CSE 565 - Software Verification Validation and Testing Project Portfolio

Kedar Sai Nadh Reddy Kanchi
School of Computing and Augmented Intelligence
Arizona State University, Tempe
kkanchi@asu.edu

Assignment 1

The software development process is incomplete without testing. Unit testing is a vital aspect of software testing that is used to ensure code quality. Unit testing frameworks are tools that help developers to test their code automatically and efficiently. The goal of this academic project is to provide an in-depth analysis and evaluation of unit testing frameworks in software development.

The project is divided into two parts. The first part of the project involves providing an overview of what a unit testing framework is, its scope, purpose, and an example of usage with a use case. The overview will also cover how a developer can use a framework and the benefits it provides. Furthermore, the overview will include a comparison of two frameworks in terms of common capabilities as well as any differences for a language such as Java, C++, Python, and JavaScript.

The second part of the project entails implementing the Heap-sort algorithm in Java, C++, Python, or JavaScript and testing the code with a unit testing framework of choice. The report will include a copy of the code, unit testing framework test cases, and an output report in the form of a screenshot from the testing tool.

Work Completed

To begin the project, I thoroughly investigated unit testing frameworks by performing a web search on "What is a unit testing framework?" and reading through multiple articles like "Unit testing" [1] & "Unit Test Frameworks by Paul Hamill" [2] to gain a comprehensive understanding of the topic. Through this research, I collected valuable information which allowed me to provide an in-depth analysis and evaluation of the scope, purpose, and usage of unit testing frameworks. I also provided an example of usage with a use case.

Continuing with the project, I conducted further research by performing a web search on "How would a developer utilize a unit testing framework?" and read through multiple articles like "What is Unit Testing and Why Developer Should Learn It"[3] & "Why Is Unit Testing Important in Software Development?" [4] to understand the Unit Testing Life Cycle to understand how developers utilize unit testing frameworks, and why they are widely used in software development. By singling out relevant information from the articles, I could comprehensively discuss how a developer would utilize a unit testing framework and its benefits.

For the last part of the project's first section, I compared and contrasted two unit testing frameworks, JUnit and Jest, that are used extensively with Java and JavaScript. To accomplish this, I performed additional web searches for JUnit and Jest, analyzed various articles like "Jest Framework" [5], "JUnit Tutorial — Testing Framework for Java" [6] & "What are the differences between Jest and JUnit?"[7], and identified the required information. I then documented the similarities and differences between the two frameworks regarding their capabilities.

To begin the second part of the project, I chose JavaScript as the language to implement the Heap Sort Algorithm and began searching for the algorithm online. I found a JavaScript implementation of the Heap Sort Algorithm online[8] (code attached in the appendix) and used it to implement the algorithm for this project. I then tested the Heap Sort code with the Jest unit testing framework suitable for JavaScript.

Because I was unfamiliar with Jest, I relied on a YouTube tutorial[9] and the official Jest documentation[10] to gain a thorough understanding of creating test cases with this framework. Once I was confident in my ability to generate test cases, I proceeded to generate the necessary test cases (code attached in the appendix) to ensure the functionality of the Heap Sort code. I ran the test cases through Jest after creating them, and the output report confirmed the code's correct operation.

Results

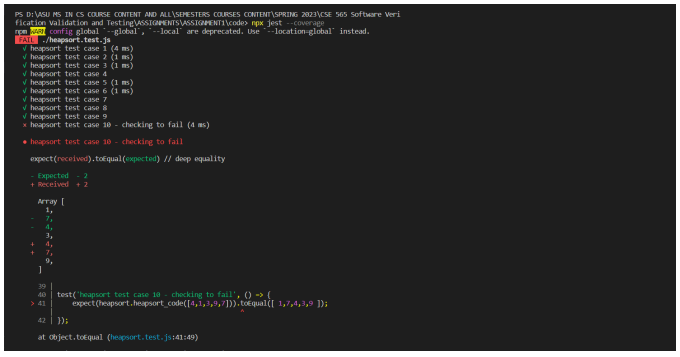


Figure 1: Jest Testing Coverage of the Heap Sort Algorithm

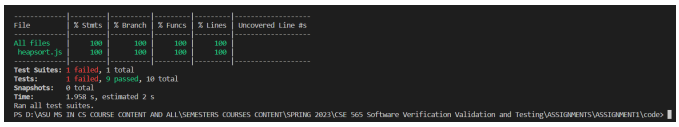


Figure 2: Total Jest Testing Coverage of the Heap Sort Algorithm

Lessons Learned

Throughout this project, I have comprehensively understood unit testing frameworks and their significance & purpose in software development. One important lesson I have learned is that unit testing frameworks can significantly enhance the software development process by automating the testing process, reducing the time and effort required to find and fix bugs, and increasing the overall quality of the code. Additionally, I have learned that understanding the unit testing life cycle is critical to utilizing unit testing frameworks effectively.

Furthermore, I have learned that it is vital to clearly understand the software development life cycle, specifically the unit testing life cycle, to utilize unit testing frameworks effectively. By understanding the unit testing life cycle, I could comprehend how developers utilize unit testing frameworks and their significance in software development.

I also learned that researching and comparing multiple frameworks can be time-consuming, but it is necessary to ensure that the best framework is selected for the project's requirements. In this project, I compared two-unit testing frameworks, JUnit and Jest, to determine their similarities and differences. This project has helped me understand the importance of selecting the appropriate testing framework for a specific programming language and project scope.

Lastly, generating test cases can be challenging, and it is essential to understand the framework's capabilities and features before beginning the testing process. This project has taught me to rely on documentation and tutorials to ensure that I thoroughly understand the testing framework before generating test cases.

