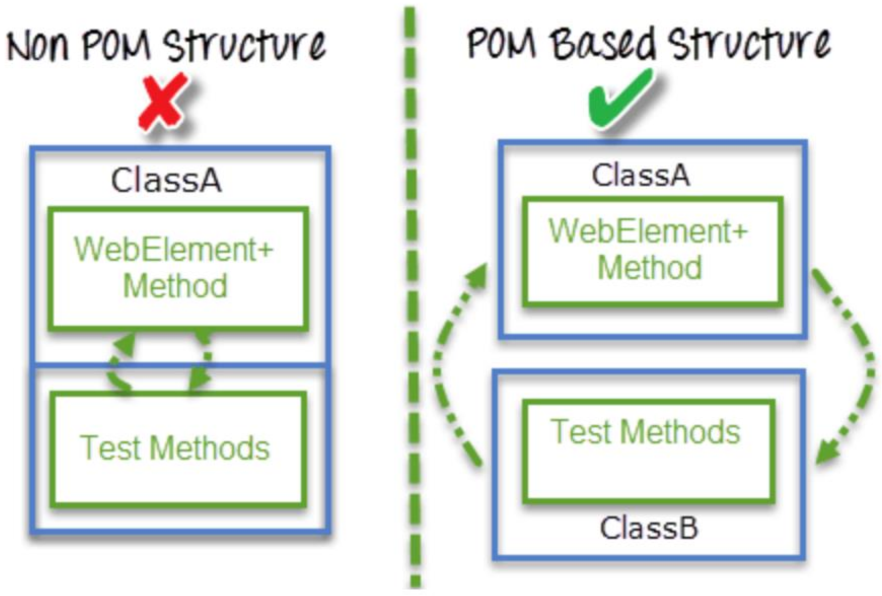


Figure 13 POM vs Non-POM comparison (Guru99 2018)

3.5 Git and GitHub

3.5.1 Git Overview

Developed in 2005 by Linus Torvalds, Git has become one of the most popular open source version control systems (VCS) up to now (Atlassian 2018). VCS is a tool that development team can track their source code status at any time of development, together with the ability to return to any particular stage of the product at a fingertip (Loeliger & McCullough 2012). In fact, any changes or modification to the source code is stored in a log as a stage so that in case something goes wrong, the team can easily revert the changes. Git falls into Distributed VCS to differentiate with Local VCS and Centralized VCS, which means each developer’s copy of the code is a well-documented repository of the full version of the software (Atlassian 2018).

Git is well-known for its various benefits including:

- Performance: it is admittedly that the power of Git surpasses any of its alternatives. Git allows each team member to work on their own versions of the source code without the fear of messing up the whole system, online or offline. Later on, when changes to the files are ready to merge, they can commit at once and push to publicly be peer-reviewed (Atlassian 2018).
- Security: unlike other VCS, the content in Git is encrypted with a hashing algorithm called SHA-1. Developers can confidently exchange content of their repository knowing that it is secured and non-malicious (Atlassian 2018).
- Flexibility: Git supports any project size from small to large and compatible with various systems and protocols (Somasundaram 2013, 15).

3.5.2 GitHub

As of 2015, with nine million developers, GitHub remained the most common VCS for open source projects (Westby 2015, 211). GitHub was born to provide hosting service for Git projects, which tremendously helps to raise network collaboration (McQuaid 2014, 224). For example, with GitHub, developers can (Brown 2016):

- Fork a repository: contribute to developing a project based off on other’s pre-existing project.
- Pull Request: submit your contribution to the project owner.
- Changelogs: keep track of all the changes to a repository.

For the purpose of the artefact development, the thesis author utilizes Git and GitHub as a means of team communication and collaboration.

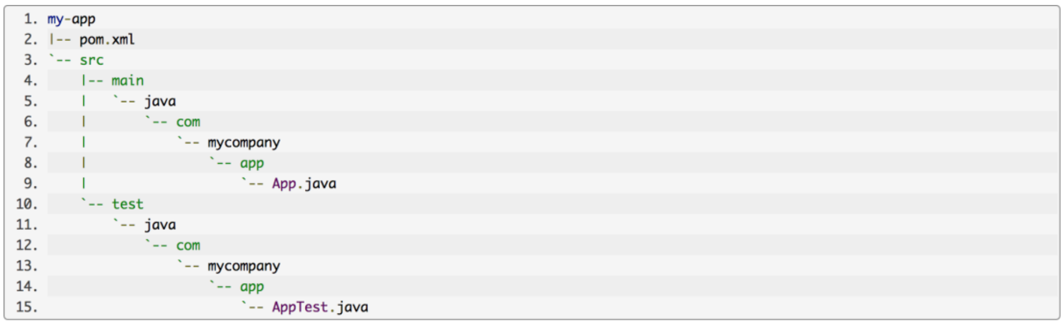
3.6 Java

3.6.1 Java as a Programming language

Served as a programming language and computing for multiple platforms, Java was developed in 1995 by Sun Microsystems. As a result of being a compiled language, Java enables developers to write code once and run everywhere (Techopedia 2018). Up to 2016, Java was the most popular programming language with 9 million developers (Wikipedia 2018), many documentations available and community supports. Java program development is in need of Java software development kit (SDK) which contains a compiler, interpreter, documentation generator (Techopedia 2018). The language itself provides excellent tools with IDE. Together with JUnit for unit testing, Java is a good candidate for test automation.

