

2023

Implementing Test Automation with Selenium WebDriver

Ramana Inturi

West Virginia University, ramanakumariinturi@gmail.com

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>



Part of the [Other Computer Engineering Commons](#)

Recommended Citation

Inturi, Ramana, "Implementing Test Automation with Selenium WebDriver" (2023). *Graduate Theses, Dissertations, and Problem Reports*. 12220.

<https://researchrepository.wvu.edu/etd/12220>

This Problem/Project Report is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Problem/Project Report in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Problem/Project Report has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Implementing Test Automation with Selenium WebDriver

Ramana Kumari Inturi

Problem Report submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements

for the degree of
Master of Science
in
Electrical Engineering

Matthew Valenti, Ph.D., Chair
Natalia Schmid, D.S.
Sarika Khushalani Solanki, Ph.D.

Department of Lane Department of Computer
Science and Electrical Engineering

Morgantown, West Virginia
2023

Keywords: Automation Testing, Selenium, Java, Maven, TestNG, Page Object Model Framework,
Law Enforcement

Copyright 2023 Ramana Kumari Inturi

Abstract

Implementing Test Automation with Selenium WebDriver

Ramana Kumari Inturi

Many software programs, such as applications for designing, modeling, simulating, and analyzing systems, are now commonly available as web-based applications. The testing of such sophisticated web applications is highly challenging and can be extremely tedious and error-prone if done manually. Recently automation tools have become increasingly used for testing web-based applications, as they minimize human involvement and repetitive work.

For this problem report project, we have built and implemented an automation testing framework for web applications. The project specifically uses a tool called Selenium WebDriver, which has been used to develop the testing framework. By using this framework, testers may quickly and effectively write their test cases. The benefits of Selenium WebDriver include that it does not require in-depth research and training by testers, and due to the framework's ability to take screenshots, it provides a useful way for developers to study their code. The framework relies on the Chrome web browser, along with Java running in Eclipse, to provide a user-friendly interface for constructing and running test suites.

To validate the testing framework, we performed a case study involving NanoHub (nanoHUB.org), which is a well-known platform that provides valuable resources for those involved in nanotechnology research and education. NanoHub serves as an open-access repository for a wide range of tools, simulations, and information related to nanoscale science and engineering, and it is designed particularly to model and simulate electronic systems and nanoscale phenomena. Testing a website such as NanoHub.org typically encompasses a blend of functional testing, usability testing, and performance testing. Based on the results of this testing, several observations are made about the testing framework in general, and its application to NanoHub in particular.

The comprehensive testing approach documented in this report is aimed at ensuring the platform functions as intended, provides a user-friendly experience, and delivers optimal performance. This testing is particularly crucial when dealing with tools and simulations related to electronic systems.

Acknowledgments

This project was only accomplished with the help of many people. Without them, I would never have been able to finish this project, write this report, or achieve any of what I did. To the following people, I would like to extend my gratitude for their support, aid, and encouragement:

To Dr. Matthew Valenti for overseeing this report. He has also provided me with ongoing guidance, responded to my countless academic inquiries, and guided me toward a thorough completion.

To the WV State Police, I extend my heartfelt gratitude for your invaluable assistance in securing a graduate research assistantship at ICAC. Your warm welcome and the enjoyable work environment have been genuinely appreciated.

To Tim Ricks for suggesting and overseeing the project, and patiently guiding me along the process even when I had obvious questions.

To Natalia Schmid and Dr. Sarika Khushalani Solanki for joining my committee.

To Brianne Jankowski and Natalie Henson for their support in explaining the application's functionality and their ongoing dedication to addressing and resolving the defects I have logged.

Table of Contents

Chapter 1: Introduction.....	1
1.1 Research Background.....	1
1.2 Questions and Objectives for the Research.....	2
1.3 Research Method.....	3
1.4 Gathering and Analyzing Data.....	4
Chapter 2: Selenium Automation Testing.....	5
2.1 Software Testing Fundamentals	5
2.2 Automation Testing.....	7
2.2.1 An Explanation of Automated Testing.....	7
2.3 Selenium WebDriver.....	9
2.3.1 Selenium Overview	9
2.3.2 Selenium WebDriver Behavior	11
2.3.3 UI Locator Overview	11
2.3.4 Common Methods in Selenium.....	13
2.3.6 POM (Page Object Model) Overview	15
2.3.7 Framework and Tool Selection Reasoning	16
2.4 Java.....	17
2.4.1 Java as a Programming Language.....	17
2.4.2 Maven Overview	18
Chapter 3: Case Study	19
3.1 About NanoHub	19
3.2 Framework Development.....	20
3.2.1 Tool Versions	20
3.2.2 Test Suite Architecture.....	21
3.2.3 Testing Suite Implementation	22
3.3 TestNG	26
3.3.1 Use of TestNG in Selenium	26
3.3.2 Advantages of TestNG	27
Chapter 4: Framework Execution	28
4.1 Test Suite Execution.....	28
4.2 Analysis of the Results.....	31
4.2.1 Results and Test Reports	31
4.2.2 Comparison and Conclusion	33
Chapter 5: Conclusion	34
5.1 Answering the Research Question	34

5.2 Limitations	34
5.3 Reliability and Validity	35
5.4 Suggestions for Further Study.....	35
5.5 Summary	35
List of References	36
Electronic Sources	37
Appendix 1:.....	39
Appendix 3	42

Chapter 1: Introduction

1.1 Research Background

Web development has experienced rapid and substantial growth in recent years. As web technology becomes more approachable, there is a growing number of users who are using the Internet daily. In fact, the number of Internet users tripled between 2005 and 2022 as shown in Figure 1.1 below.

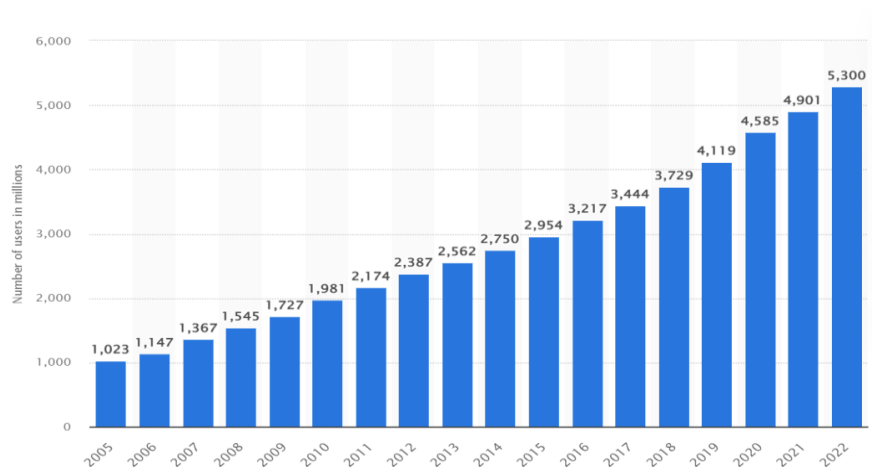


Figure 1.1 Number of internet users worldwide from 2005 to 2022 (in millions) (from Statista 2023)

Nonetheless, research has demonstrated that due to factors like subpar design, excessive complexity, or insufficient information, consumer engagement with a website diminishes within a mere 3 to 4 seconds (Spilka 2016). Evidently, creating a website to serve as a business or organization's online presence is difficult enough. A slow, unresponsive website, however, does the business no favors. In fact, having a bad website can give the impression to potential buyers that the brand is shady and unreliable (Spilka 2016).

Furthermore, modifications must be made on a daily or weekly basis to achieve fast, continuous growth. Therefore, it is crucial to guarantee that the live product does not experience

development regression or downtime. The development team is therefore faced with a challenge: how to keep a web application in good working order with frequent updates and releases in a limited amount of time?

To accomplish this, *quality assurance* (QA) is crucial to the development cycle. The development team can accelerate the feature release process by establishing a quick, dependable, and effective QA procedure. In the past, manual testing made up a sizable portion of the quality assurance process. *Automation testing* is gradually overtaking manual testing due to its capability for cross-browser testing with deep penetration depending on the test suite, though this tendency is not obvious. Automation testing is a better option for regression and time efficiency than manual testing, where the costs of time, money, and effort continue to be a significant burden.

1.2 Questions and Objectives for the Research

Automated testing involves giving computer commands to test actions that programmers have developed. But rather than having to create every module, class, and function from scratch, developers can save time by using a testing framework. To test web functionality or simulate user behavior, the Selenium Framework, for instance, is a potent tool that automates several browsers using web drivers (Selenium 2023). This report shows why and how the Selenium Framework is used in web development QA by considering the answer to the following question:

- **Research Question: How does the Selenium Framework reduce the amount of time spent on quality assurance (QA) for a specific web application?**

This study aims to assess the time efficiency that the Selenium Framework provides when applied to a test suite. To provide a concrete example, the website nanohub.org is used as a case study. The study concludes by identifying the key benefits of using the Selenium Framework for

test automation based on statistical findings. In addition, this report provides a thorough tutorial on how to use Java and the Selenium Framework to automate the testing of web applications.

1.3 Research Method

The second chapter of this report elaborates on the theory behind the chosen research strategy and methodology. In terms of research design, there are three ways according to Saunders (2023).

Figure 1.3 explains the difference between each approach:

- **Deductive approach**: Based on an acknowledged theory, the researcher analyzes a few possibilities and concludes.
- **Inductive approach**: Unlike deductive reasoning, this approach starts with the gathering of information or trustworthy standards, followed by a review of patterns, similarities, and differences, before coming up with ideas and conceptions.