Assignment 2

This project involves testing a database program that collects information and provides discounts for a product and its users based on various input factors. The project utilizes a pre-existing jar file containing the program with at least ten seeded defects. The goal of this project is to identify those ten seeded defects.

Testing aims to identify any flaws or bugs in the program and ensure that it performs its intended function correctly. Equivalence partitioning and boundary value analysis are commonly used in software testing to determine the most critical input parameter values. Furthermore, cause/effect analysis aids in the identification of relationships between inputs and outputs, allowing test cases to be refined further and the testing process to be more effective.

To achieve the objective of this project, we need to employ various software testing techniques such as equivalence partitioning, boundary value analysis, and cause/effect analysis, which have been discussed in lectures. These techniques will help us develop a comprehensive set of test cases that can be used to identify the defects in the program. Once the test cases are developed, the tests will be executed according to the instructions to ensure that the program meets the required standards of functionality and reliability. By executing these tests, we aim to ensure that the program operates as intended and that all identified defects are effectively addressed.

Work Completed

In this project, I completed the first testing stage by using the Boundary Value Testing Technique to create test cases for three requirements: Username length, User Age, and Product Rating. This technique involves selecting test cases that explore the boundaries or limits of the input domain of a software application. The input domain refers to the range of valid and invalid input values the software application can accept. The objective of boundary value testing is to identify any defects or bugs that may exist at the boundaries of the input domain, as these are often the areas where defects are most likely to occur. To ensure that all possible scenarios were covered by using the Boundary Value Testing Technique exclusively, I created multiple test cases for each of the three requirements with different values that either met the requirement or were just above or below the required limits.

- **Requirement 1.1:** Username can be 5-10 characters long.
  - I created 4 different test cases with usernames of length 4, 5, 10, and 11.
- **Requirement 2.1:** User Age must be 18 years of age and above.
  - I created three different test cases with user ages of 16, 17 and 18.
- **Requirement 7.1:** Rating of Product must be between 1 – 10.
  - I created 4 different test cases with usernames of length 0, 1, 10, and 11.

I have completed the second testing stage using the Equivalence Partitioning Testing Technique. This technique involves dividing the input domain of a software application into groups of equivalent values called equivalence classes. This technique aims to reduce the number of test cases required to test the software application while ensuring that all possible scenarios are covered. Instead of testing each input value, only one representative value from each equivalence class is selected. For this project, I used the Equivalence Partitioning testing technique exclusively for the following requirements:

- **Requirement 1.2:** Username can only contain letters and no numbers.
  - I created two different test cases where one test case tests the username containing numbers, and another test case tests the username which does not contain numbers.

- **Requirement 1.3:** Username cannot contain a hyphen (-) or an underscore (_).
  - I created two test cases: one test case tests the username containing a hyphen (-), and the other tests the username containing an underscore (_).

So, till now, in this project, I have developed test cases using only the boundary value testing technique or the equivalence portioning testing technique. However, software testing often uses these techniques to complement each other's strengths and weaknesses. This is because Boundary value testing effectively tests the boundaries of the input domain, while equivalence partitioning effectively reduces the number of test cases required by grouping inputs into equivalence classes. By using these techniques together, we can ensure that all input values within the input domain are tested, including those at the boundaries, and reduce the required test cases. So, following this learning, I used a combination of the Boundary Value and Equivalence Partitioning Testing Techniques to test three requirements: Username length, User Age, and Rating of the Product. By using these techniques together, I reduced the required test cases while ensuring that all input values within the input domain were tested.

- **Requirement 1.1:** Username can be 5-10 characters long.
  - I created 3 different test cases with usernames of length 4, 6, and 11.
- **Requirement 2.1:** User Age must be 18 years of age and above.
  - I created three different test cases with user ages of 16, 17 and 100.
- **Requirement 7.1:** Rating of Product must be between 1 – 10.
  - I created 2 different test cases with usernames of length 0 and 11.

In the final stage of this project, I utilized the Cause/Effect Analysis testing technique. Cause/effect analysis involves identifying potential causes of defects or problems by analyzing the relationships between different factors or variables in a system. This technique is based on the principle that every defect or problem in a system has underlying causes that can be identified by examining the relationships between the different factors contributing to the defect. In cause/effect analysis, the tester begins by identifying the defect or problem to be tested and then identifies the factors or variables that could potentially contribute to the defect. These factors include inputs, conditions, and other variables affecting the system's behavior. Therefore, as part of this project, I have exclusively used cause/effect analysis testing techniques for specific requirements best suited for this technique.

- **Requirement 9.1:** New users cannot be given any discounts regardless of reward status, product, and season.
  - I have identified and created 12 different test cases that could potentially contribute to the defect.
- **Requirement 9.2:** If product category is unknown, user gets no discount.
  - I have identified and created 12 different test cases that could potentially contribute to the defect.
- **For combination of Requirement 9.1 Requirement 9.2**
  - I have identified and created 12 different test cases that could potentially contribute to the defect.
- **Requirement 9.3.1:** If bought in Spring, then the returning users who are bronze members who bought Electronics get a 10% discount voucher.
- **Requirement 9.4.1:** If bought in Winter, then the returning users who are gold members who bought Electronics get a 25% discount voucher.
- **Requirement 9.5.1:** If bought in Summer, then the returning users who are gold members who bought Electronics get a 15% discount voucher.
- **Requirement 9.6.1:** If bought in Fall, then the returning users who are silver members who bought Electronics get a 15% discount voucher.
- **Requirement 9.7:** Any other combination of returning user, reward status, product, and season does not get a discount
  - I have identified and created 8 different test cases that could potentially contribute to the defect.

Results

Overall, I have developed a comprehensive set of 65 test cases using these testing techniques and identified the seeded defects. Combining these testing techniques helped me ensure that all possible scenarios were covered and reduced the required test cases.