3.6.2 Maven Overview

Maven is a Java build management tool to define how the .java files are compiled to .class and much more (Maven 2018). As a result, Maven was developed to indicate a standard way to build the project by automatically downloading all the required libraries declared in the pom.xml file (pom stands for Project Object Model). Figure 14 below shows a standard project structure run by Maven.



```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Figure 14 Project Structure with Maven (Maven 2018)

The pom.xml file is a collection of all the required dependencies that the project needs in order to run such as Junit or selenium-java. It is written in XML format. Furthermore, with Maven, the file is executed as expected, meaning that the necessary for “compile”, “test”, “package” and ”clean” steps are redundant (Maven 2018). Especially in automation testing, there are a lot of plug-ins to be utilized in test reports (Surefire) and to run the test suite with Maven poses no difficulties with Maven commands (to be discussed in section 4.4)

4 ARTEFACT IMPLEMENTATION – CASE: MEETINGPACKAGE.COM

4.1 Introduction

4.1.1 Company Information

MeetingPackage.com (Figure 15) is the case company for this research. Founded in 2014, MeetingPackage.com's objective is to provide a SaaS model for meetings and events. In fact, the main product of the company is an online marketplace where venue owner and the customer can visit when there is a need to hire meeting rooms or events. Unprecedentedly, MeetingPackage.com is the pioneer in the industry who tries their best for the sake of minimizing time and money as well as increase the probability of successful meetings hire (MeetingPackage.com 2018).

For Meeting Venues (Conference centers, hotels, venues, etc.), the platform provides all the necessary information and tools in order to make the venues become available on the market as soon as possible. They offer venue creation, seasonal pricing, multi-user functionality and complete management of rooms and packages. For Customers, the platform requires little-to-none user manual with its elegant UI and fast search time. The user can easily find, book and pay for their booking within three clicks without any hidden fee (MeetingPackage.com 2018).

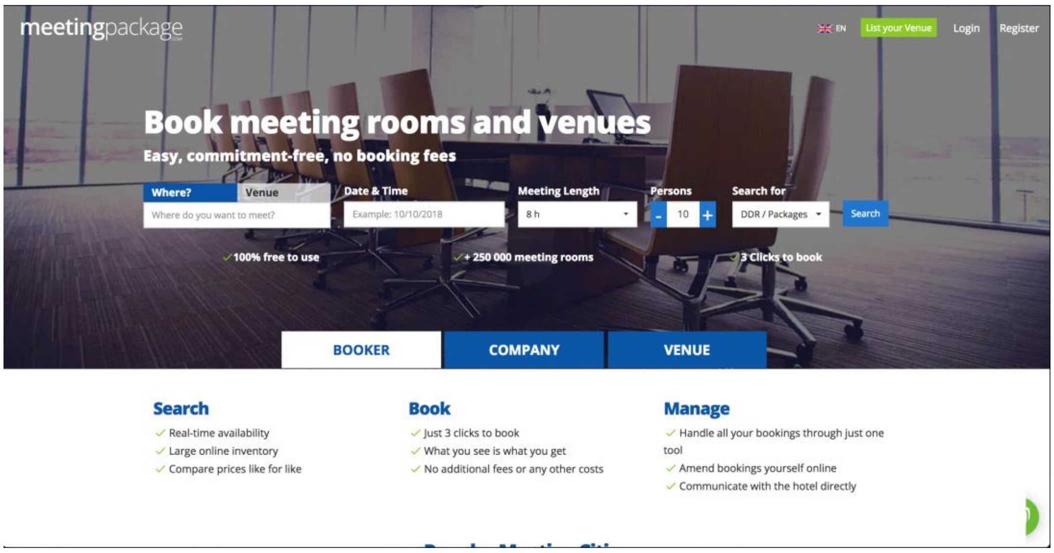

meetingpackage
EN
List your Venue
Login
Register
Book meeting rooms and venues
Easy, commitment-free, no booking fees
Where?
Venue
Where do you want to meet?
Date & Time
Example: 10/10/2018
Meeting Length
8 h
Persons
10
Search for
DDR / Packages
Search
100% free to use
+ 250 000 meeting rooms
3 Clicks to book
BOOKER
COMPANY
VENUE
Search
Real-time availability
Large online inventory
Compare prices like for like
Book
Just 3 clicks to book
What you see is what you get
No additional fees or any other costs
Manage
Handle all your bookings through just one tool
Amend bookings yourself online
Communicate with the hotel directly

Figure 15 MeetingPackage.com Home Page (MeetingPackage.com 2018)

4.1.2 Analysis of Preceding Test Process

With a huge number of functionalities, the platform provides, it is not an easy task when it comes to testing of the product. First, the thesis briefly discusses the development team and then demonstrates the testing effort the team needs to make before the artefact is developed and its related problems. Finally, a SWOT analysis is conducted to emphasize the need of the artefact to the company.

Agile software development methodology or in particular, Scrum methodology is practiced by the team with two-week increment. Before each sprint, the product owner gathers business ideas that he conveys from the stakeholders, along with any reported bugs or urgently needed changes, adds to the product backlog items. In each sprint, the team organizes a meeting to prioritize and allocate from the product backlog to the sprint backlog. After that, each developer is assigned to each story which they need to break down into smaller tasks and add to the Not started to-do list like the Figure 16 below:

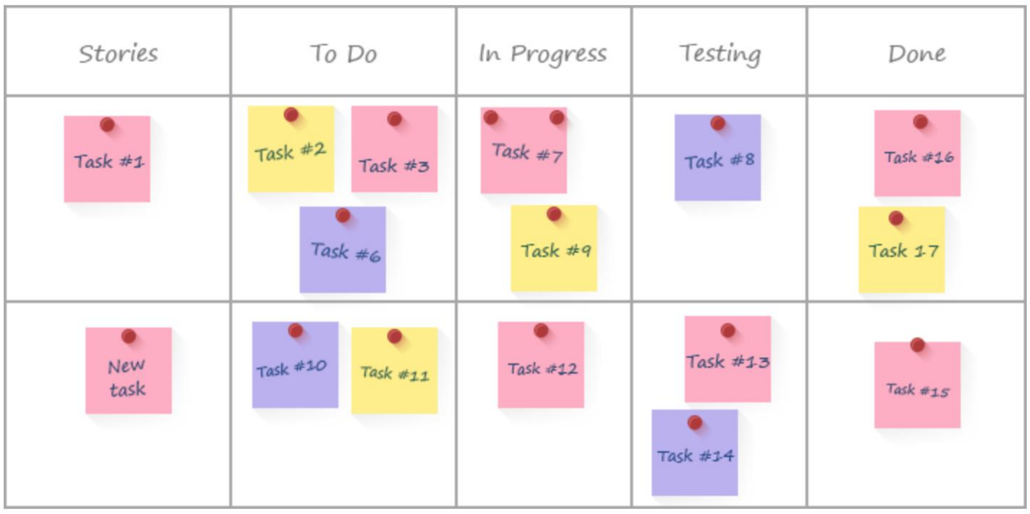


Figure 16 Working in the sprint (Gurendo 2015)

Test planning and exit condition are designed at the beginning of each sprint so that the developer and the product owner agree upon the criteria of the story complete. After the developer’s implementation, each task is moved from Not Started to In Progress and then Done. Then, the assigned developer ensures the newly developed functionality stay working and bug-free in his/her environment. After that, the assigned developer or the Product Owner perform UAT and identify any bugs or false requirements by deploying the new story to a staging environment (a product-like environment). At this point, testers are required to check on the new required story as well as performing regression testing. Were no major problems to be found, the story would be marked successful and wait to be deployed to the live production environment. Lastly, another check from the stakeholder is required to evaluate the success of the story. If there are any problems found in the meantime, developers need to fix or refactor their code and restart the process. A graph to visualize the process is presented under the following Figure 17:

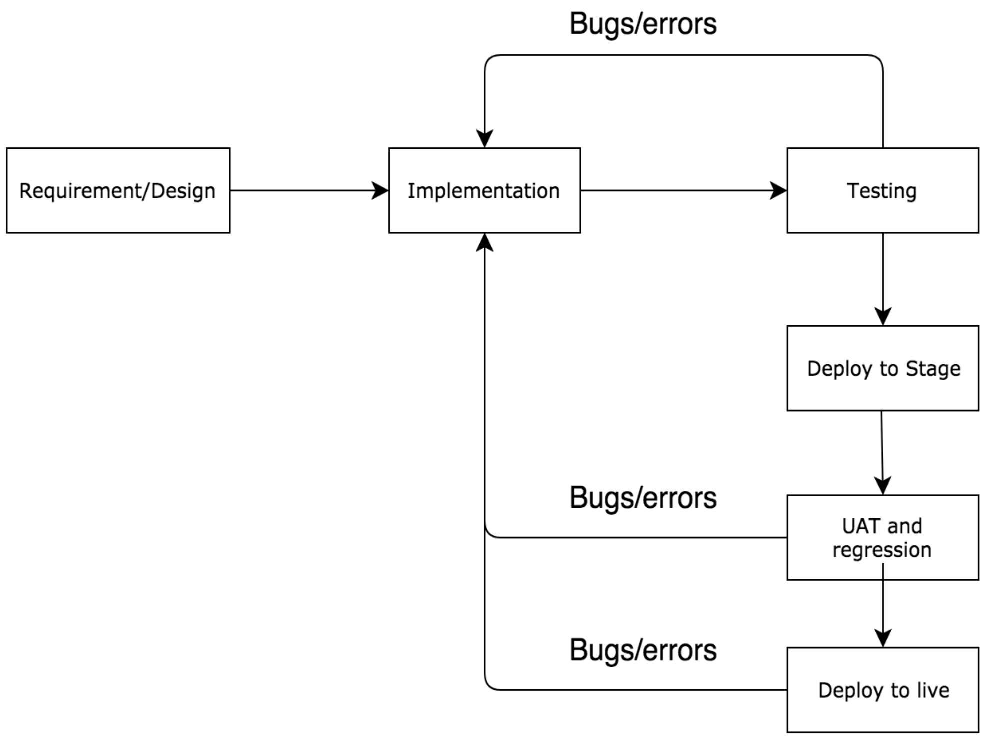


Figure 17 Development pipeline

In each sprint, the developers play the role of the testers after the implementation is complete. Taking into consideration the time-box of the sprint (two weeks), it is clear that the testing effort turns into a major problem. Another reason may be that the lack of skillsets from the development team, which leads to increasing development time and bug fixing. In order to identify the problem with convincing reasoning, a thorough SWOT analysis needs to be conducted in chapter 4.1.3

4.1.3 SWOT Analysis

Figure 18 below represents the Strengths, Weaknesses (internal factors) and Opportunities, Threats (external factors). These four forces provide great insight into the situation and identify the problematic issues:

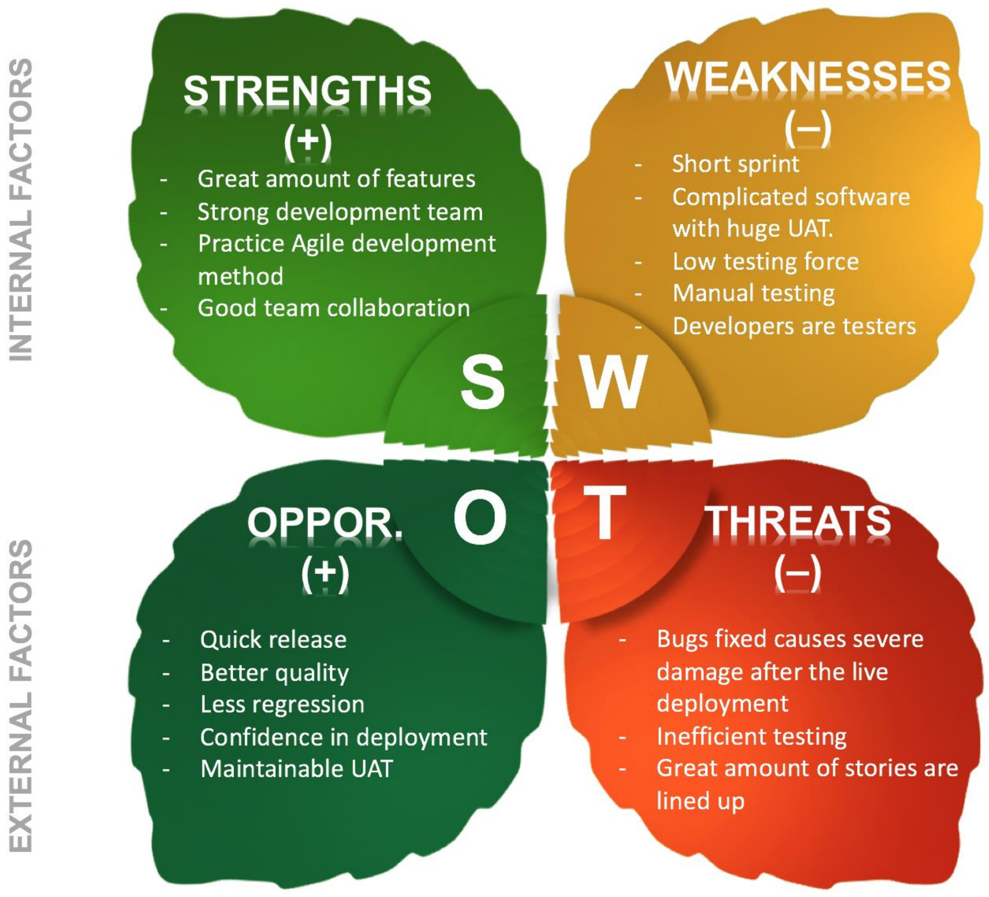

INTERNAL FACTORS
EXTERNAL FACTORS
STRENGTHS
(+)
- Great amount of features
- Strong development team
- Practice Agile development method
- Good team collaboration
WEAKNESSES
(–)
- Short sprint
- Complicated software with huge UAT.
- Low testing force
- Manual testing
- Developers are testers
S
W
O
T
OPPOR.
(+)
- Quick release
- Better quality
- Less regression
- Confidence in deployment
- Maintainable UAT
THREATS
(–)
- Bugs fixed causes severe damage after the live deployment
- Inefficient testing
- Great amount of stories are lined up

Figure 18 SWOT Analysis

In details, the developers are not able to devote themselves fully to the testing due to the fact that they are occupied with maybe more than one story. In fact, sometimes, blockers (bugs/errors/downtime from the live product) come suddenly and they are prioritized first, which leads to even more lack of time for developers. Secondly, with no proper testing practice, the UAT that lacks maintenance goes obsolete over, which leads to a redundant testing effort. As the development goes on and the bugs list increases exponentially, the whole team needs to pause, reassess and reprioritize the backlog items. The development time exceeds the allocated time-box which means a bad practice of Agile software development method.

Based on the analysis, it is undeniable that improvement is required to happen in the testing effort. With a better solution, the team can reduce the testing time and increase the quality of the product. By taking advantages of the strengths with Agile software development method and strong team collaboration, the team can make use of Automation testing to automate the UAT, fasten the finding-fixing bug time span. Undeniably, this improvement reduces the apparent threats the team is and will be having, at the same time, welcomes the opportunity for “lesser regression – better development”.

4.1.4 Artefact and Tool Selection Reasoning

There are undoubtedly many available frameworks and testing tools when it comes to Automation testing. It is imperative to select the tools that can be beneficial in terms of implementation and cost-effectiveness. It needs to be selected based on the company’s situation and resources. In this section, the author explains his reasonings in the tool selection and comparison to other tools on the market.

To choose which automation testing framework to implement, there are some conditions to take into consideration based on the company’s situation and requirements:

- The tool needs to be an excellent candidate in GUI testing: The product of the company is a desktop web-based application. GUI testing or end-to-end testing plays an important role since it affects directly the user’s behaviors.
- The tool supports functional testing. Due to the company’s product, a POM design should be easy to implement.
- The tool needs to support cross-browser testing.
- The tool needs to be easily adopted without taking too many resources, which means the development of the tool should be an independent work apart from the team source code.
- The tool needs to be cost-efficient. Ideally, the tool should be an open source framework.
- The tool needs to have a clear test report with detailed statistics and can be stored for future references.

From then, the thesis author shortlisted the potential automation testing frameworks and present their comparisons in the following Table 3:

Table 3 Comparisons between shortlisted frameworks

	Selenium	Robot Framework	Appium	AutoIT
Popularity	Very[1]	Medium[2]	Medium	Medium
POM design	Yes	No[3]	No	No
Desktop web-based GUI testing	Yes	Yes	No	No
Cross-browser testing	Yes	Yes	No	No
Resources	Low	Medium	Medium	High
Pricing	Free	Free	Free	Free
Clear test reports	Yes	Yes	Yes	No
Language supported	Many	Many	Many	Many
Color indication: 1) Green – good 2) Orange – normal 3) Red – not good				

With careful assessment based on the company's situation, apparently, Selenium Framework comes forth as the bright selected tool for test automation. For the programming language, Java is chosen due to its ability to support POM design and its stability which is discussed more in the next chapter.

4.2 Artefact Development

4.2.1 Plan and Tools

As the source code of the company is hosted in GitHub, the test suite is designed to be in the same repository for a better management. The objective of the test suite is to replace the current functional UAT being used in the company. The test suite runs on a macOS machine.

The test suite follows the standard installation referring to section 3.4.5. The components involved in the test suite includes:

- JDK and JRE
- Eclipse IDE
- Selenium WebDriver
- ChromeDriver and Chrome browser:

As Selenium supports many web browser, it is just a preference to choose Chrome as the testing browser. The developer can easily download other web drivers and its corresponding browser to execute the test in. However, it is imperative to save the web drivers in the same folder as of the test suite, or else, the path linked to the web driver needs to be declared.

- Maven and pom.xml

To install Maven, simply go to Terminal and insert “brew install mvn”. Terminal fetches the data and downloads Maven to the machine.

It downloads all the necessary libraries to run the Java file when you add them in the pom.xml file. Details of the pom.xml for the artefact can be found in Appendix 2.

- Maven Surefire plugin for test reports.

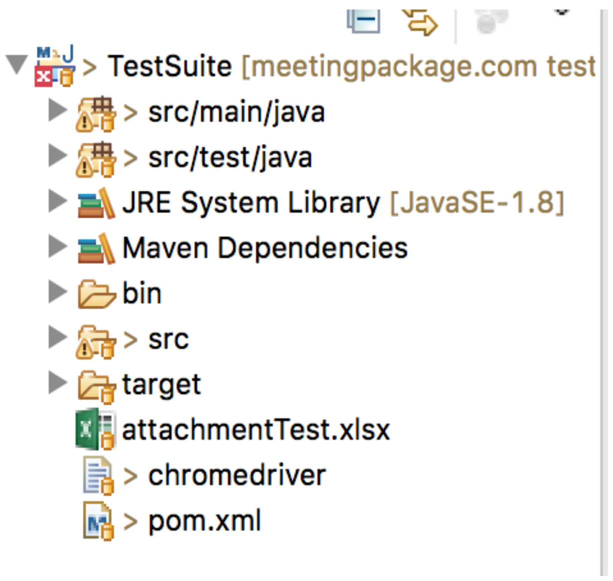
The details of the requirements that the test suite needs can be found in the following Table 4:

Table 4 System requirements to run the test suite

Name	Version	Description
Selenium-Java	3.6.0	Selenium version for Java
Java JDK	1.8.0	Java environment
Chrome Browser	61.0.3163.100	Chrome browser to generate an instance
Chrome Driver	2.32.498537	Driver to automate and control the browser
Junit	4.12	Java unit testing framework
Maven	1.8	Java build management tool
Surefire Plugin	2.20	Maven test reports plugin

4.2.2 Test Suite Architecture

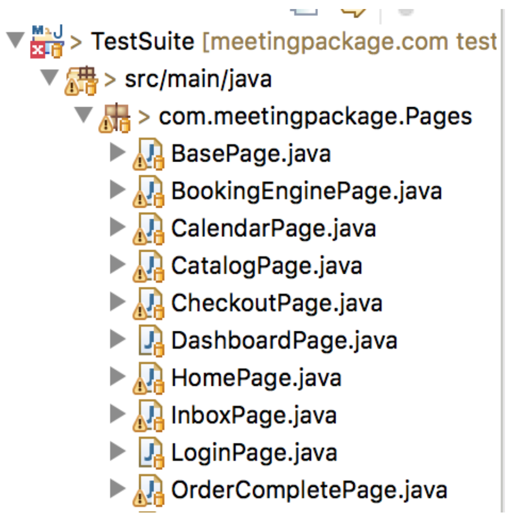
The architecture of the test suite is demonstrated using the following Figure 19:



```
▼ > TestSuite [meetingpackage.com test
  ▶ > src/main/java
  ▶ > src/test/java
  ▶ JRE System Library [JavaSE-1.8]
  ▶ Maven Dependencies
  ▶ bin
  ▶ > src
  ▶ target
    attachmentTest.xlsx
    > chromedriver
    > pom.xml
```

Figure 19 Architecture of the test suite

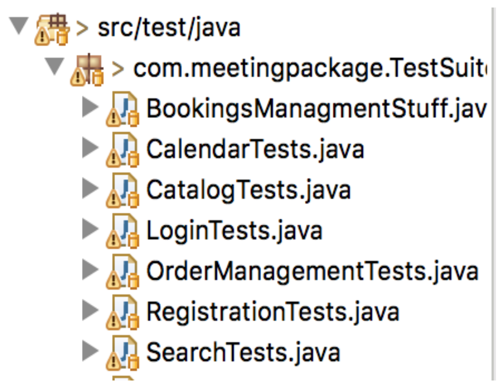
Apparently, the POM design is well-integrated in the test suite. For each page of the website, there is a corresponding .java page in the test suite (Figure 20).



```
▼ > TestSuite [meetingpackage.com test
  ▼ > src/main/java
    ▼ > com.meetingpackage.Pages
      ▶ BasePage.java
      ▶ BookingEnginePage.java
      ▶ CalendarPage.java
      ▶ CatalogPage.java
      ▶ CheckoutPage.java
      ▶ DashboardPage.java
      ▶ HomePage.java
      ▶ InboxPage.java
      ▶ LoginPage.java
      ▶ OrderCompletePage.java
```

Figure 20 Architecture of the Pages

The test cases are divided into different test scenarios based on the website’s functionality. In order for the test to run, each file must end with the keyword Test(s) (Figure 21):



```
▼ > src/test/java
  ▼ > com.meetingpackage.TestSuit
    ▶ BookingsManagmentStuff.jav
    ▶ CalendarTests.java
    ▶ CatalogTests.java
    ▶ LoginTests.java
    ▶ OrderManagementTests.java
    ▶ RegistrationTests.java
    ▶ SearchTests.java
```

Figure 21 Architecture of the Tests

The pom.xml file and the Chrome Driver for Linux are placed within the same folder (target), together with the test reports which can be later found once the test suite is executed.

4.2.3 Implementation of Test Suite

Due to the company's confidential policy, the content of the test suite is not allowed to present in this thesis. However, for the purpose of this thesis, a demonstration of a simple test case and its implementation is presented below:

Test case: User logs in successfully with correct credentials on MeetingPackge.com

Precondition: User using Chrome browser.

Steps:

- Step 1: Go to MeetingPackage.com
- Step 2: Go to the login page (Figure 22)

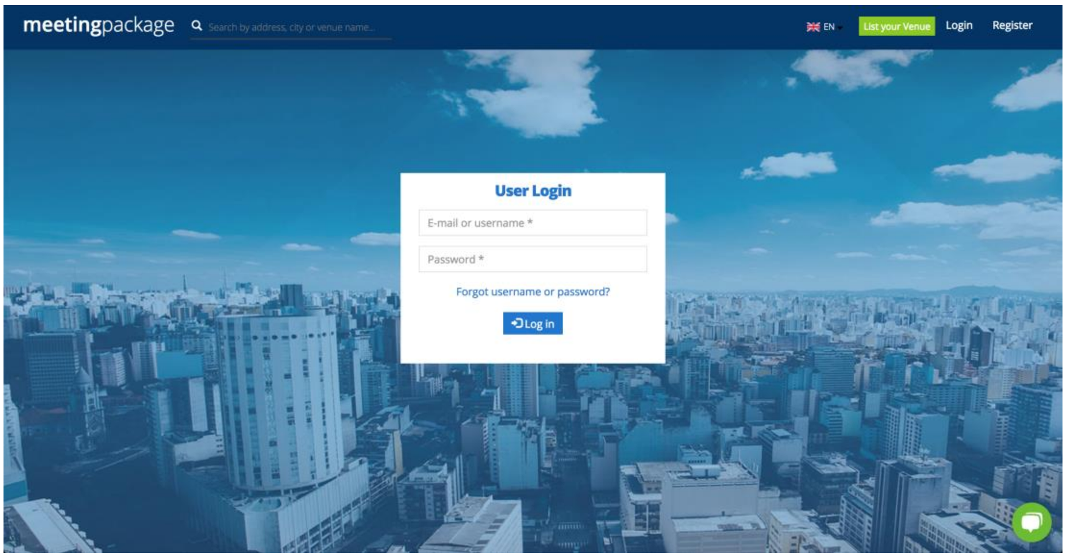


Figure 22 MeetingPackage.com Login Page (MeetingPackage.com 2018)

- Step 3: Enter the user's credential (Figure 23)

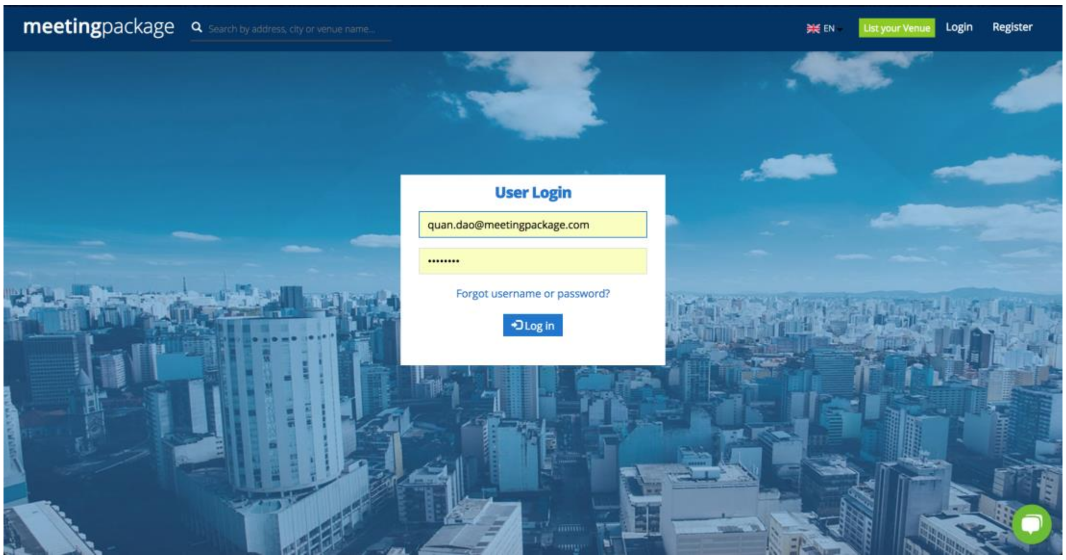

meetingpackage
Search by address, city or venue name...
EN
List your Venue
Login
Register
User Login
quan.dao@meetingpackage.com
••••••••
Forgot username or password?
Log in

Figure 23 MeetingPackage.com LogIn Page (MeetingPackage.com 2018)

- Step 4: Click the login button

Expected results: User's homepage is open indicating the login is successful (Figure 24):

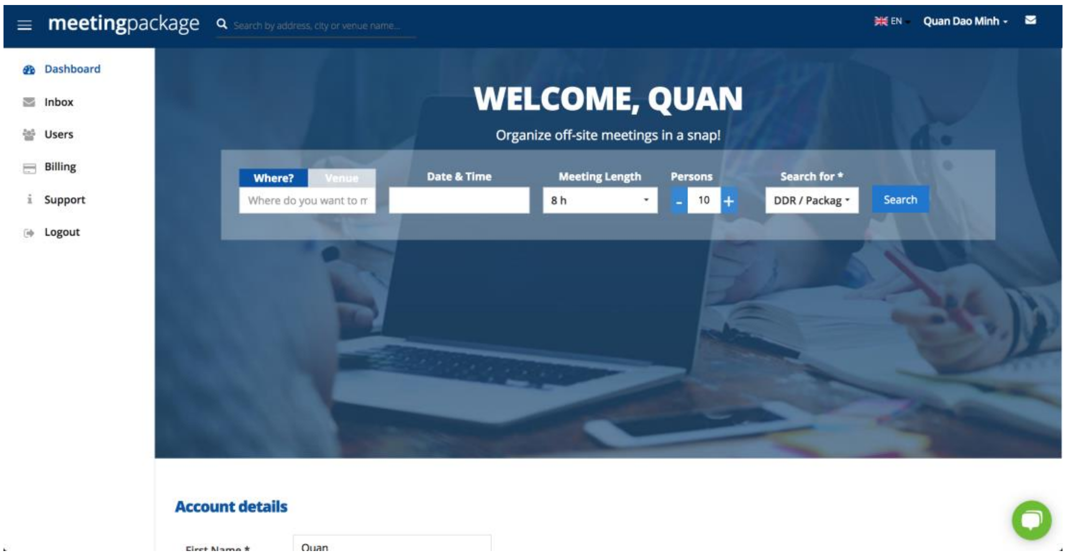
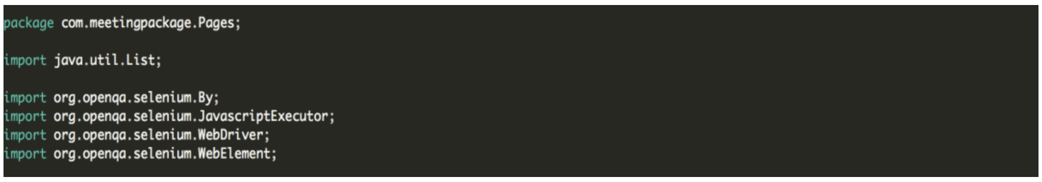


Figure 24 MeetingPackage.com Dashboard Page (MeetingPackage.com 2018)

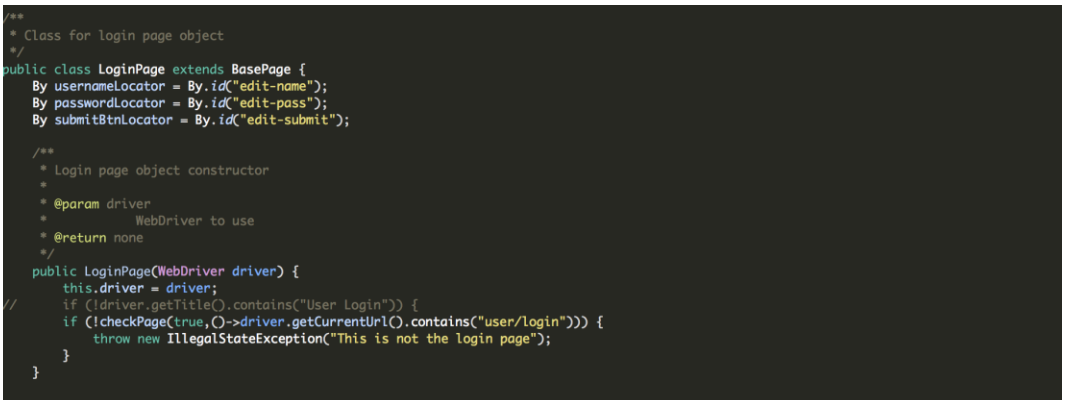
The test case implementation starts with building the Login java page. The Login Page imports all the necessary libraries such as By, JavascriptExecutor, WebDriver and WebElement from Selenium (Figure 25).