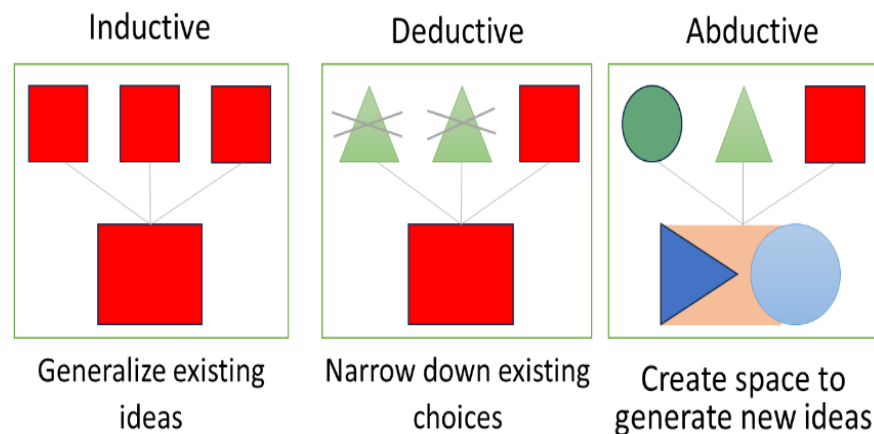


Figure 1.3 Three research approaches (AssignmentPoint 2023)

- **Abductive approach**: An established concept is either developed or revised with limited knowledge, enabling the subsequent assessment of additional information in future research.

1.4 Gathering and Analyzing Data

An essential part of any research project is data collection. The choice of data collection methods affects the research's outcome, which is just as important as choosing the research design and strategy. The researcher must choose the right purpose to have a convincing data collection. The format of the data collection and the way the data is handled are decided once it has been defined.

This report considers two methodologies, a scenario-based descriptive methodology and an observational data collection methodology. To determine the usefulness and limitations of the Framework, the data is examined afterward in a descriptive scenario and contrasted with the behavior that existed prior to the Framework's development.

Chapter 2: Selenium Automation Testing

2.1 Software Testing Fundamentals

The purpose of testing is to identify product flaws or faults in certain environments and conditions (Myers, 1979). There are a variety of key guidelines to follow when testing software. Any development must adhere to these principles to succeed, especially when using the Agile software development methodology. Prior to planning, the business must decide on its test strategy (overall testing approach) and test policy (regulations for testing). The project testing is supported by these laws and guidelines. (Graham, Veenendaal, Evans & Black 2014.)

It is obvious that developers or testers must conduct the necessary research and create a test plan first. The test aim and test parameters are included in the plan. To save time and resources, the test strategy must adhere carefully to the specific scope of each project. The project manager or test lead of the team now decides on the required hardware, software, and people resources. Additionally, risk assessment must be considered to prevent any unexpected circumstances or unwelcome challenges. The team then decides on the testing policy, the exit condition for the testing effort, and the testing approach that is best for the project. (Grahrai 2017.) Quality control is necessary, nevertheless, in case the plan takes an unforeseen turn. Testers must inform the project management in time to change the original test plan, for instance, if the testing effort takes longer than anticipated. The project manager or test lead evaluates the significance of the problem, makes the required adjustments, and informs the client. After each sprint, updates are necessary for these tasks so that the team and the client can monitor the progress. It is obvious that the test coverage must always be adhered to throughout the project to have a successful test plan. (Graham, Veenendaal, Evans & Black 2014.)

The test analysis and design process come next after the test plan. Testers choose their test scenarios and conditions after carefully observing the project's needs and conditions. For instance, the software should issue a warning when a user enters a string value into an integer field. These issues may appear minor and non-dangerous, yet they have the potential to seriously destroy the software.

Developers frequently call for the testers to have more in-depth knowledge and experience because they are uncommon in project requirements. Each demand must be as detailed as possible to prevent ambiguity or confusion on the part of the team or the clients. The design phase is where testers offer the testing environment, database, licenses, hardware, software, etc. following a thorough analysis.

Execution is part of the testing development process. Testers run thorough test scenarios and test cases at this phase. Test cases specify the desired outcome following a certain input or modification (Bath & McKay, 2008). All test cases linked to a project function or requirement are included in the following example Test Scenarios:

Functionality: Users should receive a warning if they enter the incorrect password five times or more.

- Test Scenario #1: If users input the wrong password five times or more, a warning should appear.
 - Test case #1: If Users input the wrong password five times; a warning should appear.
 - Test case #2: If Users input the wrong password six times; a warning should appear.
- Test Scenario #2: If Users input the 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.

A test suite is created by compiling every test case. The execution phase then involves running the test suite in a testing environment. The testing environment must be carefully designed after the production environment to accurately reflect the behaviors that will really occur once the software is deployed. Testers then verify the results and compile them to find any problems or errors. If the test has errors, the test case fails, but if the system is down or there are not enough resources, the test case would fail and result in an error. For developers to comprehend and adjust their codes, testers must provide a thorough description of all issues and errors. (Graham, Veenendaal, Evans & Black 2014.). The test suite's consistency must be ensured. If the test suite operates improperly and produces false findings, the testing effort becomes unnecessary. Gathering and assessing the test report is the final rule of software testing. The project manager or test lead determines whether the testing effort is sufficient, meaning the exit condition has been satisfied.

The test report and the tester's documentation (what was tested, what was not tested, what were the known issues, etc.) are typically the foundations upon which they base their decisions. The percentage of successful test cases, failed test cases, and errors is displayed in the test report. Every test attempt is followed by the recording of these statistics so that the team may monitor their performance and the project's progress (Graham, Veenendaal, Evans & Black 2014).

2.2 Automation Testing

2.2.1 An Explanation of Automated Testing

Manual testing is the practice of running test cases while interacting with people (Itkonen, Mäntylä, & Lassenius 2009). As opposed to this, developers can run a given number of test cases

using software or a machine and compare the outcomes to what was anticipated (Dustin, Garrett, & Gauf 2009). In other words, *automated testing* entails using a set of test scripts that may be run automatically (Henry 2008). Figure 2.2.1 explains the three levels of automated testing, which are as follows:

Level 1 - Unit Test: A unit test is regarded as a module test because it evaluates a single application component (Bentley 2004). Two commonly used frameworks for unit testing are JUnit for Java and NUnit for .NET. By ensuring that there are no errors in one line of the source code, JUnit and NUnit he testers by ensuring that the function provides the desired result when given a specific input. Before writing the actual code, developers can create unit tests for the functionality in agile software development (Henry, 2008).

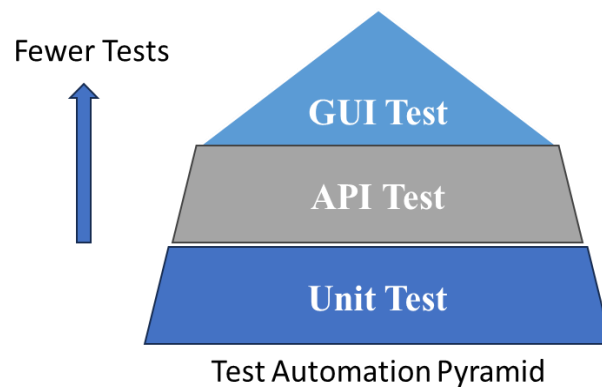


Figure 2.2.1 Test Automation Pyramid (Smartbear 2023)

Level 2 - API Test: API testing refers to the idea of directly testing APIs to ascertain their dependability, performance, security, and most crucially - functionality (Crispin & Gregory 2008). APIs are a set of functions that return a collection of values after being called or interacting with the user interface. API testing now assists developers and testers in keeping up with short sprints and a quick rate of development (Henry 2008).

Level 3 - GUI (Graphical User Interface) Test: Testers can command the computer to imitate any repetitive operations that an end user could complete (manual testing) if the application includes a GUI. The GUI test puts a greater emphasis on the end-user perspective because it can

identify incorrect designs, missing, or masked components, in addition to functional issues. Unit or API tests cannot identify such issues. When creating GUI test cases, it is crucial to record and have a fallback because even minor GUI changes can cause the test case to fail (Henry 2008).

2.2.2 When to Perform Automation Testing

Prior to any sprint or development, it is good practice to analyze and design the testing approach. Regression testing, which involves testing an existing product and newly added or modified functionalities, is a promising application for test automation. To guarantee that the updated version or changes do not affect the previous versions, regression testing is helpful. Additionally, it is advisable to automate any repeated tests that are unlikely to change during the development sprint or cycle. These tests are typically very data-driven, meaning the same function can be entered with a wide variety of data or input.

2.3 Selenium WebDriver

2.3.1 Selenium Overview

Jason Huggins created the open-source Selenium software testing framework in 2004 and released it under the Apache 2.0 license (Selenium 2023). By managing the browser web driver, it serves to simulate end-user interaction with a web application. Most of the current generation's most common web browsers are supported by Selenium, and the test suite can be developed in a variety of languages (Selenium 2023). The Selenium Framework now consists of four components:

- Selenium IDE: A Firefox add-on that integrates the Selenium Core enables testers to record and replay interaction steps on a web page. It has a context menu with a list of assertions and verification points (Selenium 2023).
- Selenium Core: A JavaScript-based program that may be used to execute test scripts written in Selenese (a language comprised of commands). It must be installed on

the testing server, which is off-limits to testers (Selenium 2023).

- Selenium WebDriver: This is the most widely used tool for web application test automation. Figure 2.3.1 shows how Selenium WebDriver's API enables programmers to create and execute a whole suite of tests. The program controls the browser without requiring a connection to a remote server, unless the target system is not the native one, and passes the Selenese test script to the Selenium Core based on various test cases. This shortens the connection time and accelerates the test run-time as a result (Avasarala 2014, 13).

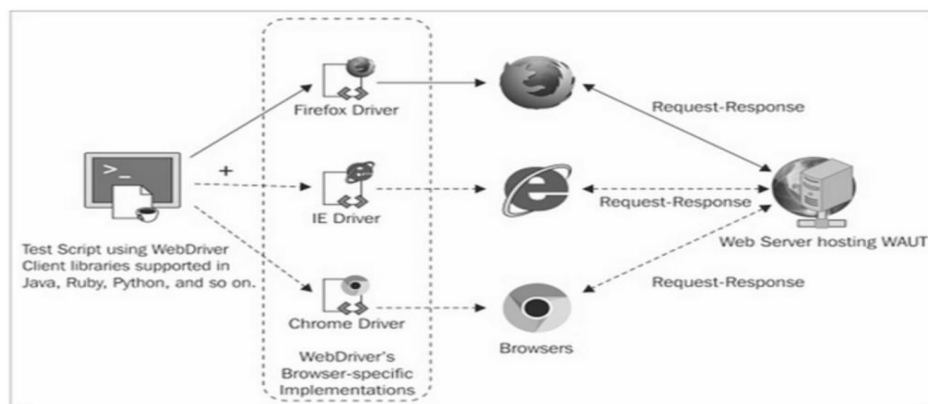


Figure 2.3.1 Interaction of WebDriver (Avasarala 2014, 13)

- Selenium Grid: This gives users the option to run tests simultaneously across several computers using different operating systems and browsers (Selenium 2023).