The ten identified defects are as follows:

- The user name is field is accepting a name which has more than 10 characters. This is invalid because as per the requirements the user name can only contain 5-10 characters.
  - **Requirement Mapping:** 1.1
  - **Observed Output:** The discount given is 15%
  - **Expected Output:** Username length is invalid
- The user name field is accepting the underscore character as a valid character. This is should happen because as per the requirements the user name should not contain the underscore character.
  - **Requirement Mapping:** 1.3
  - **Observed Output:** The discount given is 15%
  - **Expected Output:** Username contains _
- The application gives an output even when the user enters the age as 17. This should not happen because as per the requirements the application should not calculate for the output i.e. in this case discount when the age of the user is less than 18.
  - **Requirement Mapping:** 2.1
  - **Observed Output:** The discount given is 15%

  - **Expected Output:** You must be 18 or older to register in the database.
- A rating of 0 which is less than 1 is accepted by the application and the discount is being calculated to be shown as an output. This is not the ideal behaviour because as per the requirements the range of values that should be accepted by the ratings field is between 1 and 10 including both the values.
  - **Requirement Mapping:** 7.1
  - **Observed Output:** The discount given is 15%
  - **Expected Output:** Rating must be between 1 and 10.
- Even when the user status is set to New, the application is calculating for the discount and giving an output "The discount given is 10%". This is the wrong output as the ideal output should have been "No discount given" because as per the requirements New users cannot be given any discounts regardless of reward status, product, and season.
  - **Requirement Mapping:** 9.1
  - **Observed Output:** The discount given is 10%
  - **Expected Output:** No discount given.
- Even when the product category is set to unknown - the combination of where the user status is set to Returning, the reward member status is set to Silver, and the season when the item was bought is set to Summer, the application gives an output "the discount given is 10%" instead of "No discount given" which is not the ideal case. This because as per the requirements, if product category is set to unknown, then the user gets no discount.
  - **Requirement Mapping:** 9.2
  - **Observed Output:** The discount given is 10%
  - **Expected Output:** No discount given
- As per the requirements, if the item is bought in Spring, then the Returning users who are Bronze members who bought Electronics should get a 10% discount voucher. But when the same combination is entered into the application the output given is "the discount given is 5%" which is incorrect.
  - **Requirement Mapping:** 19.4.1
  - **Observed Output:** The discount given is 5%
  - **Expected Output:** The discount given is 10%
- As per the requirements, if the item is bought in Winter, then the Returning users who are Gold members who bought Electronics should get a 25% discount voucher. But when the same combination is entered into the application the output given is "the discount given is 20%" which is incorrect.
  - **Requirement Mapping:** 9.5.1
  - **Observed Output:** The discount given is 20%
  - **Expected Output:** The discount given is 25%
- As per the requirements, if the item is bought in Fall, then the Returning users who are Silver members who bought Electronics should get a 15% discount voucher. But when the same combination is entered into the application the output given is "the discount given is 10%" which is incorrect.
  - **Requirement Mapping:** 9.6.1
  - **Observed Output:** The discount given is 10%
  - **Expected Output:** The discount given is 15%
- As per the requirements, if any other combination of inputs(returning user, reward status, product, and season) set apart from the ones mentioned in 9.3.1, 9.4.1, 9.5.1, and 9.6.1 then the user will not get a discount. But in this case if the user status is set to Returning, the user member status is set to Gold, the season in which the user purchases the item is set to Fall and the product category is set to Electronics, then the output given by the application is "The discount given is 10%" instead of the output "No discount is given".
  - **Requirement Mapping:** 9.7
  - **Observed Output:** The discount given is 10%
  - **Expected Output:** No discount given

Lessons Learned

This project provided valuable hands-on experience in software testing, highlighting the importance of rigorous testing of software applications to improve the ability to identify and mitigate defects in software programs. The project showed that different testing techniques could be used to ensure software program quality and performance. Specifically, the project demonstrated the benefits of the Boundary Value Testing Technique, Equivalence Partitioning Testing Technique, and Cause/Effect Analysis Testing Technique.

The Boundary Value Testing Technique proved to be a practical approach to identifying defects or bugs that may exist at the boundaries of the input domain, while the Equivalence Partitioning Testing Technique reduced the number of test cases required to test the software application. By combining these techniques, we could test all input values within the input domain, including those at the boundaries, and reduce the required test cases. Using equivalence partitioning in software testing can save time and effort while improving software application quality and reliability.

Moreover, the Cause/Effect Analysis Testing Technique proved to be a powerful method of identifying potential causes of defects or problems by analyzing the relationships between different factors or variables in a system. This technique helped identify underlying causes contributing to the defects, enabling efficient mitigation of these issues. The cause/effect analysis technique is beneficial for testing complex systems where the relationships between different variables are not always apparent. It can help testers to identify the root causes of defects or problems, rather than just the symptoms, and develop more targeted and effective test cases to address these underlying causes.

The project demonstrated the importance of utilizing different testing techniques to ensure software application quality and performance. Identifying and mitigating defects or bugs early in the development cycle can reduce the likelihood of encountering issues during production, thus increasing software programs' overall reliability and quality.

Assignment 3

In today's fast-paced mobile application industry, applications must be thoroughly tested to ensure a seamless user experience, quality, and reliability. Testing is crucial; it helps identify and fix bugs and other issues before deployment. However, testing all possible combinations of input variables can be a daunting task, mainly when many variables and combinations are involved.

One solution to this problem is using the Design of Experiments (DOE). This statistical method can help optimize testing efforts by creating test cases based on pairwise combinations of requirements, allowing the testers to test many variables and their combinations efficiently.