```java
package com.meetingpackage.Pages;

import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
```

Figure 25 Required packages

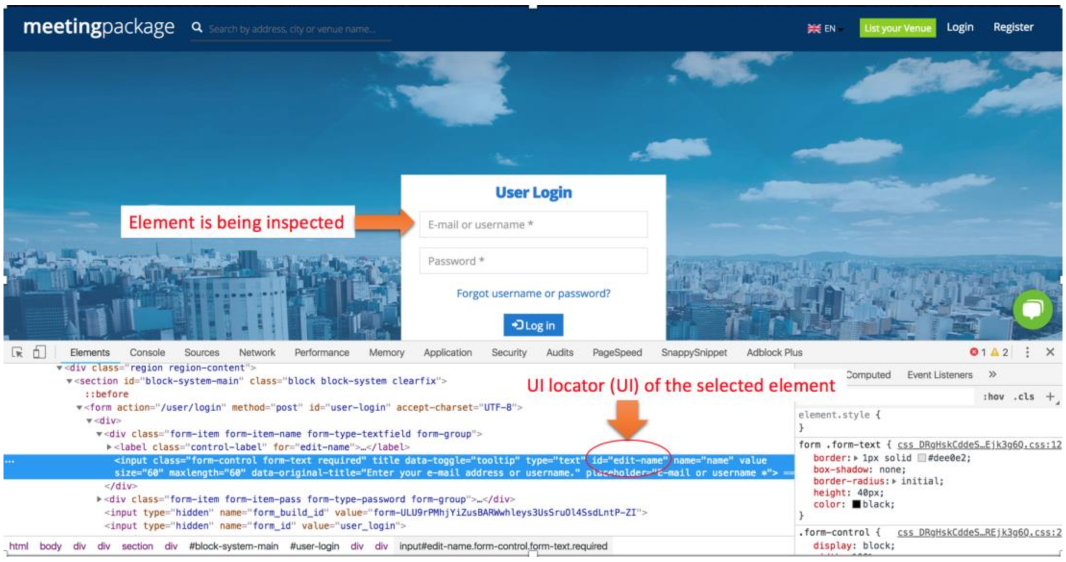
After that, the developer locates the UI locator with its ID on the DOM and pass it to the variables. In a good practice, it is recommended to check the current page every time it loads by checking the URL matched (Figure 26).



```
/**
 * Class for login page object
 */
public class LoginPage extends BasePage {
    By usernameLocator = By.id("edit-name");
    By passwordLocator = By.id("edit-pass");
    By submitBtnLocator = By.id("edit-submit");

    /**
     * Login page object constructor
     *
     * @param driver
     *            WebDriver to use
     * @return none
     */
    public LoginPage(WebDriver driver) {
        this.driver = driver;
//        if (!driver.getTitle().contains("User Login")) {
        if (!checkPage(true,()->driver.getCurrentUrl().contains("user/login"))) {
            throw new IllegalStateException("This is not the login page");
        }
    }
```

Figure 26 Selenium Login Page

Lastly, a function is implemented. The Driver locates the username and password textbox by the UI locator and sends the desired credentials by the function sendKeys from Selenium. The developer can inspect the element on Chrome browser to get the UI locator (Figure 27)

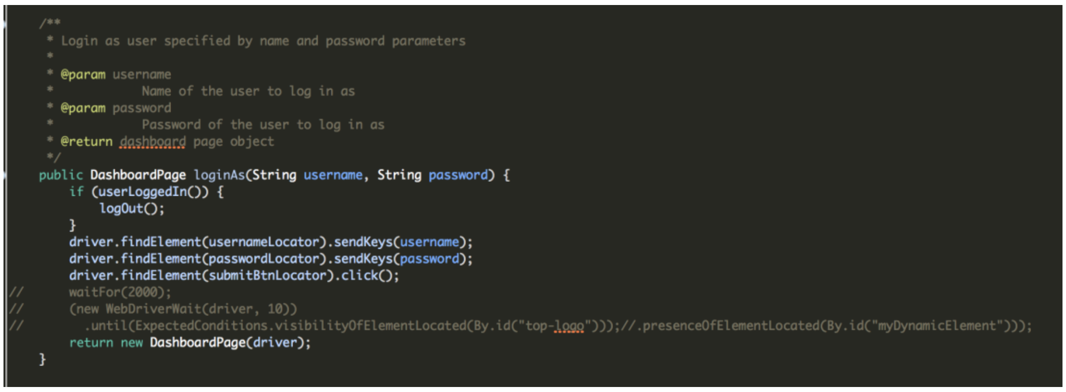


```
▼<div class="region region-content">
  ▼<section id="block-system-main" class="block block-system clearfix">
     ::before
   ▼<form action="/user/login" method="post" id="user-login" accept-charset="UTF-8">
     ▼<div>
       ▼<div class="form-item form-item-name form-type-textfield form-group">
         ▶<label class="control-label" for="edit-name">…</label>
          <input class="form-control form-text required" title data-toggle="tooltip" type="text" id="edit-name" name="name" value
          size="60" maxlength="60" data-original-title="Enter your e-mail address or username." placeholder="E-mail or username *"> ==
        </div>
       ▶<div class="form-item form-item-pass form-type-password form-group">…</div>
        <input type="hidden" name="form_build_id" value="form-ULU9rPMhjYiZusBARWwhleys3UsSru0l4SsdLntP-ZI">
        <input type="hidden" name="form_id" value="user_login">
html body div div section div #block-system-main #user-login div div input#edit-name.form-control.form-text.required
```

```
element.style {
}
form .form-text {   css_DRgHskCddeS…Ejk3g60.css:12
  border: ▶ 1px solid #dee0e2;
  box-shadow: none;
  border-radius: ▶ initial;
  height: 40px;
  color: black;
}
.form-control {   css_DRgHskCddeS…REjk3g60.css:2
  display: block;
```

Figure 27 MeetingPackage.com Login Page (MeetingPackage.com 2018)

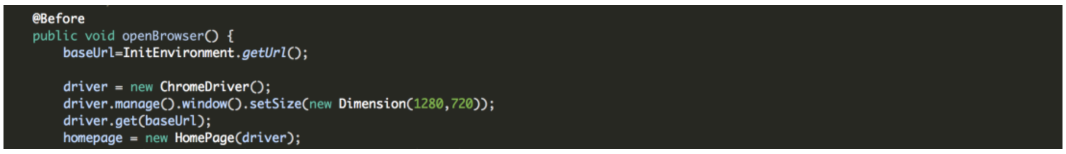
Then an action “Click” on the submit button is executed. The function will return the Dashboard page of the logged-in user, if not, the system will inform that “This is not the login page” as a test result (Figure 28)



```
    /**
     * Login as user specified by name and password parameters
     *
     * @param username
     *            Name of the user to log in as
     * @param password
     *            Password of the user to log in as
     * @return dashboard page object
     */
    public DashboardPage loginAs(String username, String password) {
        if (userLoggedIn()) {
            logOut();
        }
        driver.findElement(usernameLocator).sendKeys(username);
        driver.findElement(passwordLocator).sendKeys(password);
        driver.findElement(submitBtnLocator).click();
//      waitFor(2000);
//      (new WebDriverWait(driver, 10))
//        .until(ExpectedConditions.visibilityOfElementLocated(By.id("top-logo")));//.presenceOfElementLocated(By.id("myDynamicElement")));
        return new DashboardPage(driver);
    }
```

Figure 28 Selenium Login Page

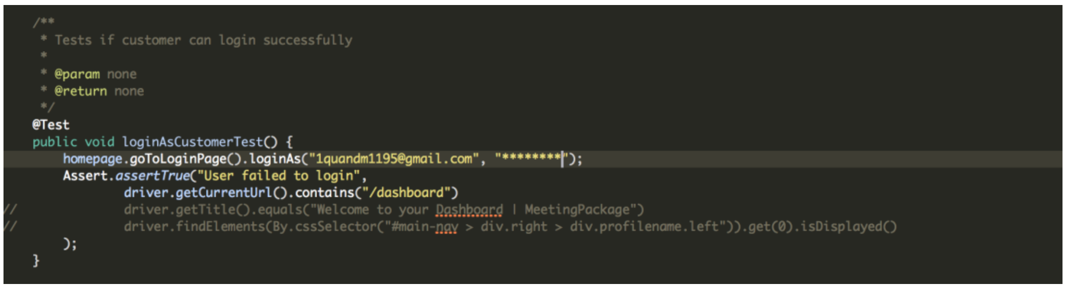
In the test case, the Before function indicates what the driver should do before any tests are executed. Here, the driver must open the driver, resize the dimension and get to the Homepage of MeetingPackage.com. Failure to execute this leads to an error at runtime (Figure 29).