2.3.2 Selenium WebDriver Behavior

Figure 2.3.2 provides a detailed breakdown of the WebDriver behavior.

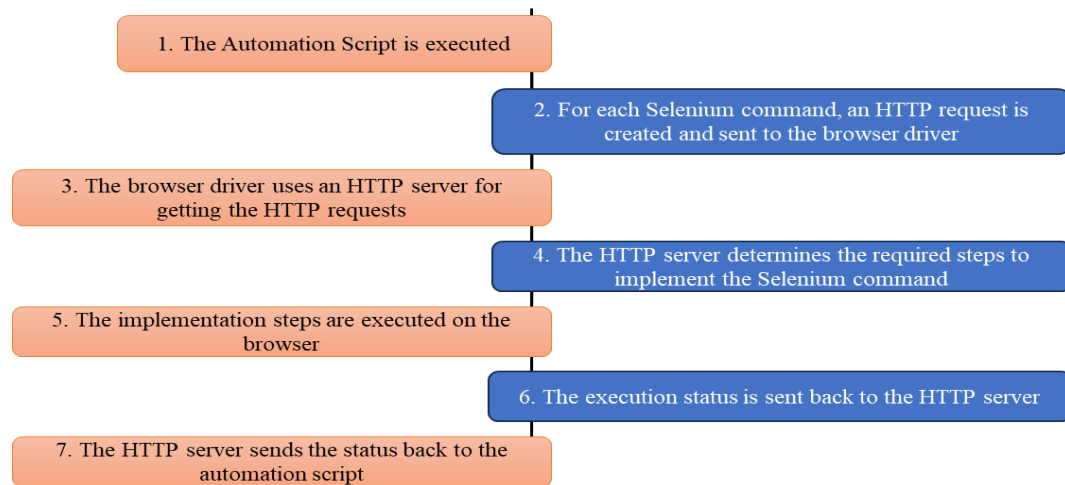


Figure 2.3.2 Demonstration of Selenium WebDriver operation (Quora 2023)

A broad variety of client libraries are supported by Selenium WebDriver, including those written in PHP, C# (NUnit), Java (TestNG, JUnit), Perl, Ruby (RSpec), and Python (Robot, PyUnit, unittest) (Selenium 2023). Additionally, it supports the following modern web browsers: Internet Explorer, Chrome, Firefox, Safari, and PhantomJS (a headless browser) (Selenium 2023).

2.3.3 UI Locator Overview

The Selenium WebDriver refers to the body, input, iframe, and other HTML components that are present on a webpage as *Web Elements*. WebDriver may simply carry out operations and connect with others by locating these elements (Selenium 2023). Because of this, it is crucial that the testers correctly identify the element involved in each test case; otherwise, the test case will fail because it cannot be located (Selenium 2023). Thanks to the clever WebDriver API, there are several ways to find the UI locators based on the various *DOM (Document Object Model)* elements on the web page (tag name, class name, CSS Selector, XPath, ID, Name, LinkText) elements (Selenium 2023). If more than one element matches and is found after a DOM query, the system

throws an error unless it is kept in an array (Selenium 2023). Following is the syntax of UI locators (Selenium 2023):

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="redyellow">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 elements on the DOM and below is an example:

For example: `By.xpath("//input");`

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

2.3.4 Common Methods in Selenium

Table 2.3.4 below lists the Selenium WebDriver methods that are used most frequently:

Method	Command
init webdriver	<code>WebDriver driver = new ChromeDriver();</code>
open url	<code>driver.get("google.com");</code>
init webElement	<code>WebElement button1 = driver.findElement(By.id("button"));</code>
click an element	<code>driver.findElement(By.id("button")).click();</code>
enter text	<code>driver.findElement(By.id("textbox")).sendKeys("Hello");</code>
refresh the page	<code>driver.navigate().refresh();</code>
navigate	<code>driver.navigate().forward();</code> or <code>driver.navigate().back();</code>
drag and drop	<code>WebElement element = driver.findElement(By.name("origin"));</code> <code>WebElement target = driver.findElement(By.name("new"));</code> <code>(new Actions(driver)).dragAndDrop(element,new).perform();</code>
get text	<code>driver.findElement(By.id("textbox")).getText();</code>

Table 2.3.4 Common methods and commands in Selenium (Selenium 2023)

2.3.5 Selenium Installation System Requirements

There are a few necessary system prerequisites (Selenium 2023) to have a running Java environment ready for developing the Selenium test suite:

- Java Runtime Environment (JRE) on the native computer: The Java Software Development Kit (JDK) has integration for JRE.

- Users can choose the operating system they are using in Figure 2.3.5(a) when downloading and installing JDK from Oracle's official website for Java (Oracle 2023).

Java SE Development Kit 11		
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> 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 2.3.5(a) List of supported JDK (Oracle 2023)

- Install Eclipse IDE (Integrated Development Environment) at Eclipse's official website (Eclipse 2023), as shown in Figure 2.3.5(b):

Figure 2.3.5(b) Eclipse Download Page (Eclipse 2023)

- Browser Driver: To run test scripts on a certain browser, developers must install the matching browser driver and configure the path pointing to the location of the drivers.
- The common browsers and their access studying drivers are listed in Figure 2.3.5(c) below:

After all the required components are installed, a complete guide to set up a Maven project (mentioned in 2.4.2) can be found in Appendix 1.

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 2.3.5(c) Browsers and the corresponding drivers (guru99 2023)

2.3.6 POM (Page Object Model) Overview

POM has been established as a design pattern for easier maintenance and code recycling, according to Selenium (Selenium 2023).

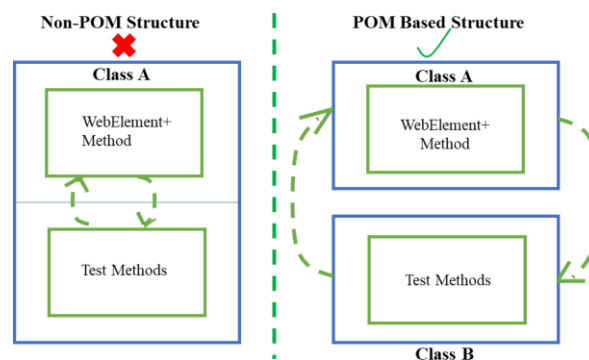


Figure 2.3.6 POM vs Non-POM comparison (Guru99 2023)

Each page in POM is built as an object-oriented class, allowing tests to interact with the page by using the class's methods (Figure 2.3.6). The test itself does not change when anything on

the page does, so the class material needs to be updated. It is simple to locate and update new characteristics at any point in the development because the entire content of the page is contained in a single file. Furthermore, since the page class is independent of the tests, different test kinds (such as acceptance tests, behavior-driven tests, and data-driven tests) can all use the same page class. Finally, methods have realistic names, which makes it easier for developers to read and maintain (Selenium 2023).

2.3.7 Framework and Tool Selection Reasoning

When it comes to automation testing, there are surely a lot of frameworks and testing tools accessible. Selecting tools that can be useful in terms of deployment and cost-effectiveness is essential. It must be chosen considering the resources and circumstances of the business. The problem report discusses his selection of the tool in this section, along with a comparison to comparable tools available on the market.

According to the needs and circumstances of the firm, there are a few factors to consider when deciding which automation testing framework to use:

The instrument must be a top pick for GUI testing: For instance, when the product is a desktop web application, end-to-end testing, often known as GUI testing, is crucial since it directly influences user behavior.

Functional testing is supported by the tool. A well-designed product should make a POM design simple to apply. Cross-browser testing needs to be supported by the tool.

The tool must be simple to use without consuming a lot of resources, hence it should be developed independently from the team's source code. The tool must be affordable. Ideally, the tool should be an open-source framework.

The tool must offer an easy-to-read test report with specific statistics that may be saved for further use.

The various automated testing frameworks were then narrowed down by the report, who subsequently compared them in Table 2.3.7:

	Selenium	Robot Framework	Appium	Auto IT
Popularity	High	Medium	Medium	Medium
POM design	Yes	No	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

Table 2.3.7 Comparisons between shortlisted frameworks

The Selenium Framework appears to be the wisely chosen tool for test automation after careful evaluation considering the situation. Java is chosen as the programming language due to its reliability and support for POM design, both of which are covered in greater detail in the following chapter.

2.4 Java

2.4.1 Java as a Programming Language

Sun Microsystems created Java in 1995, which was used for computing across several platforms and as a programming language. Java allows programmers to write code once and run it anywhere because it is a compiled language (Techopedia 2023). According to Wikipedia (as of

2023), Java had 9 million developers and was the most widely used programming language up to 2016. It also has extensive documentation and community support. A Java software development kit (SDK), which includes a compiler, interpreter, and documentation generator, is required for the creation of Java programs (Techopedia 2023). The language itself comes with a great IDE and tools. Java is a strong contender for test automation along with TestNG for unit testing.

2.4.2 Maven Overview

Maven is a Java build management tool to define how the .java files are compiled to .class and much more (Maven 2023). As a result, Maven was developed to indicate a standard way to automatically download each of the necessary libraries listed in the pom.xml file (which stands for Project Object Model) to create the project. An example of a Maven-run project structure is shown in Figure 2.4.2 below.

All the project's necessary dependencies, such as Junit or the Selenium-Java, are listed in the pom.xml file. It is formatted in XML. The requisite "compile", "test", "package", and "clean" steps are unnecessary with Maven because the file executes as expected (Maven



Figure 2.4.2 Project Structure with Maven (Maven 2023)

2023). There are numerous plug-ins to be used in test reports, particularly for automation testing, and running the test suite with Maven is simple with Maven commands (to be covered in Chapter 4).