In this project, the testing team is tasked with utilizing a DOE tool to create test cases that cover all possible pairwise combinations of requirements for a newly developed mobile application. We will use a DOE tool to develop test cases based on the given specifications to achieve this.

The primary objective of this project was to conduct research and identify a tool that supports DOE that can support the creation of pairwise combination test cases for the mobile application and to demonstrate its application in creating test cases.

Additionally, the project required me to prepare a report that aims to provide an overview of the selected DOE tool, explain its scope and purpose, and provide an example of its usage through a use case.

Furthermore, we will provide details on the number of test cases created by the tool, along with high-resolution screenshots. Finally, we will evaluate the DOE tool based on its usability, coverage, and features and provide our assessment in my own words.

Work Completed

To start the project, I scoured the internet for articles and information about the Design of Experiments (DOE) and spent considerable time reading through multiple websites like "WHAT IS DESIGN OF EXPERIMENTS (DOE)?" [11] and watching lecture videos to form a solid understanding of what DOE is and when it is used. With this knowledge, I began my report by explaining DOE and its use.

After understanding DOE, I learned the basics of pairwise testing, which involves creating test cases that cover all possible pairwise combinations of requirements. I again took to the internet to learn about pairwise testing, spending a significant amount of time reading through multiple websites like "Pairwise Testing Or All-Pairs Testing Tutorial With Tools And Examples"[12] & "Pairwise Testing Guide: How To Perform Pairwise Testing"[13] and watching lecture videos to understand the objective, benefits, and limitations of pairwise testing. I then explained pairwise testing in my report.

Now, with the foundations required for the project formed, I moved on to the next thing, which was to research different DOE tools and identify a suitable tool.

Since we are creating test cases that cover all possible pairwise combinations, I scrounged through the net searching for dependable articles and information about different Pairwise Testing tools available in the market. After reading through multiple articles and singling out helpful information, I understood that Pairwise testing tools are simple test case generators that allow testers to input and constrain a model of input parameters before generating test configurations based on the model. These tools are simple to use with many input parameters and can also add constraints to the input parameters and generate test configurations. To perform Pairwise testing, numerous tools available on the internet apply the all-pairs testing technique that facilitates us to effectively automate the Test Case Design process by generating a compact set of parameter value choices as the desired Test Cases. Some well-known tools from the industry are mentione below.

- **PICT**: 'Pairwise Independent Combinatorial Testing', provided by Microsoft Corp.
- **IBM FoCuS** : 'Functional Coverage Unified Solution', provided by IBM.
- **ACTS**: Advanced Combinatorial Testing System', provided by NIST, an agency of the US Government.
- **Hexawise**
- **Jenny**:
- **Pairwise** : by Inductive AS
- **VPTag** : free All-Pair Testing Tool

After going through the available tools listed above, the tool that I have used for the project is PICT (Pairwise independent combinatorial testing). The PICT tool from Microsoft is a well-documented pairwise test generator that also supports many advanced features (Microsoft, 2019[16]). PICT is an open-source and cross-platform program programmed in C++ and is single-threaded. The PICT tool is also compatible with various software systems, such as embedded devices, mobile apps, and web applications. It can be a versatile and adaptable testing tool for various software development projects. [14].

Now, as per the project's requirements, after identifying a suitable DOE tool for this project, I moved on to explain the PICT tool's scope and purpose. In simple terms, the scope of the PICT tool is that it supports higher interaction strength testing via a command line switch by allowing testers to generate more effective test cases and test configurations than manually generated tests in a fraction of the time that hands-on test case design requires. Again, in simple terms, the PICT tool's primary purpose is to generate a compact set of parameter value options that represent the test cases that the testers should use to achieve complete combinatorial coverage of project parameters. The PICT tool also includes constraints, value weights, and other advanced features described in Czerwonka (2006) [15]. Following these sections in the report, as per the project requirement, I have explained about usage of the PICT tool with a working example.

There are two different kinds of PICT tools available.

- **Online Version**: Pairwise Pict Online[17] - An online service that easily generates pair-wise test cases. It's powered by Microsoft Pict under the hood.

- **Offline Version**: PICT[16] - PICT runs as a command line tool

For this assignment I have used the offline version. So for this, I have visited the PICT releases web-page[16] and downloaded the executable (.exe) file and used this executable (.exe) file to generate pairwise combination test cases based upon the given requirements.

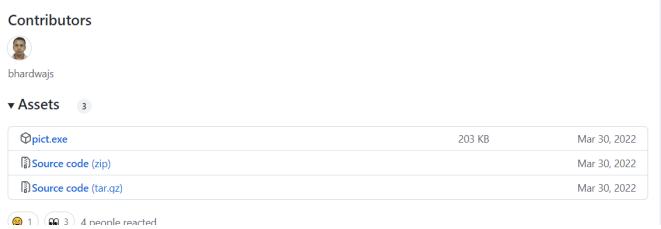


Figure 3: PICT offline latest version available releases

So, now with the DOE tool identified for the project, I moved on to creating the test cases. Taking the given requirements for the newly developed mobile application, the first step that I did was to develop the model file that would act as the input to the PICT tool. A model consists of the following sections: parameter definitions, [sub-model definitions], and the [constraint definitions] as you can see in the figure below[18].

Type of Phone	Parallel Tasks Running	Connectivity	Memory	Battery Level
iPhone X	Yes	Wireless	1 GB	< 20%
iPhone 8	No	3G	2 GB	20 -39%
Samsung S9		4G	4 GB	40 - 59%
Huawei Mate		Edge	6 GB	60 - 79%
Google Pixel 3				80 - 100%

Figure 4: Given requirements for the mobile application

```
parameter definitions
[sub-model definitions]
[constraint definitions]
```

Figure 5: Syntax to be followed for the input model file

Therefore, following the specified syntax, I created my input model file (CSE565 Assignment 3 parameters) as a text file.