```
    @Before
    public void openBrowser() {
        baseUrl=InitEnvironment.getUrl();

        driver = new ChromeDriver();
        driver.manage().window().setSize(new Dimension(1280,720));
        driver.get(baseUrl);
        homepage = new HomePage(driver);
```

Figure 29 Selenium Login Test

Then the test script is created. In Java, it is imperative to have the hook @Test to indicate it is a test script. The steps are well-defined as function name (goToLoginPage, loginAs). Assertion is a way of result reporting. After the login button is clicked, if the current URL that the driver gets contains the word “/dashboard” in it, the assertionTrue is correct, meaning the test passed. If there is no “/dashboard” in the current URL, the assertTrue is false, which mean the test fails (Figure 30).



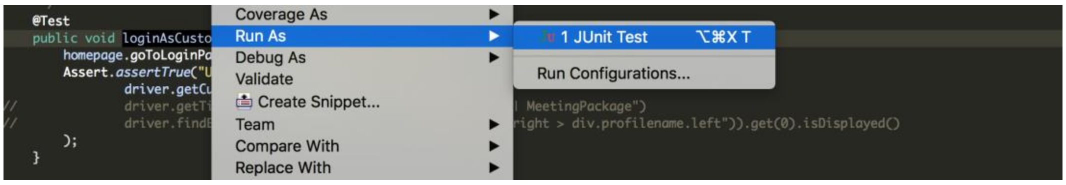
```
    /**
     * Tests if customer can login successfully
     *
     * @param none
     * @return none
     */
    @Test
    public void loginAsCustomerTest() {
        homepage.goToLoginPage().loginAs("1quandm1195@gmail.com", "********");
        Assert.assertTrue("User failed to login",
                driver.getCurrentUrl().contains("/dashboard")
//              driver.getTitle().equals("Welcome to your Dashboard | MeetingPackage")
//              driver.findElements(By.cssSelector("#main-nav > div.right > div.profilename.left")).get(0).isDisplayed()
        );
    }
```

Figure 30 Selenium Login Test

4.3 Test Suite Execution

The test suite can be executed using Maven or by Junit depending on the scope of the test:

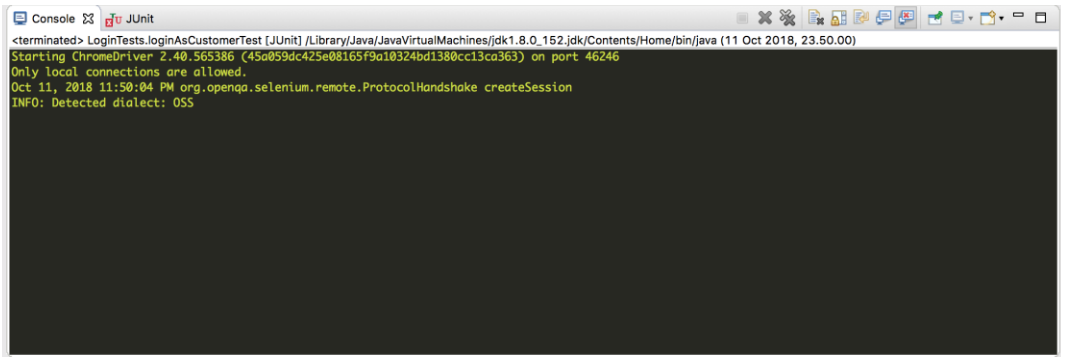
- Single test case: Right-click on the test case, choose Run As and select JUnit test (Figure 31).



```
    @Test
    public void loginAsCusto
        homepage.goToLoginPa
        Assert.assertTrue("U
                driver.getCu
//              driver.getTi                                    | MeetingPackage")
//              driver.findE                                    right > div.profilename.left")).get(0).isDisplayed()
        );
    }
```

Figure 31 Selenium Login Test

After the execution, the program fires off the test case with a notice from the console (Figure 32).



```
<terminated> LoginTests.loginAsCustomerTest [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java (11 Oct 2018, 23.50.00)
Starting ChromeDriver 2.40.565386 (45a059dc425e08165f9a10324bd1380cc13ca363) on port 46246
Only local connections are allowed.
Oct 11, 2018 11:50:04 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
```

Figure 32 Selenium Login Test

An instance of Chrome Browser is created, the test is executed based on the test script. The programme indicates that the driver is up and running by the prompt “Chrome is being controlled by automated test software.” (Figure 33).

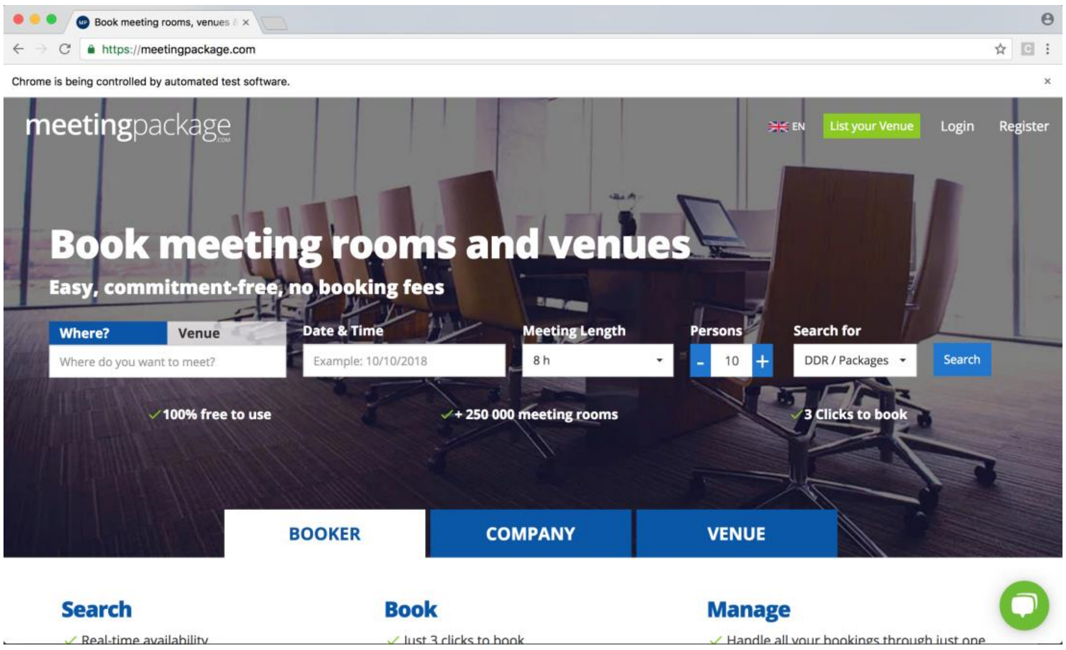
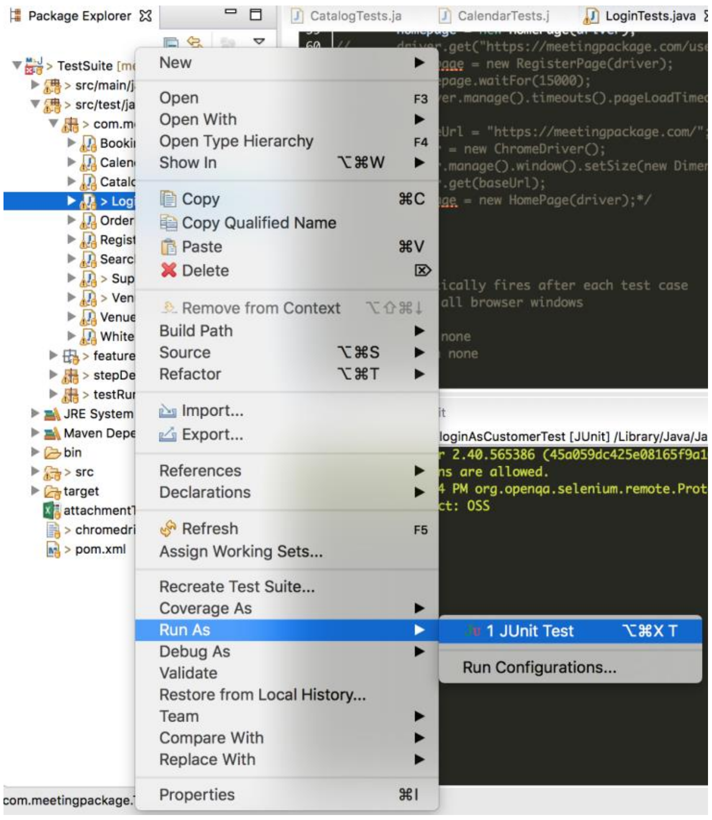


Figure 33 Selenium Browser Instance

- Multiple test cases (one test scenario): Right-click on the test scenario and choose Run As, select JUnit to run the test scenario. All the test cases are executed one by one (Figure 34).

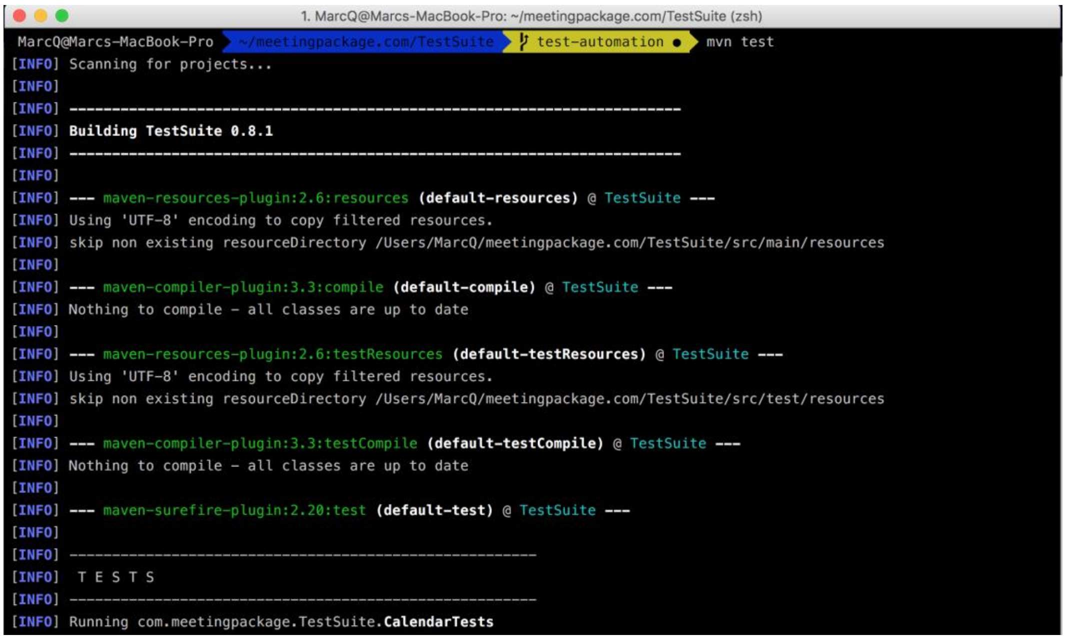