Chapter 3: Case Study

3.1 About NanoHub

To validate the testing framework, we performed a case study involving NanoHub (nanoHUB.org). NanoHub serves as a free open-access repository for a wide range of tools, simulations, and information related to nanoscale science and engineering, and it is designed particularly to model and simulate electronic systems and nanoscale phenomena. In the domains of nanotechnology, materials science, and other related disciplines, *NanoHub* is a platform for computational education, research, and collaboration. It houses an expanding selection of cloud-based simulation tools that are openly available via a web browser. NanoHUB also has thousands of other resources, such as seminars, courses, presentations, and teaching aids, in addition to these tools. These documents provide users with information on our simulation tools as well as broader nanoelectronics, materials science, photonics, and other subjects.

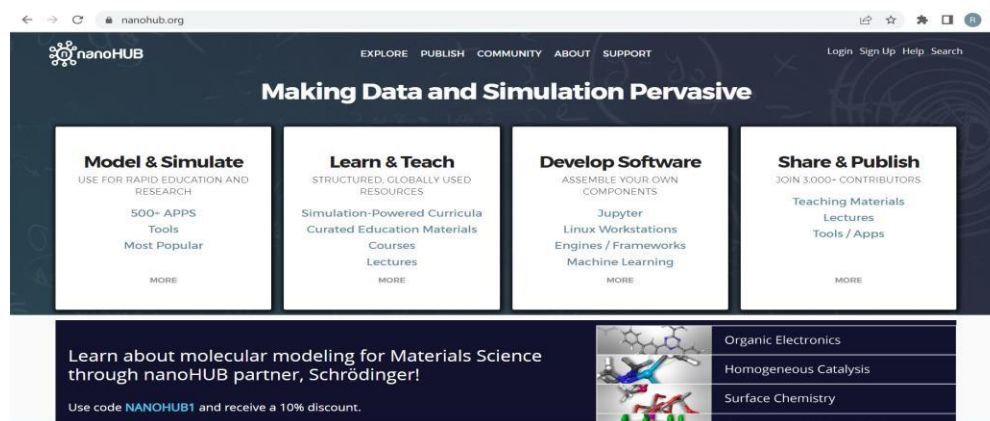


Figure 3.1 nanoHub.org Home Page (nanoHub.org 2023)

Testing a website such as NanoHub.org typically encompasses a blend of functional testing, usability testing, and performance testing. Based on the results of this testing, several observations are made about the testing framework in general, and its application to NanoHub in

particular. The comprehensive testing approach documented in this report is aimed at ensuring the platform functions as intended, provides a user-friendly experience, and delivers optimal performance. This testing is particularly crucial when dealing with tools and simulations related to electronic systems.

3.2 Framework Development

3.2.1 Tool Versions

The goal of the test suite is to take the place of the case study's present functional *UAT* (UAT - Software is tested in the real world by its intended end users during the user acceptance testing (UAT) phase of development. UAT is sometimes referred to as application testing or end-user testing.) A Windows computer is used to execute the test suite. The test suite is installed using the common installation described in section 2.3.5. The test suite's components include the following:

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

Chrome is only being used as an example because Selenium supports a broad variety of web browsers. The developer can simply download the necessary browser and other web drivers to run the test. The web drivers must, however, be saved in the same folder as the test suite; otherwise, the path associated with the web driver must be specified.

- When all the necessary Selenium, Java, and TestNG dependencies are included in the pom.xml file, it downloads all the required libraries to run the Java program, and the versions are mentioned in (Table 3.2.1).

- The Framework's pom.xml is described in detail in Appendix 2

Name	Version	Description
Selenium-Java	4.10.0	Selenium version for Java
Java JDK	1.8.0	Java environment
Chrome Browser	112.0.5615.121	Chrome browser to generate an instance
TestNG	7.7.1	Java unit testing framework
Maven	10.0	Java build management tool

Table 3.2.1 System requirements to run the test suite.

3.2.2 Test Suite Architecture

The architecture of the test suite is demonstrated using the following Figure 3.2.2:

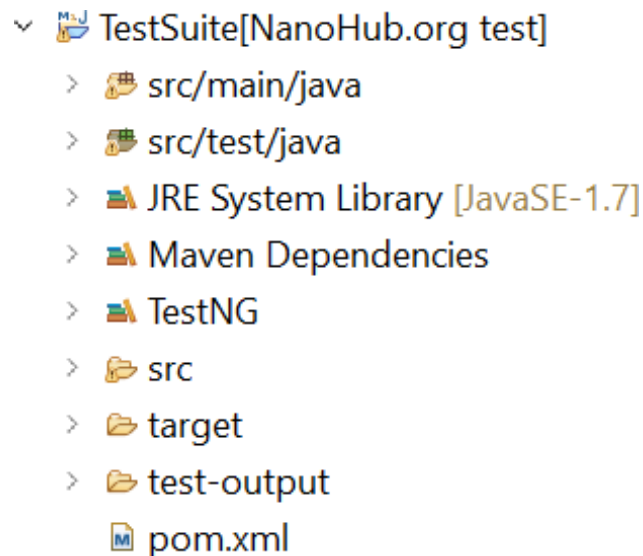


Figure 3.2.2 Architecture of the test suite

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

3.2.3 Testing Suite Implementation

However, a demonstration of a straightforward test case and its application is provided and below for the purposes of this report:

Test case: User logs in successfully with correct credentials on NanoHub.org

Precondition: User using Chrome browser. Steps:

Step 1: Go to NanoHub.org

Step 2: Go to the login page (Figure 3.2.3(a))

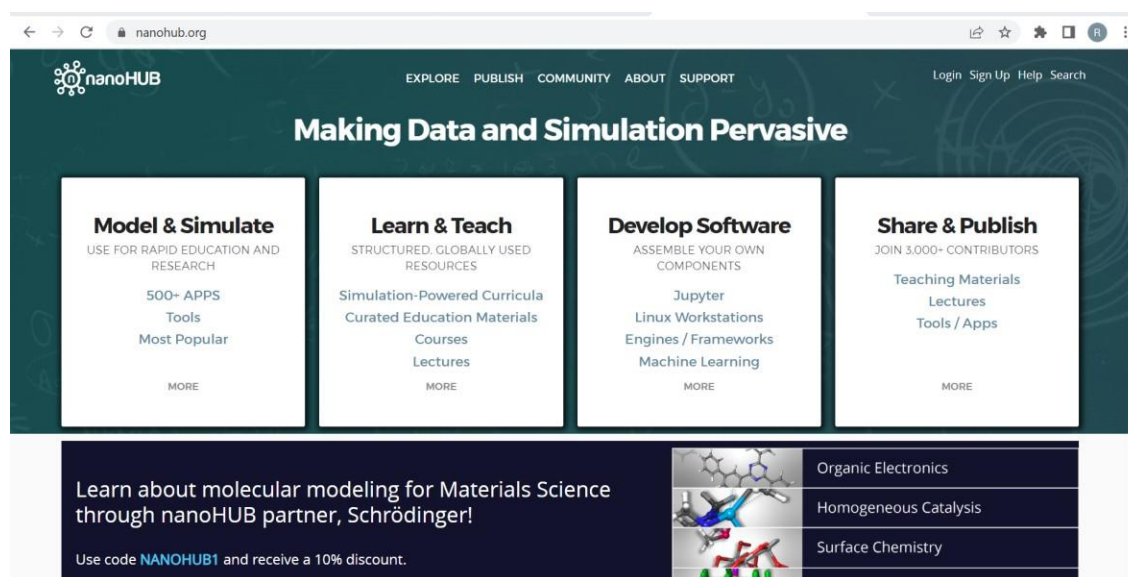


Figure 3.2.3(a) NanoHub.org Login Page (NanoHub.org 2023)

Step 3: Click on ‘Sign in with your nanoHub.org account’ and then enter the user’s credential (Figure 3.2.3(b)).

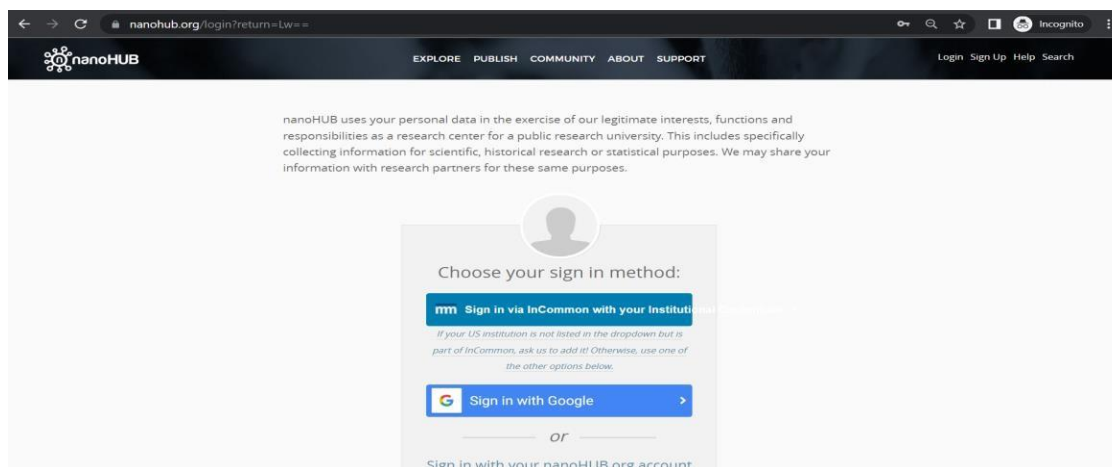


Figure 3.2.3(b) NanoHub.org Login Page (NanoHub.org 2023)

Step 4: Click the login button.