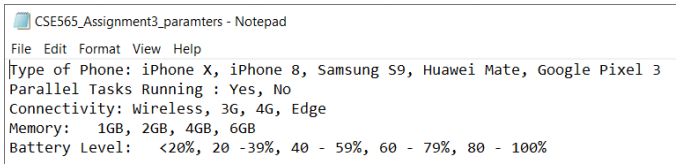


Figure 6: CSE565_Assignment3_paramters.txt file

Now, with the input model file created, I have opened my Command Prompt and navigated to the folder where I have my executable (.exe) file. After navigating the folder, I run the command" .pict CSE565 Assignment3 paramters.txt," and the test cases are generated.

With the tool identified and the test cases generated, to finish the report, I evaluated the usability, coverage, and features of the PICT tool. I scoured the internet for dependable articles and information like "Pairwise Independent Combinatorial Testing"[18] and formed a solid understanding of the tool's capabilities. Using the gathered information, I then completed the sections Assessment of the PICT tool in terms of usability, coverage, and features of the PICT tool and Assessment of the PICT tool in my own words in terms of usability, coverage, and features of the PICT tool, as per the project's requirements.

Results

So, to summarize, the number of test cases Generated by DOE tool i.e. the PICT tool are 25.

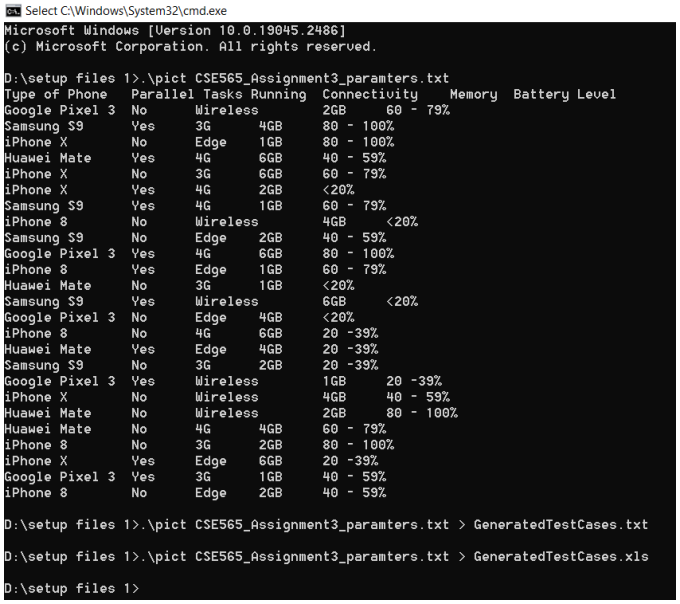


Figure 7: Generated Test Cases in Command Line Interface

Because of the readability issue of the output in the command line, I have saved the output in a different text file by running the command ".pict

CSE565_Assignment3_paramters.txt > GeneratedTestCases.txt" as you can see in the Figure above. And from the text file, the output has been moved into an excel file for better readability (which you can see at the end of this report) by running the command "./pict CSE565_Assignment3_paramters.txt > GeneratedTestCases.xls" as you can see in the figure above.

	A	B	C	D	E
1	Type of Phone	Parallel Tasks Running	Connectivity	Memory	Battery Level
2	Google Pixel 3	No	Wireless	2GB	60 - 79%
3	Samsung S9	Yes	3G	4GB	80 - 100%
4	iPhone X	No	Edge	1GB	80 - 100%
5	Huawei Mate	Yes	4G	6GB	40 - 59%
6	iPhone X	No	3G	6GB	60 - 79%
7	iPhone X	Yes	4G	2GB	<20%
8	Samsung S9	Yes	4G	1GB	60 - 79%
9	iPhone 8	No	Wireless	4GB	<20%
10	Samsung S9	No	Edge	2GB	40 - 59%
11	Google Pixel 3	Yes	4G	6GB	80 - 100%
12	iPhone 8	Yes	Edge	1GB	60 - 79%
13	Huawei Mate	No	3G	1GB	<20%
14	Samsung S9	Yes	Wireless	6GB	<20%
15	Google Pixel 3	No	Edge	4GB	<20%
16	iPhone 8	No	4G	6GB	20 -39%
17	Huawei Mate	Yes	Edge	4GB	20 -39%
18	Samsung S9	No	3G	2GB	20 -39%
19	Google Pixel 3	Yes	Wireless	1GB	20 -39%
20	iPhone X	No	Wireless	4GB	40 - 59%
21	Huawei Mate	No	Wireless	2GB	80 - 100%
22	Huawei Mate	No	4G	4GB	60 - 79%
23	iPhone 8	No	3G	2GB	80 - 100%
24	iPhone X	Yes	Edge	6GB	20 -39%
25	Google Pixel 3	Yes	3G	1GB	40 - 59%
26	iPhone 8	No	Edge	2GB	40 - 59%

GeneratedTestCases

Figure 8: Generated Test Cases in Excel File

Lessons Learned

The project has provided several valuable lessons related to DOE and Pairwise Testing. One key takeaway is that DOE is a powerful and flexible data collection and analysis tool that can be applied to various experimental situations. Its ability to manipulate multiple input factors to determine their impact on a desired output. By adjusting multiple inputs simultaneously, DOE can identify significant interactions that may go unnoticed when experimenting with a single factor at a time. Furthermore, DOE can be employed to validate hypothesized input-output relationships and develop predictive models suitable for what-if analysis. It has demonstrated good code coverage, and many companies have successfully used it in software testing.