```
river.get("https://meetingpackage.com/use
page = new RegisterPage(driver);
epage.waitFor(15000);
er.manage().timeouts().pageLoadTime
eUrl = "https://meetingpackage.com/";
 = new ChromeDriver();
.manage().window().setSize(new Dimen
.get(baseUrl);
ge = new HomePage(driver);*/

ically fires after each test case
 all browser windows

none
 none
```

```
loginAsCustomerTest [JUnit] /Library/Java/Ja
r 2.40.565386 (45a059dc425e08165f9a1
ns are allowed.
4 PM org.openqa.selenium.remote.Prot
ct: OSS
```

Figure 34 Selenium Test Suite

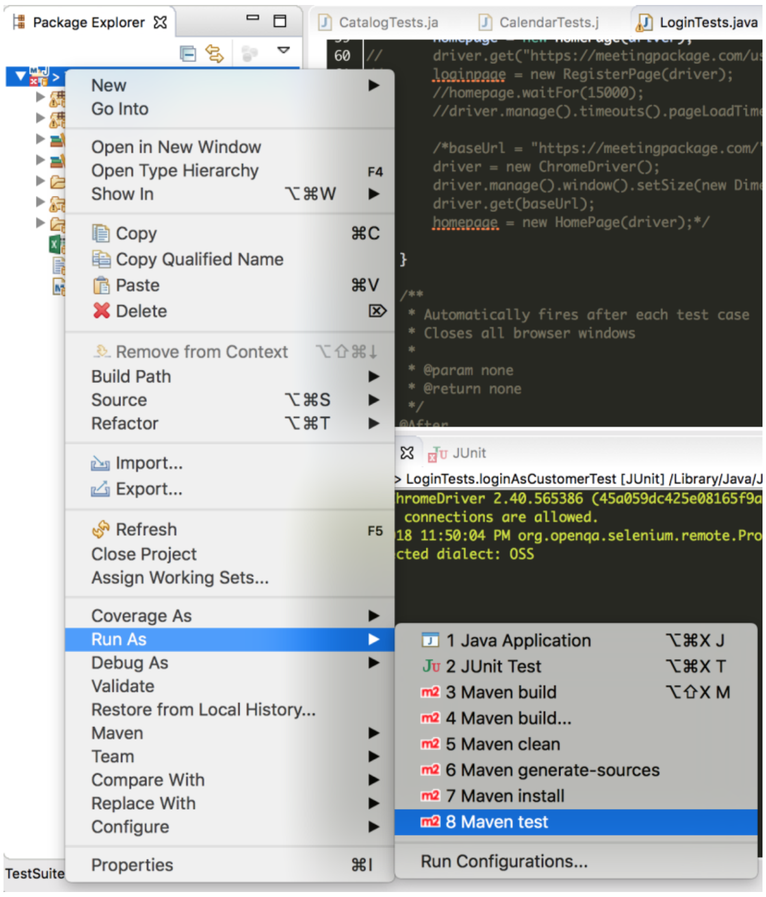
- All the test scenarios (Test suite): to run the test suite, simply navigate to the project folder in the Terminal and type in “mvn test” (Figure 35):


1. MarcQ@Marcs-MacBook-Pro: ~/meetingpackage.com/TestSuite (zsh)

```
MarcQ@Marcs-MacBook-Pro  ~/meetingpackage.com/TestSuite  test-automation ●  mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building TestSuite 0.8.1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ TestSuite ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/MarcQ/meetingpackage.com/TestSuite/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.3:compile (default-compile) @ TestSuite ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ TestSuite ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/MarcQ/meetingpackage.com/TestSuite/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.3:testCompile (default-testCompile) @ TestSuite ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.20:test (default-test) @ TestSuite ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.meetingpackage.TestSuite.CalendarTests
```

Figure 35 Terminal interface

As the result, Maven performs the test suite by compiling and run from the first test scenario. Later on, the test results can be collected in the Target folder. Another way to trigger the test suite is to right-click on the pom.xml file and select Run As and click on Maven test (Figure 36)



```
//      driver.get("https://meetingpackage.com/us
        loginpage = new RegisterPage(driver);
        //homepage.waitFor(15000);
        //driver.manage().timeouts().pageLoadTime

        /*baseUrl = "https://meetingpackage.com/"
        driver = new ChromeDriver();
        driver.manage().window().setSize(new Dime
        driver.get(baseUrl);
        homepage = new HomePage(driver);*/

    }

    /**
     * Automatically fires after each test case
     * Closes all browser windows
     *
     * @param none
     * @return none
     */
```

```
> LoginTests.loginAsCustomerTest [JUnit] /Library/Java/J
hromeDriver 2.40.565386 (45a059dc425e08165f9a
 connections are allowed.
18 11:50:04 PM org.openqa.selenium.remote.Pro
cted dialect: OSS
```

Figure 36 Selenium Test Suite

4.4 Results and Test Reports

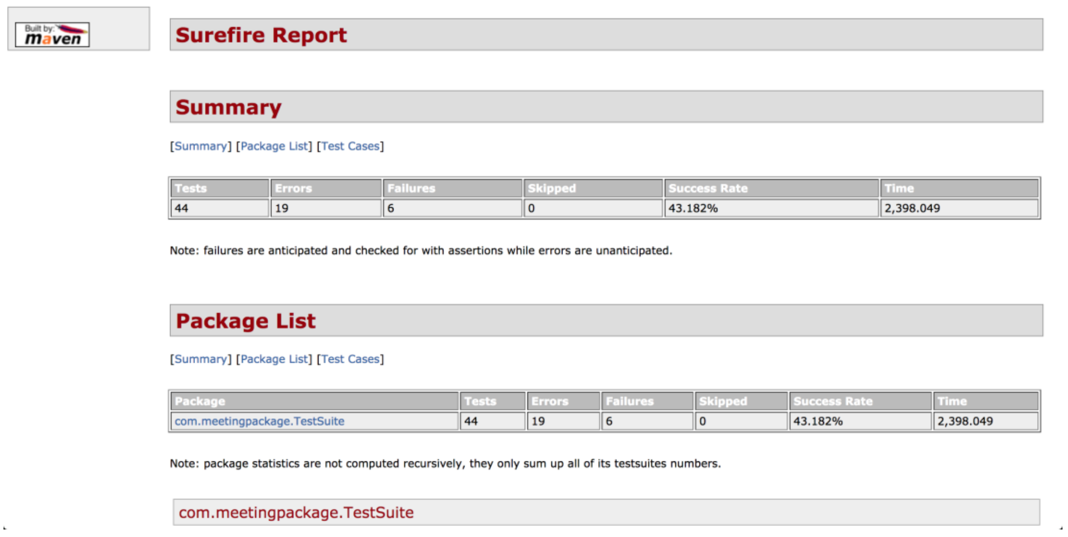
Maven Surefire plugin is utilized to generate a nice HTML and CSS test reports. After the test suite finished, navigate to the terminal and type in:

```
mvn surefire-report:report-only
```

to generate an HTML report, then to add CSS, simply run the following command:

```
mvn site -DgenerateReports=false
```

After that, the test report is located at <your project folder>/target/site/surefire-report.html>. In the Summary, the report demonstrates the number of test cases has been run, the errors, failures and time of execution (Figure 37):



Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
44	19	6	0	43.182%	2,398.049

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

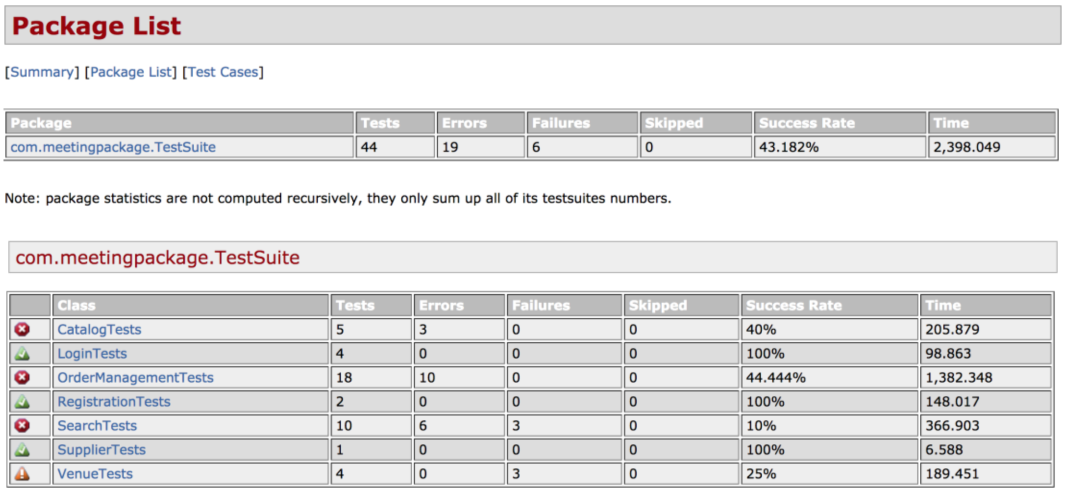
Package	Tests	Errors	Failures	Skipped	Success Rate	Time
com.meetingpackage.TestSuite	44	19	6	0	43.182%	2,398.049

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

com.meetingpackage.TestSuite

Figure 37 Test report

Furthermore, in the package list, the developer can also observe the performance of each test scenario (Figure 38):



Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
com.meetingpackage.TestSuite	44	19	6	0	43.182%	2,398.049

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

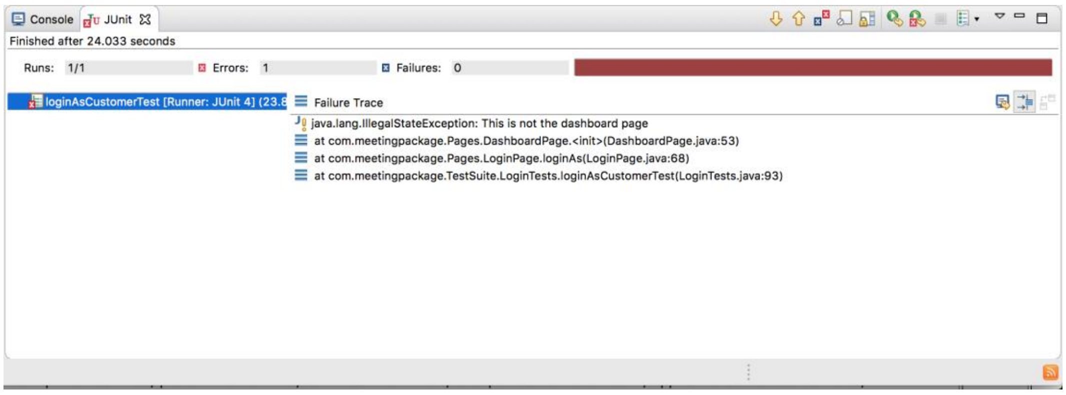
com.meetingpackage.TestSuite

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
✖	CatalogTests	5	3	0	0	40%	205.879
✓	LoginTests	4	0	0	0	100%	98.863
✖	OrderManagementTests	18	10	0	0	44.444%	1,382.348
✓	RegistrationTests	2	0	0	0	100%	148.017
✖	SearchTests	10	6	3	0	10%	366.903
✓	SupplierTests	1	0	0	0	100%	6.588
⚠	VenueTests	4	0	3	0	25%	189.451

Figure 38 Test Report

The test case section reveals the details behind the failures or errors.

Alternatively, when running a single test case or a scenario without using Maven, the result can be observed from the Junit console (Figure 39):