Expected results: The user's homepage is open indicating the login is successful (Figure 3.2.3(c)):

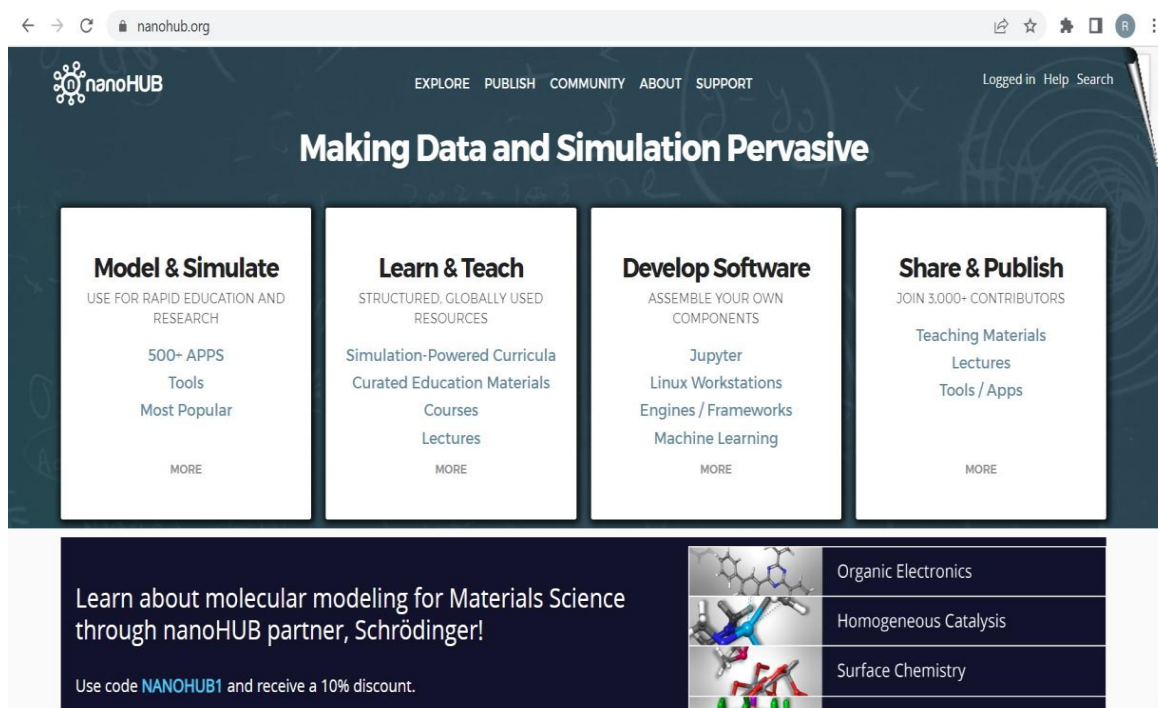


Figure 3.2.3(c) NanoHub.org Dashboard Page (NanoHub.org 2023)

The creation of the Login Java page is the first step in the test case implementation.

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import io.github.bonigarcia.wdm.WebDriverManager;
```

Figure 4.1.3(d) Required packages.

All required libraries, including By and WebDriver from Selenium, are imported by the login page (Figure 3.2.3(d)).

The developer then locates the UI locator on the DOM using the ID of the locator and passes that information to the variables (Figure 3.2.3(e)).

```
public class LoginPage {
    WebDriver driver;

    public LoginPage(WebDriver driver)
    {
        this.driver = driver;
    }

    By Login_btn = By.className("user-account-link loggedout");
    By SignInWithDifferentAccount = By.xpath("//*[@contains(text(),'Sign in with a different account')]");
    By SignInWithNanoHubAccount = By.xpath("//*[@contains(text(),'nanoHUB.org account')]");
    By Username = By.id("username");
    By Password = By.id("password");
    By SignIn_btn = By.className("login-submit btn btn-primary");

    public void ClickOnLogin() {
        driver.findElement(Login_btn).click();
    }

    public void ClickOn_SignInWithDifferentAccount() {
        driver.findElement(SignInWithDifferentAccount).click();
    }

    public void ClickOn_SignInWithNanoHubAccount() {
        driver.findElement(SignInWithNanoHubAccount).click();
    }

    public void EnterValidUserName() {
        driver.findElement(Username).sendKeys("ramana_inturi");
    }

    public void EnterValidPassword() {
        driver.findElement>Password).sendKeys("Ajay@1464");
    }
}
```

Figure 3.2.3(e) Selenium Login Page

Finally, a function is put into use. The Driver uses the UI locator to find the login and password textbox and uses Selenium's sendKeys function to send the required credentials.

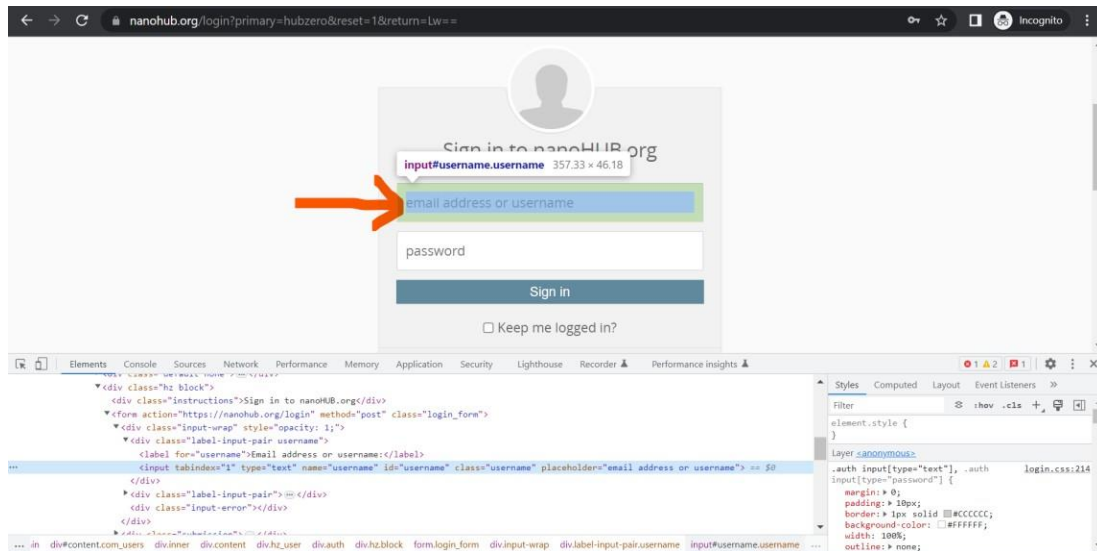


Figure 3.2.3(f) NanoHub.org Login Page (NanoHub.org 2023)

To obtain the UI location, the developer can examine the element in the Chrome browser (Figure 3.2.3(f)). The *Before* function in the test case indicates what the driver should perform before running any tests. Here, the driver must be opened, the dimensions must be changed, and the homepage of NanoHub.org must be reached. This results in a runtime fault if it is not executed (Figure 3.2.3(g)).

```
@BeforeTest
public void beforetest() {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();

    driver.manage().timeouts().implicitlyWait(0, TimeUnit.SECONDS);

    driver.get("https://nanohub.org/");
    driver.manage().window().maximize();
}
```

Figure 3.2.3(g) 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.

```
Run All
public class LoginTests extends BaseTest{

    /*Verify that users can log in with valid credentials without any errors. */
    @Test
    Run | Debug
    public void VerifyValidLogin() throws InterruptedException {
        LoginPage page = new LoginPage(driver);
        page.EnterValidUserName();
        page.EnterValidPassword();
        page.VerifyLogin();
    }

    /*Verify that users can log in with Invalid credentials without any errors. */
    @Test
    Run | Debug
    public void VerifyInvalidLogin() throws InterruptedException {
        LoginPage page = new LoginPage(driver);
        page.EnterInvalidUserName();
        page.EnterInvalidPassword();
        page.VerifyLogin();
    }
}
```

Figure 3.2.3(h) Selenium Login Page

3.3 TestNG

TestNG is an open-source platform for Java test automation. It was created in a similar manner to how JUnit and NUnit were. TestNG is a more durable framework than its competitors thanks to the few cutting-edge and practical features it offers. "Next Generation" is the meaning of the NG in TestNG.

3.3.1 Use of TestNG in Selenium

The fact that Selenium lacks a suitable format for the test results is one of its shortcomings. You can do the following with Selenium's TestNG framework:

- Create the report using the correct format.
- Include in the report the number of test cases executed, as well as the tests that were succeeded, failed, and skipped.

- Converting test cases to testing.xml will group them.
- Use invocation count and run numerous tests devoid of loops.

3.3.2 Advantages of TestNG

TestNG provides many options for both test configuration and execution, which is an advantage. You can create tests with TestNG, for example, to execute them simultaneously, in a specific order, and with different data sources. Apart from presenting test outcomes, errors, and issues, TestNG generates HTML reports that provide detailed data on test execution.

Chapter 4: Framework Execution

4.1 Test Suite Execution

For this case study, eight use cases have been created (Figure 4.1(b)). An average use case contains three test cases, and a single test case takes two minutes to perform for an automation developer.

The test suite can be executed using TestNG depending on the scope of the test: A single test case can be executed by right-clicking on the test case, choosing “Run As”, and selecting “TestNG test” (Figure 4.1(a)).

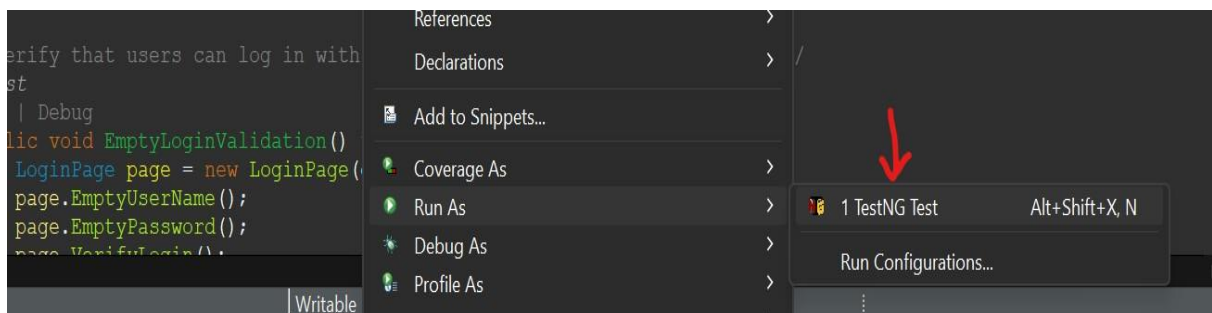


Figure 4.1(a) Selenium Login Test

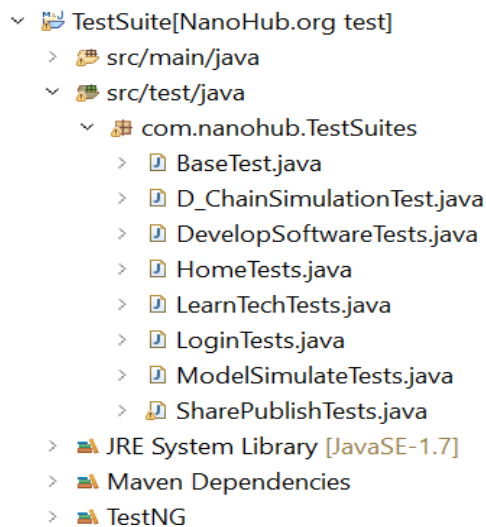


Figure 4.1(b) Selenium Login Test

Following execution, the software launches the test case and displays a notification on the console. (Figure 4.1(c)).