Another important lesson learned is that Pairwise Testing is a black-box testing that ensures full test coverage. This technique is beneficial for designing tests for applications with multiple parameters, as it covers all possible combinations of input parameters. Tests are designed to ensure that each pair of input parameters to a system has all possible discrete combinations of those parameters. Although the test suite does not cover all possible combinations, it is highly effective at detecting defects. PICT is a powerful tool that can help testers create test cases optimized for pairwise testing, covering a wide range of input parameters and dealing with constraints. Its usability, coverage, and features make it an invaluable tool for software testing, particularly for projects with many input parameters. Software developers and testers can use PICT to ensure their applications are thoroughly tested and defects-free.

Using DOE and Pairwise Testing together can provide a robust testing solution. DOE can identify the input factors that have the most significant effect on the output, and Pairwise Testing can ensure that all combinations of those factors are tested. Using both techniques, testers can create a test suite optimized for maximum code coverage and defect detection. Using the appropriate testing tools and techniques for each project is essential. In this project, the use of DOE and Pairwise Testing was suitable due to the large number of input parameters that needed to be tested. Other testing techniques may be more appropriate for projects with fewer input parameters.

Overall, the project provided important insights into the capabilities of DOE and Pairwise testing and their usefulness in software testing. Both techniques are powerful tools that can help testers improve the efficiency, effectiveness, and flexibility of the testing process and enhance the quality of software products. Software developers and testers should consider incorporating these techniques into their testing processes to ensure comprehensive test coverage and detect defects early in the development cycle to optimize testing strategies and improve software quality.

Assignment 4

This project consists of four tasks.

In Task 1, the VendingMachine.java code is provided, and the goal is to use structural-based testing techniques to develop a set of test cases using JUnit for code coverage. The test cases should aim to reach 100% statement coverage and at least 90% decision coverage, except for the False decision for line 32 (input < 45).

Task 2 involves creating a report describing the tool used for Task 1, including its scope, purpose, and an example of usage. Additionally, the types of code coverage provided by the tool and the test cases' scope and coverage developed in Task 1 should be explained. Finally, the report should include a screenshot of the tool's report showing the coverage achieved by the test cases and an evaluation of the tool's usefulness in terms of usability, coverage, and tool features.

Task 3 requires selecting a static source code analysis tool and executing it on StaticAnalysis.java to identify the code's two different data flow anomalies. Task 4 involves continuing the report created in Task 2, which should include a description of the two data flow anomalies and why they are considered data anomalies. Furthermore, the report should include a high-resolution screenshot showing the analysis performed for detecting data anomalies and evaluating the tool's usefulness regarding usability, coverage, and tool features.

Work Completed

I began this project by selecting an appropriate Integrated Development Environment (IDE) for the task. After considering the available options, I used IntelliJ IDEA, commonly

paired with JaCoCo or Emma for high-level applications. However, since the task was to test the functionality of a single file, I opted to use the less resource-intensive IntelliJ Code Coverage Runner. IntelliJ IDEA is an Integrated Development Environment (IDE) for Java, Kotlin, and other programming languages, and IntelliJ Code Coverage Runner is one of IntelliJ's main features. It is an inbuilt code coverage tool that allows developers to measure how much their tests exercise their code. [19]

I then explained the scope and purpose of the IntelliJ Code Coverage Runner tool, drawing on information from dependable articles and online resources[19]. In simple terms, the scope of the IntelliJ Code Coverage Runner in IntelliJ IDEA is that the tool allows developers to measure code coverage for a wide range of programming languages and frameworks, with the ability to analyze code coverage for the unit, integration, and functional tests. At the same time, the purpose of the IntelliJ Code Coverage Runner tool is to analyze code and determine which statements or lines have been executed during tests, enabling developers to identify untested or under-tested areas of their code and create more effective tests to improve code quality. Following these two sections, I described a working example as a use case to showcase the usage of the IntelliJ Code Coverage Runner tool.

I then described the different types of code coverage the IntelliJ Code Coverage Runner tool provided, drawing on information from the lecture videos. The IntelliJ Code Coverage Runner tool in IntelliJ IDEA offers various types of code coverage to assist developers in determining how thoroughly a given test suite executes a program's source code. The following are the types of code coverage analysis provided by the IntelliJ IDEA code coverage runner tool: Statement/Line Coverage, Branch/Decision Coverage, Method Coverage, and Class Coverage.

Now, with the identified IDE and the tool for this project, I moved to develop a set of test cases using JUnit for code coverage. To summarize the VendingMachine.java code, there are 24 lines/statements and eight decisions/conditions in the dispenseItem function. The eight decisions/conditions result in 16 branches because each decision has two possible outcomes (true or false). The 8 decisions/conditions in the code are:

- **Decision 1**: item == "candy"
- **Decision 2**: item == "coke"
- **Decision 3**: item == "coffee"
- **Decision 4**: input $>$ cost
- **Decision 5**: input == cost
- **Decision 6**: input $<$ 45
- **Decision 7**: input $<$ 25
- **Decision 8**: input $<$ 20

So, I started creating test cases covering all 24 lines/statements and the 15 decision branches. After multiple attempts, I was finally able to create a set of 5 test cases that resulted in the required code coverage, which is 100% statement coverage and at least 90% decision coverage.

With the test cases developed, I moved on to the next part, describing each test case, its scope, and coverage in the report. Each test case was described in detail in the report, including the test case's description, scope, and coverage. The coverage of each test case was divided into two subsections, Statement/Line Coverage, and Decision/Branch Coverage. The Statement/Line Coverage subsection showcased which lines/statements of the code were covered by that particular test case and the percentage in total. The Decision/Branch Coverage subsection showcased which decision and which branch (True/False) of that decision were covered by that particular test case and the percentage of total decision coverage. Lastly, to complete Task 2, I had to include in the report the evaluation of the IntelliJ Code Coverage Runner tool in my own words. Using the tool firsthand and with some information from some dependable websites, I could efficiently complete this section, describing the tool's usability, coverage, and features.