```
Console  JUnit
Finished after 24.033 seconds
Runs: 1/1    Errors: 1    Failures: 0

loginAsCustomerTest [Runner: JUnit 4] (23.6
Failure Trace
java.lang.IllegalStateException: This is not the dashboard page
at com.meetingpackage.Pages.DashboardPage.<init>(DashboardPage.java:53)
at com.meetingpackage.Pages.LoginPage.loginAs(LoginPage.java:68)
at com.meetingpackage.TestSuite.LoginTests.loginAsCustomerTest(LoginTests.java:93)
```

Figure 39 Junit

The failure trace helps the developer to understand the problem and can right-click to jump into the problematic code. The system throws an error and prints out the message which followed by all the related files.

5 ANALYSIS OF THE RESULTS

In order to understand the benefit that the artefact yields, it is imperative to gather data of both scenarios (before and after the artefact), then make a comparison between the time consumptions under the same time condition. The result of the analysis determines the effectiveness of the artefact to the case company. This section introduces the thesis author's methods to gather data and calculate the performance.

5.1 Time Consumption before the Artefact

The company UAT has a total of 50 use cases. On average, each use case contains 5 test cases and it takes 5 minutes for one developer to execute 1 test case. However, there is approximately 20% of the use case defined as complicated and it requires at least 10 test cases to cover. Furthermore, the maintenance time for the UAT is 10% of the testing time for each round and the development time is three working days (24 hours). These data are recorded and collected during the development time by the thesis author using field notes as the data collection method. This maximizes the accuracy of the data collected as it directly records the current behaviors. With a simple calculation, when solely performing the UAT testing, one developer needs to spend (in hours):

Manual testing time = (50 * 0.2 * 10 * 5) + (50 * 0.8 * 5 * 5)

= 500 + 1000

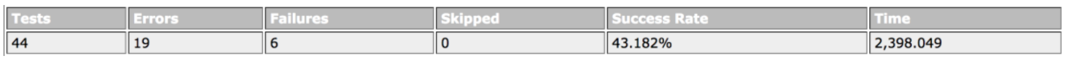
= 1500 (minutes) /60 = 25 Hours

For one developer, 25 hours equal to more than a half of a full working week. As the sprint is only two weeks, it means that the developer needs to spend one entire week just for testing due to the fact that there are at least two rounds of testing on different environments (refer to section 4.1.2). It is evident it is not possible to cover the testing of the whole system within the desired time period.

5.2 Time Consumption after the Artefact

The thesis author observes and collects the data of time required to execute the test suite. From 50 use cases, there are 44 automation test cases due to the fact that in one test case, there can be more than one assertion, which means there are test cases sharing the same initial condition.

Figure 40 below presents the time to execute the test suite in seconds.



Tests	Errors	Failures	Skipped	Success Rate	Time
44	19	6	0	43.182%	2,398.049

Figure 40 Test Report

After converting, it takes roughly 40 minutes (0,67 hours) for the test suite to run and present the results. However, the time it takes to develop and maintain the test suite also needs to be taken into consideration. The test suite is implemented in two weeks (37,5 working hours/week), together with its maintenance which could add up 30% of the testing time for each round.

5.3 Comparison and Result

After the aforementioned time-consumption to perform the functional testing, the thesis author decided to draw out one condition to compare between the two scenarios: Given that in two months of development (four sprints), one developer needs to perform the entire functional testing for four times each sprint, the Equation 1 refers to a method of calculating the time he/she needs to spend on:

Total time consumption (Hours) = Development time + (time for 1 round of testing + maintenance time) * rounds of testing * number of sprints

Equation 1 Time consumption for testing

Table 5 is the result when parsing the data to the formula:

Table 5 Formula and Calculations

	Before the artefact	After the artefact
Data parsing to Equation 1	3 + (25 + 25*10%) * 4 * 4	37,5*2 + (0,67 + 0,67*30%) * 4 * 4
Total time consumption	443 (Hours)	126,43 (Hours)

As a result, the time reduction that the artefact brings up is = (443 – 126,43) /443 * 100% = **<u>71,46%</u>**

With a time reduction of 71,46 percentage, it is clear that the test suite has made a great impact on the testing effort. Even though it takes a certain amount of effort to implement at first, later on, the developer is completely unoccupied while the test suite is running. Therefore, there is no doubt that the further into development in both time and testing demand, the higher the benefit rises. In the end, the artefact stands out as an effective answer for the case company.

6 CONCLUSION

6.1 Answering the Research Question

The objective of this thesis is to demonstrate how Selenium Framework benefits the case company with automation testing. In the theoretical part, the thesis author mentioned all the necessary knowledge and methodologies to support his reasonings and actions. With the design science research method, the thesis author follows strictly the steps in the research, from raising the problem, analyzing it to suggest a solution, develop it and evaluate the results. Finally, based on the data collected after the implementation of the artefact, the thesis author answers the research question:

How does Selenium Framework reduce time consumption in Quality Assurance at MeetingPackage.com?

Simply put, based on the finding after analyzing the data, a reduction of 71,46% in time consumption successfully proves that the artefact made a significant impact to MeetingPackage.com in the development process. The comparison has been made in a same fair condition in order to maximize the validity of the results. With the adoption of the artefact, the development team can save a tremendous amount of time and resources, increase their productivity and fasten the delivery. In other words, the artefact revolutionized the development process of the case company.

6.2 Limitations

The thesis compares the time consumption from designing to the execution of the test suite. However, there can be more criteria to compare, for example, the complexity of the development, the condition of the resources and the ability to maintain the artefact. Therefore, it is suggested to take into consideration more criteria to identify the effectiveness of the artefact.

6.3 Reliability and Validity

In this thesis, the thesis author observes and analyzes the data during the development process. For the estimation of time consumption when it comes to UAT testing, the time for each test case is an average approximation. In addition, the automation test results have some error test cases, which means that the test case could not run for some internal reason. As a result, the time consumption for the test suite may increase when the problem is resolved. Therefore, the reliability may not be entirely correct, however, the significant benefit of the artefact remains at least more than 65% without any doubt. Lastly, there can be some other components could affect the study, for example, the skill of the author, the size of the company, the size of the development team, the selected solution or programming language.

6.4 Suggestions for Further Study

As the main product of the case company is a web application, the study main target is the website test automation. However, with Java, the developer can run the test cases concurrently, which means that there can be more than two test cases being executed at the same time. Needless to say, the time to run the whole build reduces significantly. Further study can focus on parallelism of the test cases and design structure for concurrent builds.

Furthermore, implementation of the test suite with continuous integration can be of further study for the company. Continuous integration helps the development more efficiently because the team can commit code more frequently, trigger the build to run the test suite, and receive results after each commit. Therefore, it results in lesser integration problem, no broken code is leaked to the live product and decrease the time of peer-review code.

7 SUMMARY

Over the past few years, development teams on all over the world have been striving for better performance by experimenting with different development methodologies, and Agile practice comes forth as one possible candidate. By employing the Scrum method, the development team gains a great amount of benefit in terms of product transparency and quality. However, struggles to reduce the testing time while maintaining short sprints remain unsolved. The goal of this study is to identify how test automation resolves that underlying problem through demonstrating the implementation of the artefact (Java test suite with Selenium Framework) and analyzing its results.

This research contains three main sections. Theoretical framework including Agile software development methodology, Scrum, testing principles, Git, Selenium WebDriver and Java is described in chapter 3. Chapter 4 explains the case company’s situation, the reasoning, and implementation of the artefact, followed by chapter 5 in which data collection and analysis are discussed. Finally, chapter 6 concludes the thesis, its limitations, validity and suggests further study.

To summarize, it can be concluded that test automation may require time and resource at first, but in the long run, its benefits and advantages are undeniable for any company, especially for regression and functional testing.

LIST OF REFERENCES

Published Sources

Ashmore, S. & Runyan, K. 2015. Introduction to Agile Methods. Crawfordsville, Indiana, United States: Pearson Education, Inc.

Avasarala, S. 2014. Selenium WebDriver Practical Guide. Birmingham B3 2PB, UK: Packt Publishing.

Barab, S. & Squire, K. 2004. Design-based Research: Putting a Stake in the Ground. The Journal of the Learning Sciences, 13(1), 1 –14.

Bath, G. & McKay, J. 2008. The Software Test Engineer's Handbook. Santa Barbara, CA: Rocky Nook Inc.

Bentley, J.E. 2004. Software Testing Fundamentals - Concepts, Roles, and Terminology, Wachovia Bank, Charlotte NC.

Crispin, L. & Gregory, J. 2008. Agile Testing: A Practical Guide for Testers and Agile Teams. Crawfordsville, Indiana, United States: Pearson Education Inc.

Dustin, E., Garrett, T. & Gauf, B. 2009. Books on Google Play Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. New York City, New York, United States: Pearson Education Inc.

Ekas, L & Will, S. 2013. Being Agile: Eleven Breakthrough Techniques to Keep You from "Waterfalling Backward". Massachusetts, United States: IBM Press

Graham, D. & Veenendaal, EV. & Evans, I. & Black, R. 2014. Foundations of Software Testing: ISTQB Certification.

Henry, P. 2008. The Testing Network: An Integral Approach to Test Activities in Large Software Projects. Springer, Heidelberg.

Hevner, A., 2004. Design Science in Information Systems Research. Minneapolis: MIS Quarterly.

Itkonen, J., Mäntylä, M.V. & Lassenius, C. 2009. How do testers do it? An exploratory study on manual testing practices, in Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement.

Larman, C. 2004. Agile and Iterative Development: A Manager's Guide. Boston, MA, United States: Pearson Education, Inc.

Loeliger, J. & McCullough, M. 2012. Version Control with Git. Second Edition. Sebastopol: O'Reilly Media, Inc.

McQuaid, M. 2015. Git in Practice. Greenwich: Manning Publications Co.

Myers, 1979.G Myers. The Art of Software Testing. John Wiley & Sonc, Inc., New York.

Nguyen, H.Q., Hackett, M & Whitlock, B.K. 2006. Global Software Test Automation: A Discussion of Software Testing for Executives. Cuperrtino, CA, United States: Happy About

Peffers, K., Tuunanen, T.A., Rothenberger, M. & Chatterjee, S. 2008. A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems. Abingdon: M.E. Sharpe, Inc.

Ransome, J & Misra, A. 2014. Core Software Security: Security at the Source. Boca Raton, FL, United States: CRC Press

Resnick, S. De la Maza, M. & Bjork, A. 2011. Professional Scrum with Team Foundation Server 2010. Canada: Wiley Publishing, Inc.

Somasundaram, R. 2013. Git: Version Control for Everyone. Birmingham: Packt Publishing Ltd.

Saunders, M., Lewis, P. & Thornhill, A. 2012. Research Methods for Business Students. 6th ed. Harlow, England: Pearson Education Limited.

Schwaber, K. & Beedle, M. 2002. Agile Software Development with Scrum. Upper Saddle River, NJ: Prentice-Hall.

Westby, E.J.H. 2015. Git for Teams: A User-Centered Approach to Creating Efficient Workflows in Git. Sebastopol: O’Reilly Media, Inc.

Electronic Sources

Atlassian 2018. What is Git? Atlassian [accessed 1 October 2018]. Available at: https://www.atlassian.com/git/tutorials/what-is-git#security

Assignmentpoint 2018. About Abductive Reasoning. Assignmentpoint [accessed 2 October 2018]. Available at: http://www.assignmentpoint.com/science/mathematic/about-abductive-reasoning.html

Boer, G. 2017. What is Scrum? Microsoft [accessed 1 October 2018]. Available at: https://docs.microsoft.com/en-us/azure/devops/learn/agile/what-is-scrum

Brown, K. 2016. What Is GitHub, and What Is It Used For? How-To Geek [accessed 2 October 2018]. Available at: https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/

Eclipse. 2018. Download Eclipse Technology that is right for you. Eclipse [accessed 1 October 2018]. Available at: https://www.eclipse.org/downloads/

Fernandes, J & Fonzo, A.D. 2018. When to Automate your Testing (and when not to). Oracle [accessed 1 October 2018]. Available at: http://www.oracle.com/technetwork/cn/articles/when-to-automate-testing-1-130330.pdf

Geerts, G.L. 2011. International Journal of Accounting Information Systems. Elsevier [accessed 1 October 2018]. Available at: https://www.sciencedirect.com/science/article/pii/S1467089511000200

Guru99. 2018. How to Download & Install Selenium WebDriver. Guru99 [accessed 2 October 2018]. Available at: https://www.guru99.com/installing-selenium-webdriver.html

Guru99, 2018. Page Object Model (POM) & Page Factory in Selenium: Complete Tutorial. Guru99 [accessed 1 October 2018]. Available at: https://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html

Grahai, A. 2017. What is the Fundamental Test Process? testingexcellence.com [accessed 5 October 2018]. Available at: https://www.testingexcellence.com/fundamental-test-process-software-testing/

Gurendo, D. 2015. Software Development Life Cycle (SDLC). Scrum Model Step by Step. XB Software [accessed 5 October 2018]. Available at: https://xbsoftware.com/blog/software-development-life-cycle-sdlc-scrum-step-step/

Lotz, M. 2013. Waterfall vs. Agile: Which is the Right Development Methodology for Your Project? Segue Technologies [accessed 5 October 2018]. Available at: https://www.seguetech.com/waterfall-vs-agile-methodology/

Maven 2018. Maven in 5 minutes. Maven [accessed 5 October 2018]. Available at: https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

MeetingPackage.com 2018. MeetingPackage.com [accessed 10 October 2018]. Available at: https://meetingpackage.com/

Oracle 2018. Java SE Development Kit 11 Downloads. Oracle [accessed 2 October 2018]. Available at: https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html

Quora 2018. How does the Selenium Webdriver work? Quora [accessed 3 October 2018]. Available at: https://www.quora.com/How-does-the-Selenium-WebDriver-work

Selenium 2018. What is Selenium? Selenium [accessed 2 October 2018]. Available at: https://www.seleniumhq.org/

Smartbear 2018. What is Automated Testing? Smartbear [accessed 4 October 2018]. Available at: https://smartbear.com/learn/automated-testing/what-is-automated-testing/

Statista 2018. Number of internet users worldwide from 2005 to 2017 (in millions). Statista [accessed 1 October 2018]. Available at: https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/

Spilka, D. 2016. 15 Common Website User Experience Issues (And Solutions). Solvid.co.uk [accessed 5 October 2018]. Available at: https://solvid.co.uk/15-common-website-ux-issues/

Shuttleword, M. 2008. Abductive Reasoning. Explorable.com [accessed 10 October 2018]. Available at: https://explorable.com/abductive-reasoning

Techopedia 2018. Java. Techopedia [accessed 4 October 2018]. Available at: https://www.techopedia.com/definition/3927/java

Vaishnavi, V. & Kuechler, B. 2012. Design Science Research in Information Systems. Desrist [accessed 5 October 2018]. Available at: http://desrist.org/desrist/article.aspx

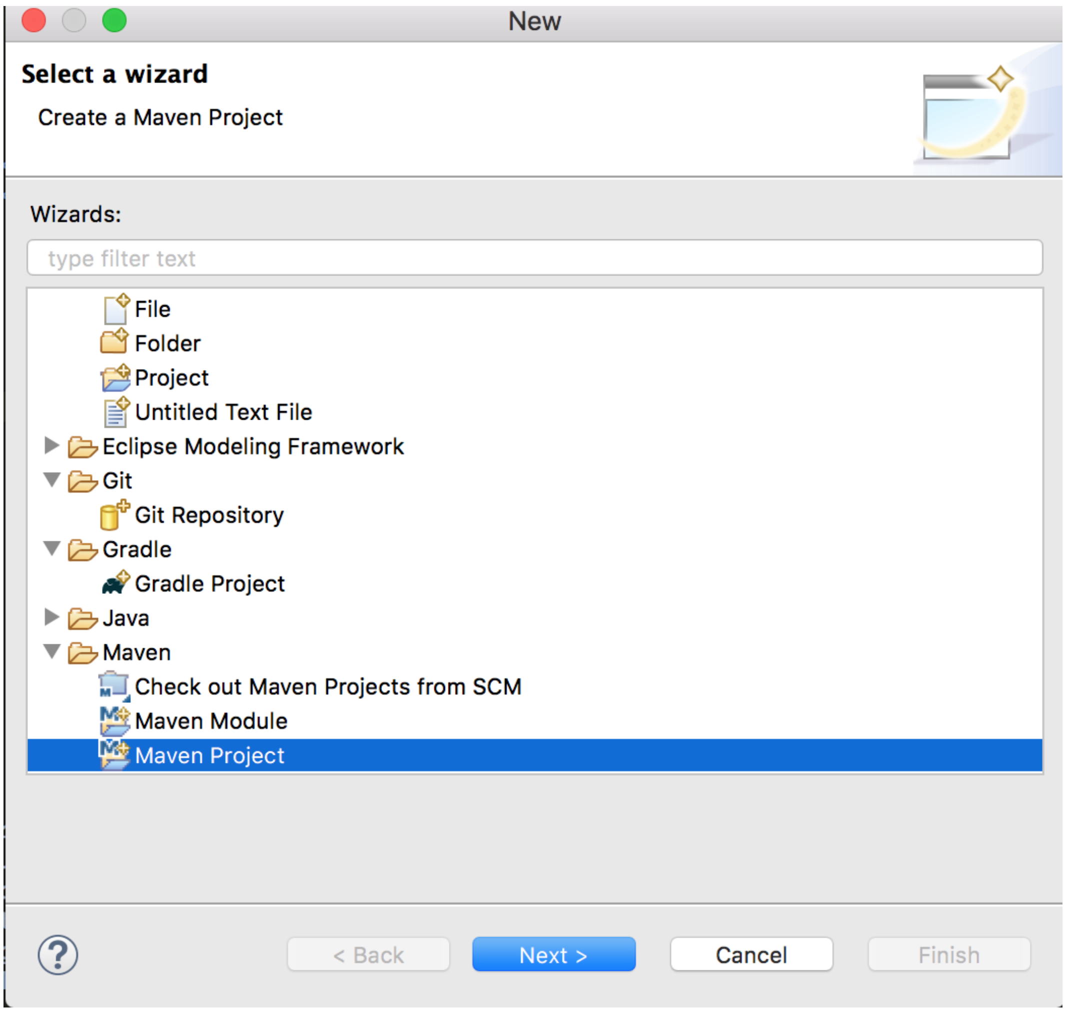
Wikipedia 2018. Java (programming language). Wikipedia [accessed 2 October 2018]. Available at: https://en.wikipedia.org/wiki/Java_(programming_language)

APPENDICES

Appendix 1: Guide to install and set up a Maven project.

From Eclipse:

Step 1: Go to File > New > Other… > Maven > Maven project > Next

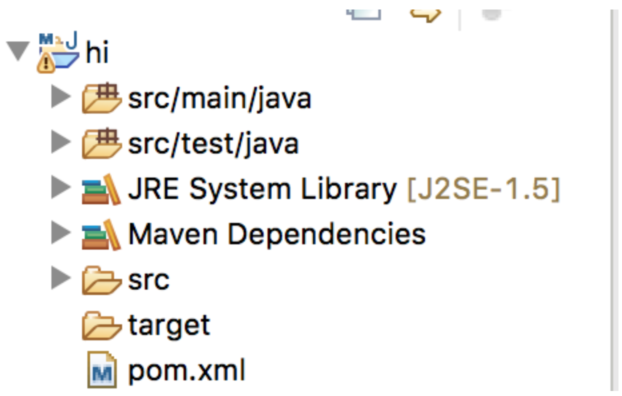


Step 2: Select the workplace folder

Step 3: Select maven-archetype-quickstart type, click Next

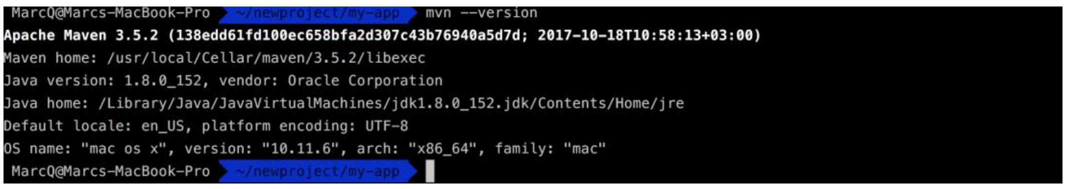
Step 4: Enter GroupID (unique name of the project) and ArtifactID (name of the jar file), click Finish.

Result: A Maven project is ready for further implementation,



From Terminal: Alternatively, you can easily create a new Maven project from the terminal or command line on Window.

Step 1: Check if Maven is installed by typing “mvn --version”. If it is installed, the result should be:



```
 MarcQ@Marcs-MacBook-Pro  ~/newproject/my-app  mvn --version
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-18T10:58:13+03:00)
Maven home: /usr/local/Cellar/maven/3.5.2/libexec
Java version: 1.8.0_152, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.11.6", arch: "x86_64", family: "mac"
 MarcQ@Marcs-MacBook-Pro  ~/newproject/my-app 
```

Step 2: Open the root folder that the Maven project should reside in, type in the Terminal this command, you can change the GroupID and ArtifactID as desired.

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

The Terminal will download and set up the Maven project.

Step 3: Now that the project is created, import it to Eclipse by importing it. (File > Import > Existing Maven project > Select the root folder). Eclipse imports the pom.xml file and displays the new project.

Appendix 2: Pom.xml file for Test Suite configuration

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.meetingpackage</groupId>
  <artifactId>TestSuite</artifactId>
  <version>0.8.1</version>
  <packaging>jar</packaging>

  <name>TestSuite</name>
  <url>http://maven.apache.org</url>

  <properties>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
 </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>3.6.0</version>
    </dependency>
  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <version>3.3</version>
          <artifactId>maven-compiler-plugin</artifactId>
          <configuration>
            <source>1.8</source>
            <target>1.8</target>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.20</version>
          <configuration>
            <trimStackTrace>true</trimStackTrace>
            <redirectTestOutputToFile>true</redirectTestOutputToFile>

          </configuration>
        </plugin>
    </pluginManagement>
  </build>
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-report-plugin</artifactId>
        <version>2.20</version>
      </plugin>
    </plugins>
  </reporting>
</project>
```