```

Javadoc Declaration Console Results of running class DuplicateTests
<terminated> DuplicateTests [TestNG] C:\Program Files\Java\jdk-17\bin\javaw.exe (Oct 7, 2023, 6:34:53 PM - 6:35:05 PM) [pid: 8732]
[RemoteTestNG] detected TestNG version 7.4.0
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Oct 07, 2023 6:34:54 PM org.openqa.selenium.remote.service.DriverService$Builder getLogOutput
INFO: Driver logs no longer sent to console by default; https://www.selenium.dev/documentation/webdriver/drivers/service/#setting-log-output
  
```

Figure 4.1(c) Selenium Login Test

A Chrome Browser instance is generated, and the test is run in accordance with the test script. The program notifies the user that the driver is operational with the prompt "Chrome is being controlled by automated test software." (Figure 4.1(d)).

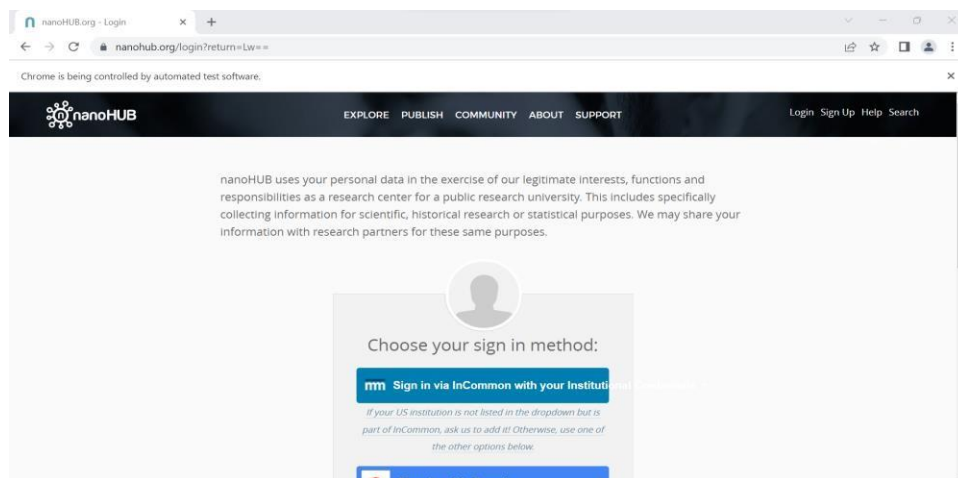


Figure 4.1(d) Selenium Browser Instance

One test scenario with several test cases: Right-click the test scenario, select *Run As*, and then choose *TestNG* to execute the test scenario.

Each test scenario is run individually (Figure 4.1(e)).

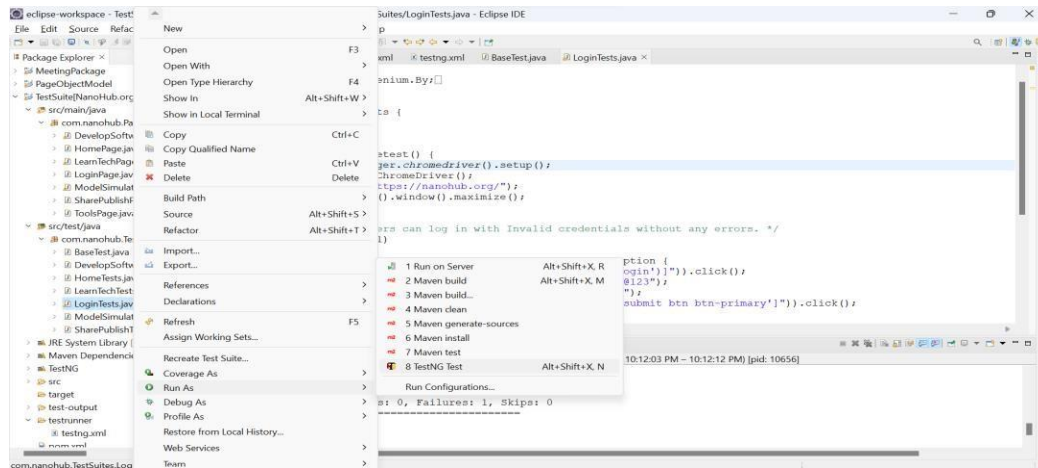


Figure 4.1(e) Selenium Test Suite

Maven executes the test suite because of compiling and running from the first test scenario.

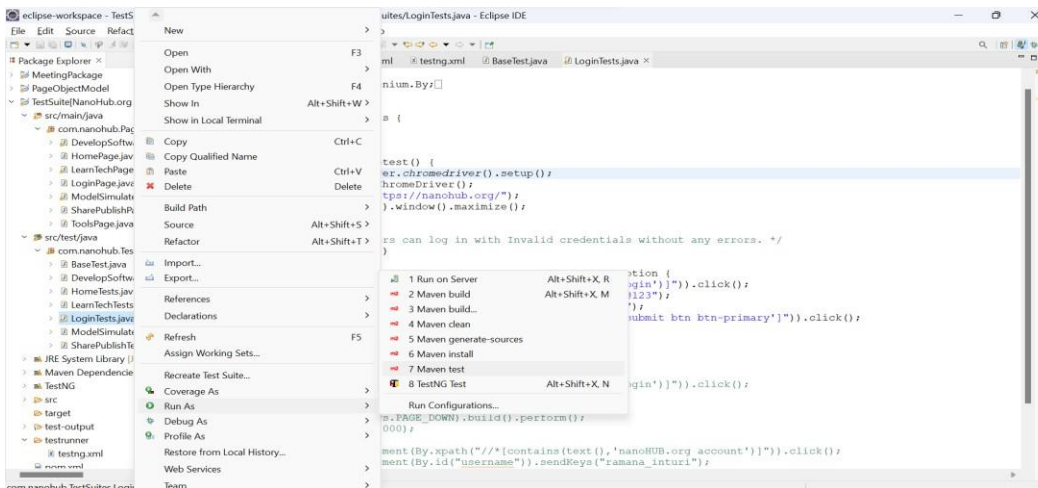


Figure 4.1(f) Selenium Test Suite

The test results may later be compiled in the Destination folder. The pom.xml file can also be opened by right-clicking it, choosing Run As, and then selecting Maven test (Figure 4.1(f)).

4.2 Analysis of the Results

4.2.1 Results and Test Reports

The pom.xml plugin is used to provide presentable HTML and CSS test reports. The test report may be found in your project testsuite>/test-output>/index.html after that. The report's summary shows how many test cases were executed, their success rates, errors, and execution times (Figure 4.2.1(a)):

The screenshot displays a web browser window showing a test report. The browser's address bar indicates the file path: `C:/Users/Ramana%20kumari%20Inturi/eclipse-workspace/TestSuite%5BNanoHub.org%20test%5D/test-output/index.html#`. The report header shows "1 suite, 5 failed tests" and a "Switch Retro Theme" button. The left sidebar, titled "nanoHub Test Suite", provides an overview of the test results, including a summary: "16 methods, 5 failed, 4 skipped, 7 passed". The main content area lists the following failed test categories and their corresponding error messages:

- com.nanohub.TestSuites.D_ChainSimulationTest**: VerifyToolsForDevelopment. Error: `java.lang.IllegalArgumentException: Driver must be set` at `org.openqa.selenium.internal.Require.nonNull(Require.java:68)`.
- com.nanohub.TestSuites.ModelSimulateTests**: VerifyModellingTools. Error: `java.lang.NullPointerException: Cannot invoke "org.openqa.selenium.WebDriver.findElement(org.openqa.selenium.By)" because "this.driver" is null` at `com.nanohub.TestSuites.ModelSimulateTests.VerifyModellingTools(ModelSimulateTests.java:19)`.
- com.nanohub.TestSuites.DevelopSoftwareTests**: VerifyDevelopSoftware. Error: `java.lang.NullPointerException: Cannot invoke "org.openqa.selenium.WebDriver.findElement(org.openqa.selenium.By)" because "this.driver" is null` at `com.nanohub.TestSuites.DevelopSoftwareTests.VerifyDevelopSoftware(DevelopSoftwareTests.java:21)`.
- com.nanohub.TestSuites.SharePublishTests**: VerifySharePublish. Error: `java.lang.NullPointerException: Cannot invoke "org.openqa.selenium.WebDriver.findElement(org.openqa.selenium.By)" because "this.driver" is null` at `com.nanohub.TestSuites.SharePublishTests.VerifySharePublish(SharePublishTests.java:23)`.
- com.nanohub.TestSuites.LearnTechTests**: VerifyLearnTech. Error: `java.lang.NullPointerException: Cannot invoke "org.openqa.selenium.WebDriver.findElement(org.openqa.selenium.By)" because "this.driver" is null` at `com.nanohub.TestSuites.LearnTechTests.VerifyLearnTech(LearnTechTests.java:18)`.

Figure 4.2.1(a) Test report

Additionally, the developer can see how each test scenario is performed in the package list (Figure 4.2.1(b)):

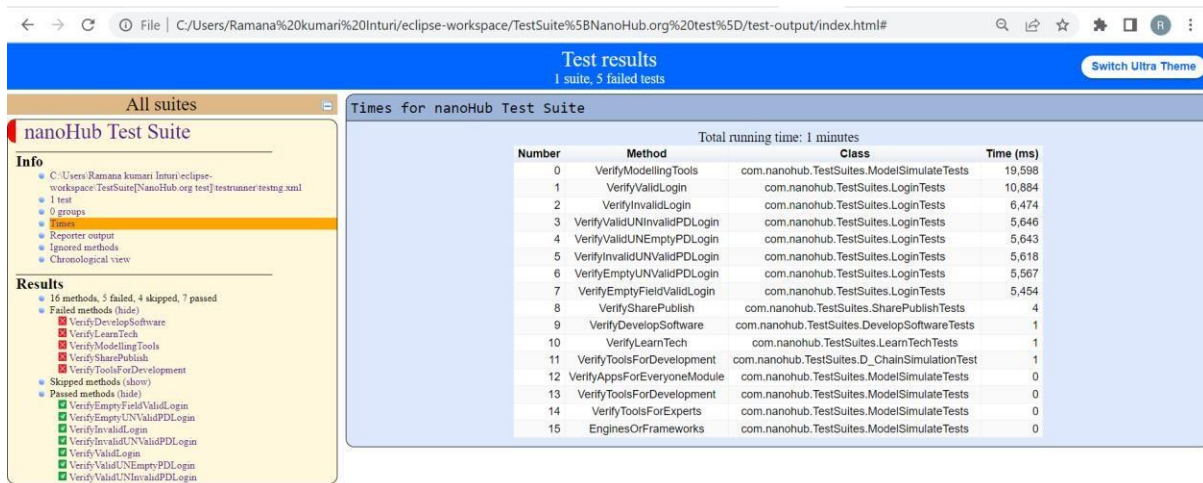


Figure 4.2.1(b) Test report

The purpose of testing is to identify any errors or defects in the case study. Three defects were discovered during the automation of the nanoHub web application, which is listed in Appendix 3.

The test case section provides information on the causes of any faults or failures.

As an alternative, the following outcome can be seen from the Eclipse console when running a single test case or a scenario without utilizing Maven (Figure 4.2.1(c)):

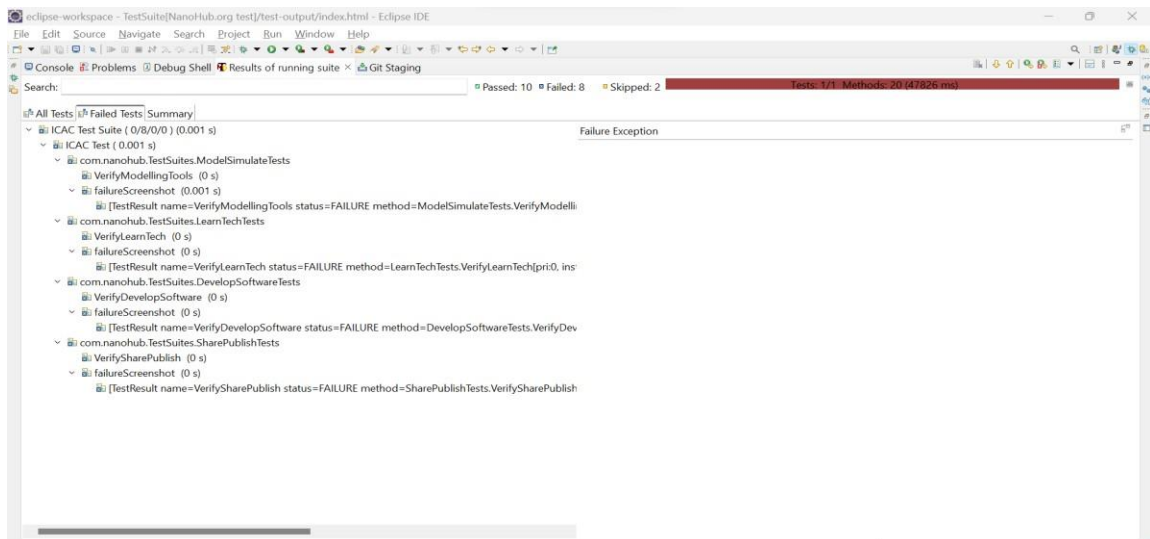


Figure 4.2.1(c) TestNG

Developers can right-click to enter the problematic code by navigating to it using the failure trail to assist them in identifying the issue. An error is generated by the system, and a message is printed along with all relevant files.

4.2.2 Comparison and Conclusion

After taking the previous (Figure 4.2.1(b)) indicated amount of time to complete the functional testing. Given that one developer must complete the entire functional testing process four times during each of the four sprints that make up the two months of development (called a sprint), Equation 1 describes a technique of estimating the amount of time that person must devote to development and testing:

$$\text{Total time consumption (Hours)} = \text{Development time} + (\text{time for 1 round of testing} + \text{maintenance time}) * \text{rounds of testing} * \text{number of sprints}$$

Equation 1 Time consumption for testing

The test suite has clearly had a significant impact on the testing effort based on the time savings. Although it initially requires some work to implement, later, the developer is entirely free while the test suite is running. It follows that the benefit increases in direct proportion to the amount of effort and testing required for development.

Chapter 5: Conclusion

5.1 Answering the Research Question

This problem report aims to demonstrate the business or organization benefits from automation testing using the Selenium Framework. The problem report discussed the information and methods required to support decisions made and actions taken in the theoretical section. The problem report concludes by responding to the research question using the information gathered following the implementation of the artifact:

- **How does the Selenium Framework reduce the amount of time spent on quality assurance (QA) for a specific web application?**

In simple terms, a decrease in time consumption based on the results of the data analysis successfully demonstrates that the artifact had a significant effect on NanoHub.org during the development process.

To ensure that the results are as accurate as possible, the comparison was conducted under identical circumstances. The development team can use the artifact to accelerate delivery, enhance productivity, and save a significant amount of time and money.

5.2 Limitations

The problem report compares the amount of time required for the test suite's design and execution. It is possible to compare different parameters as well, such as the complexity of the development, and the ability to maintain the framework. Therefore, it is advised to consider more factors in determining the artifact's effectiveness.

5.3 Reliability and Validity

If the results of the automation tests contain any error test cases, it signifies that the test case was unable to run due to an internal issue. As a result, once the issue is fixed, the test suite's time requirement can go up. As a result, the reliability may not be totally accurate, but the artifact's overall advantage is still, without a doubt, greater than 65%. Finally, there may be some more factors that have an impact on the study, such as the problem report's expertise, the size of the development team, the chosen solution, or the programming language.

5.4 Suggestions for Further Study

Since a web application is the primary output of the case study, website test automation is the research's focus. It is possible for more than two test cases to be executed simultaneously while using Java, as the developer can run the test cases concurrently. Building operations take less time. Test case synchronization and design structure for simultaneous builds are areas that require more research.

5.5 Summary

In conclusion, test automation, especially for regression and functional testing, may initially require time and resources, but in the long term, its advantages are prominent for any business or organization.

List of References

- 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.
- 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 Crispin 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.
- 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.
- Myers, 1979. G Myers. The Art of Software Testing. John Wiley & Sonc, Inc., New York.
- 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.
- Schwaber, K. & Beedle, M. 2002. Agile Software Development with Scrum. Upper Saddle River, NJ: Prentice-Hall.

Electronic Sources

- Boer, G. 2023. What is Scrum? Microsoft [accessed 2 October 2023]. Available at: <https://learn.microsoft.com/en-us/devops/plan/what-is-scrum>
- Eclipse. 2023. Download Eclipse Technology which is right for you. Eclipse [accessed 2 October 2023]. Available at: <https://www.eclipse.org/downloads/>
- Geerts, G.L. 2011. International Journal of Accounting Information Systems. Elsevier [accessed 3 October 2023]. Available at: <https://www.sciencedirect.com/science/article/pii/S1467089511000200>
- Guru99. 2023. How to Download & Install Selenium WebDriver. Guru99 [accessed 3 October 2023]. Available at: <https://www.guru99.com/installing-selenium-webdriver.html>
- Guru99, 2023. Page Object Model (POM) & Page Factory in Selenium: Guru99 [accessed 3 October 2023]. 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? testingguru.com [accessed 3 October 2023]. Available at: <https://www.testing.guru/what-is-fundamental-test-process-in-software-testing/>
- Gurendo, D. 2015. Software Development Life Cycle (SDLC). Scrum Model Step by Step. XB Software [accessed 3 October 2023]. Available at: <https://xbssoftware.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 2023]. Available at: <https://www.seguetech.com/waterfall-vs-agile-methodology/>
- Maven 2023. Maven in 5 minutes. Maven [accessed 3 October 2023]. Available at: <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
- Wikipedia 2023. Java (programming language). Wikipedia [accessed 2 October 2023]. Available at: [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
- Statista 2023. Number of internet users worldwide from 2005 to 2022 (in millions). Statista [accessed 2 October 2023]. Available at: <https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/>
- Selenium 2023. What is Selenium? Selenium [accessed 2 October 2023]. Available at: <https://www.seleniumhq.org/>
- Assignmentpoint 2023. About Abductive Reasoning. Assignmentpoint [accessed 2 October 2023]. Available at: <http://www.assignmentpoint.com/science/mathematic/about-abductive-reasoning.html>

Smartbear 2023. What is Automated Testing? Smartbear [accessed 4 October 2023]. Available at: <https://smartbear.com/learn/automated-testing/what-is-automated-testing/>

NanoHub.org 2023. NanoHub.org [accessed 3 October 2023]. Available at: <https://nanohub.org/>

Oracle 2023. Java SE Development Kit 17 Downloads. Oracle [accessed 2 October 2023]. Available at: <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

Quora 2023. How does the Selenium WebDriver work? Quora [accessed 3 October 2023]. Available at: <https://www.quora.com/How-does-the-Selenium-WebDriver-work>

Saunders (2023) Types of Research Design. Available at: <https://research.com/research/types-of-research-design>

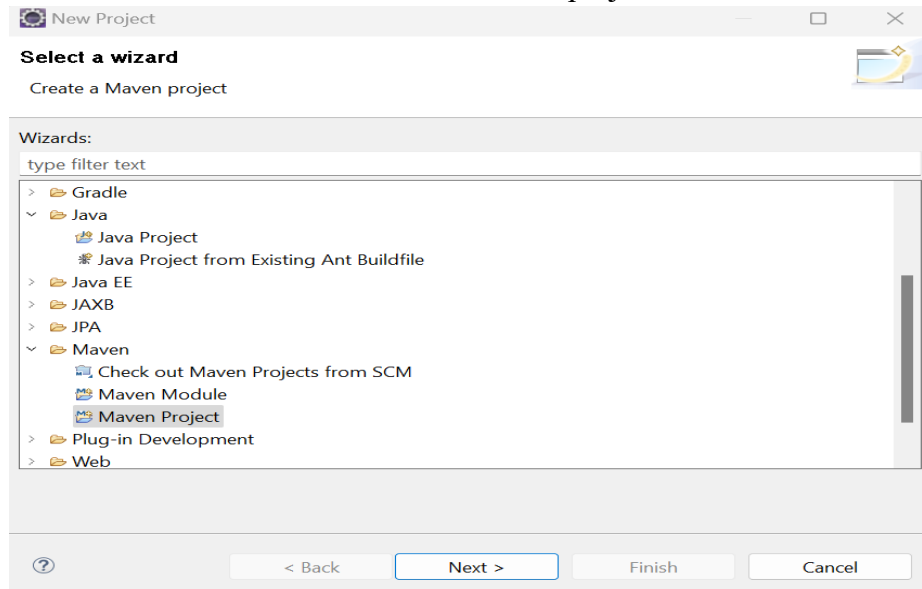
Techopedia 2023. Java. Techopedia [accessed 1 October 2023]. Available at: <https://www.techopedia.com/definition/3927/java>

Wikipedia 2023. Java (programming language). Wikipedia [accessed 3 October 2023]. Available at: [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

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.

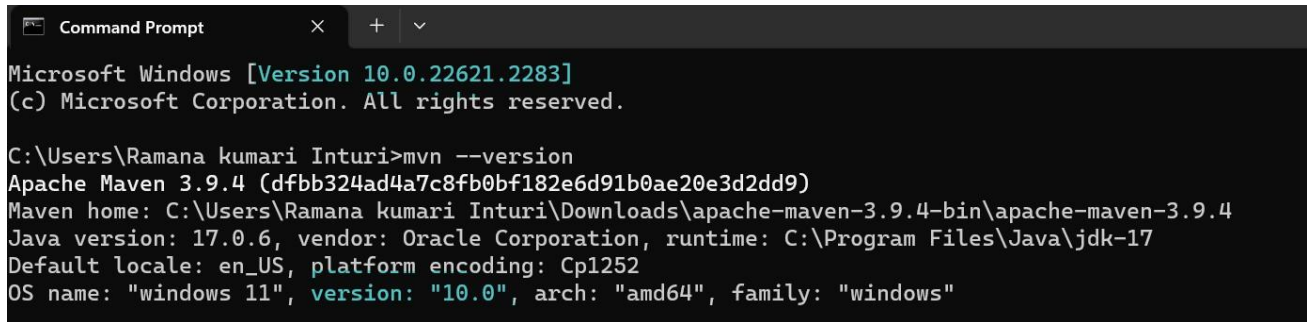
Step 4: Enter GroupID (unique name of the project) and ArtifactID (name of the jar file) and click Finish.

Result: A Maven project is ready for further implementation,

- ✓ PageObjectModel
 - > src/main/java
 - > src/test/java
 - > JRE System Library [jdk-17]
 - > Maven Dependencies
 - > src
 - > target
 - > test-output
 - 📄 pom.xml

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

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