Moving on to task 3, I had to select a static source code analysis tool. Before using which tool to be used, I wanted to learn what precisely static source code analysis is. So, for this, I scrounged through the net, searching for dependable articles and information about static source code analysis. After reading multiple articles, I understood what static source code analysis is and what was expected from me in this task. So, I scrounged through the net, searching for various static source code analysis tools available in the market. After going through all the available tools, I decided that the PMD(Programming Mistake Detector)[20] is suitable for this project. This is because PMD is a well-known open-source static source code analysis tool in the Java community for detecting potential bugs or issues in programming languages like Java, Kotlin, and others and is available as a plugin for several popular Java development environments, such as Eclipse and IntelliJ IDEA. Without running the program, PMD examines its source code for common programming errors, identifying coding issues such as unused variables, empty catch blocks, and inefficient code constructs and recommending improvements.

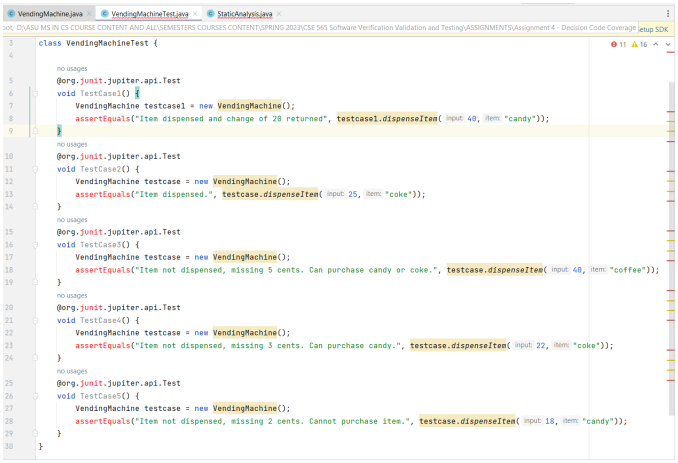
So, using this tool, I completed Task 3 to identify the two data flow anomalies in the StaticAnalysis.java code file. Lastly, to complete Task 4, I had to include in the report the evaluation of the PMD(Programming Mistake Detector) tool in my own words. Using the tool firsthand and with some information from some dependable websites, I could efficiently complete this section, describing the tool's usability, coverage, and features.

Results for Code Coverage

I created the minimum number of test cases which is 5 as shown in the attached image Figure 9, to complete the code coverage while satisfying the given condition mentioned in the project description - 100% statement coverage and at least 90% decision coverage. And, as shown in the attached images, Figures 12 and 13, the developed unit test cases provide the required coverage.

Total Statement Coverage

From Figure 10, you can see that we have a 100% statement coverage.



```
class VendingMachineTest {

    no usages
    @org.junit.jupiter.api.Test
    void TestCase1() {
        VendingMachine testcase1 = new VendingMachine();
        assertEquals("Item dispensed and change of 20 returned", testcase1.dispenseItem( input: 40, item: "candy"));
    }
    no usages
    @org.junit.jupiter.api.Test
    void TestCase2() {
        VendingMachine testcase = new VendingMachine();
        assertEquals("Item dispensed.", testcase.dispenseItem( input: 25, item: "coke"));
    }
    no usages
    @org.junit.jupiter.api.Test
    void TestCase3() {
        VendingMachine testcase = new VendingMachine();
        assertEquals("Item not dispensed, missing 5 cents. Can purchase candy or coke.", testcase.dispenseItem( input: 40, item: "coffee"));
    }
    no usages
    @org.junit.jupiter.api.Test
    void TestCase4() {
        VendingMachine testcase = new VendingMachine();
        assertEquals("Item not dispensed, missing 3 cents. Can purchase candy.", testcase.dispenseItem( input: 22, item: "coke"));
    }
    no usages
    @org.junit.jupiter.api.Test
    void TestCase5() {
        VendingMachine testcase = new VendingMachine();
        assertEquals("Item not dispensed, missing 2 cents. Cannot purchase item.", testcase.dispenseItem( input: 18, item: "candy"));
    }
}
```

Figure 9: Generated Test Cases for the Vending Machine

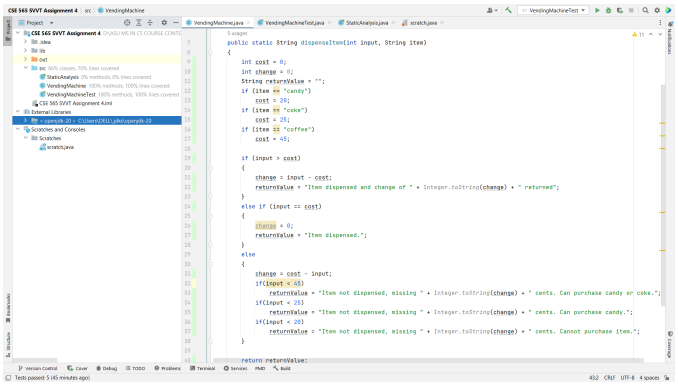
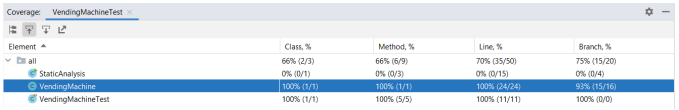