```
Command Prompt
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ramana kumari Inturi>mvn --version
Apache Maven 3.9.4 (dfbb324ad4a7c8fb0bf182e6d91b0ae20e3d2dd9)
Maven home: C:\Users\Ramana kumari Inturi\Downloads\apache-maven-3.9.4-bin\apache-maven-3.9.4
Java version: 17.0.6, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
```

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

```
<groupId>NanoHub</groupId>
```

```
<artifactId>NanoHub</artifactId>
```

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

```

http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
<?xml version="1.0" encoding="UTF-8"?>
<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>NanoHub</groupId>
  <artifactId>NanoHub</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>NanoHub</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.10.0</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/io.github.bonigarcia/webdrivermanager -->
    <dependency>
      <groupId>io.github.bonigarcia</groupId>
      <artifactId>webdrivermanager</artifactId>
      <version>5.5.0</version>
    </dependency>

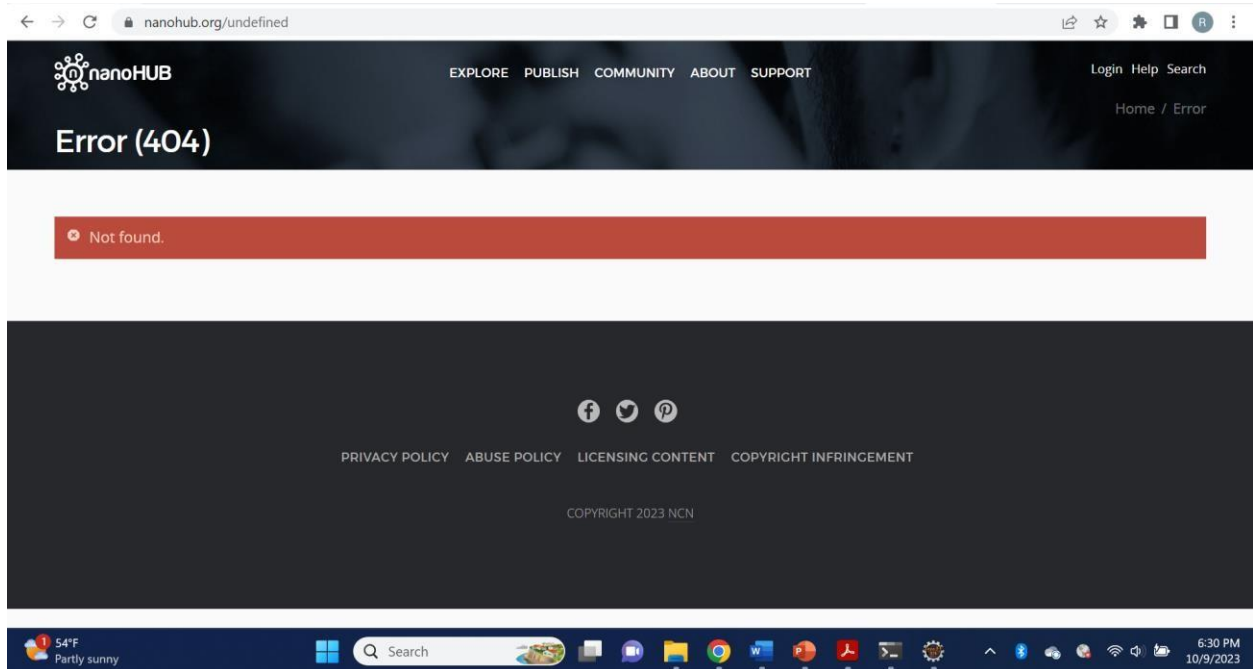
    <!-- https://mvnrepository.com/artifact/org.testng/testng -->
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>7.7.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

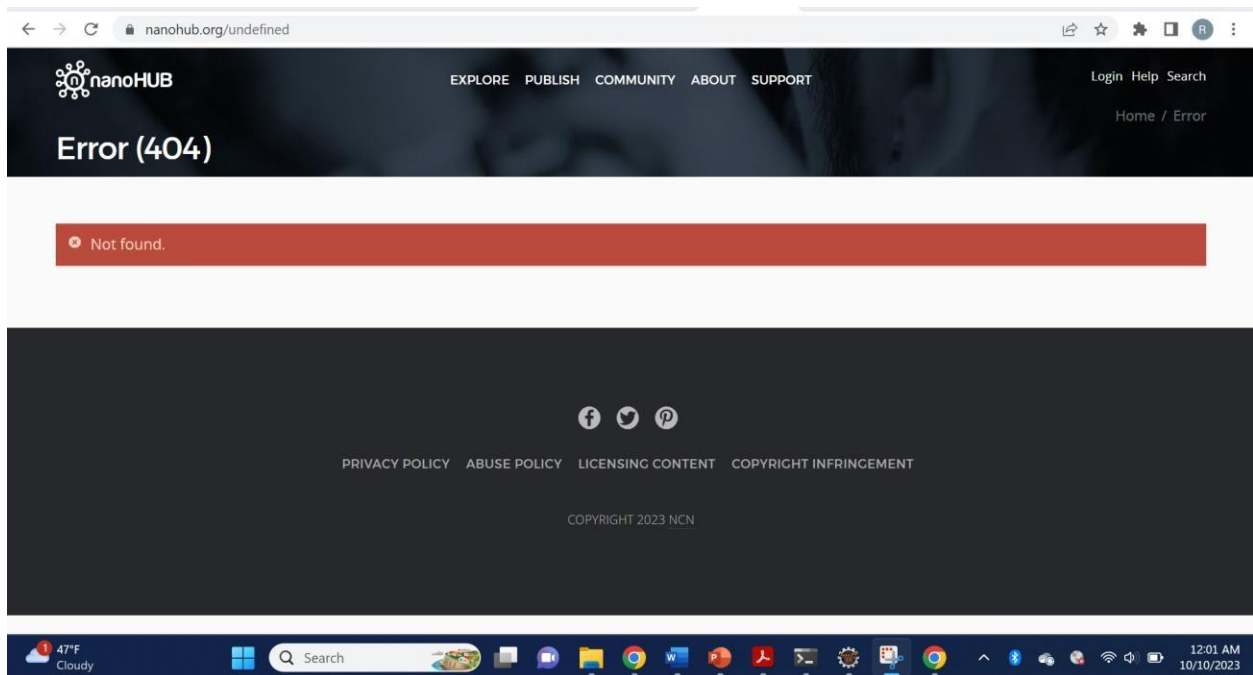
Appendix 3

Defects with the nanoHub web application during automation.

Defect 1: Go to Develop Software Click on Developer Kits module.



Defect 2: Go to Share&Publish Click on Apps, Tools & Data



Defect 3: Go to Model & Simulate □ Click on Apps for everyone □ Click on Nanoelectronics.