Figure 10: The visual representation of the code coverage

- **Line 9**: Covered by the Test Cases - 1, 2, 3, 4, and 5.
- **Line 10**: Covered by the Test Cases - 1, 2, 3, 4, and 5.
- **Line 11**: Covered by the Test Cases - 1, 2, 3, 4, and 5.
- **Line 12**: Covered by the Test Cases - 1, 2, 3, 4, and 5.
- **Line 13**: Covered by the Test Cases - 1, and 5.
- **Line 14**: Covered by the Test Cases - 1, 2, 3, 4, and 5.
- **Line 15**: Covered by the Test Cases - 2, and 4.
- **Line 16**: Covered by the Test Cases - 1, 2, 3, 4, and 5.
- **Line 17**: Covered by the Test Cases - 3.
- **Line 19**: Covered by the Test Cases - 1, 2, 3, 4, and 5.
- **Line 21**: Covered by the Test Cases - 1.
- **Line 22**: Covered by the Test Cases - 1.
- **Line 24**: Covered by the Test Cases - 2, 3, 4, and 5.
- **Line 26**: Covered by the Test Cases - 2.
- **Line 27**: Covered by the Test Cases - 2.
- **Line 31**: Covered by the Test Cases - 3, 4, and 5.
- **Line 32**: Covered by the Test Cases - 3, 4, and 5.
- **Line 33**: Covered by the Test Cases - 3, 4, and 5.

```
Cover: VendingMachineTest
Tests passed: 5 of 5 tests – 70 ms
VendingMachineTest 70 ms
TestCase1() 60 ms
TestCase2() 2 ms
TestCase3() 2 ms
TestCase4() 2 ms
TestCase5() 2 ms
C:\Users\LELL\.jdks\openjdk-20\bin\java.exe ...
---- IntelliJ IDEA coverage runner ----
tracing and tracking per test coverage ...
include patterns:
exclude annotations patterns:
.*Generated.*
Class transformation time: 0.49524295 for 896 classes or 5.527264508928571E-4 per class
Process finished with exit code 0
```

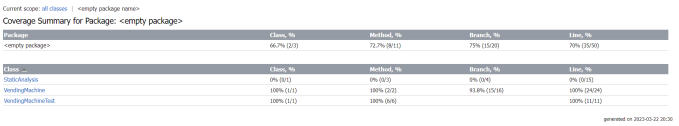
Figure 11: Terminal Output of the Total Code Coverage



Coverage: VendingMachineTest

Element	Class, %	Method, %	Line, %	Branch, %
all	66% (2/3)	66% (6/9)	70% (35/50)	75% (15/20)
StaticAnalysis	0% (0/1)	0% (0/3)	0% (0/15)	0% (0/4)
VendingMachine	100% (1/1)	100% (1/1)	100% (24/24)	93% (15/16)
VendingMachineTest	100% (1/1)	100% (5/5)	100% (11/11)	100% (0/0)

Figure 12: Total Code Coverage Statistics



Current scope: all classes | <empty package name>

Coverage Summary for Package: <empty package>

Package	Class, %	Method, %	Branch, %	Line, %
<empty package>	66.7% (2/3)	72.7% (8/11)	75% (15/20)	70% (35/50)

Class	Class, %	Method, %	Branch, %	Line, %
StaticAnalysis	0% (0/1)	0% (0/3)	0% (0/4)	0% (0/15)
VendingMachine	100% (1/1)	100% (2/2)	93.8% (15/16)	100% (24/24)
VendingMachineTest	100% (1/1)	100% (6/6)		100% (11/11)

generated on 2023-03-22 20:30

Figure 13: Exported Coverage Report

- **Line 34**: Covered by the Test Cases - 3, 4, and 5.
- **Line 35**: Covered by the Test Cases - 4, and 5.
- **Line 36**: Covered by the Test Cases - 3, 4, and 5.
- **Line 37**: Covered by the Test Cases - 5.
- **Line 40**: Covered by the Test Cases - 1, 2, 3, 4, and 5.

Total Decision Coverage

Figure 10 shows that we have a decision coverage of 93%. This is due to the fact that, as stated in the project description, all decisions have been covered with the exception of the False branch for the decision/condition on line 32 (input $<$ 45). So, the code covers 15 of the 16 branches (8 conditions), for a total of 93.75% coverage.

- **Decision 1**: item == ”candy”
  - **True Branch**: Covered by the Test Cases - 1 and 5.
  - **False Branch**: Covered by the Test Cases - 2, 3, and 4.
- **Decision 2**: item == ”coke”
  - **True Branch**: Covered by the Test Cases - 2 and 4.
  - **False Branch**: Covered by the Test Cases - 1, 3, and 5.
- **Decision 3**: item == ”coffee”
  - **True Branch**: Covered by the Test Cases - 3.
  - **False Branch**: Covered by the Test Cases - 1, 2, 4, and 5.
- **Decision 4**: input $>$ cost
  - **True Branch**: Covered by the Test Cases - 1.
  - **False Branch**: Covered by the Test Cases - 2, 3, 4, and 5.
- **Decision 5**: input == cost
  - **True Branch**: Covered by the Test Cases - 2.
  - **False Branch**: Covered by the Test Cases - 3, 4, and 5.
- **Decision 6**: input $<$ 45
  - **True Branch**: Covered by the Test Cases - 3, 4, and 5.

  - **False Branch**: NOT COVERED.
- **Decision 7**: input < 25
  - **True Branch**: Covered by the Test Cases - 4.
  - **False Branch**: Covered by the Test Cases - 3 and 5.
- **Decision 8**: input < 20
  - **True Branch**: Covered by the Test Cases - 5.
  - **False Branch**: Covered by the Test Cases - 3 and 4.

Results from the Static Analysis Tool

To perform its analysis, PMD employs a set of predefined rules or code guidelines that cover a wide range of coding issues, such as inefficient code, unused variables, and potential security vulnerabilities. It can also be customized with additional rules to meet the requirements of a specific project. These rules are based on best practices and common coding errors, and they can be tailored to the needs of a specific project. When PMD is run, it reports any violations of these rules, as well as where they occur in the source code[22].

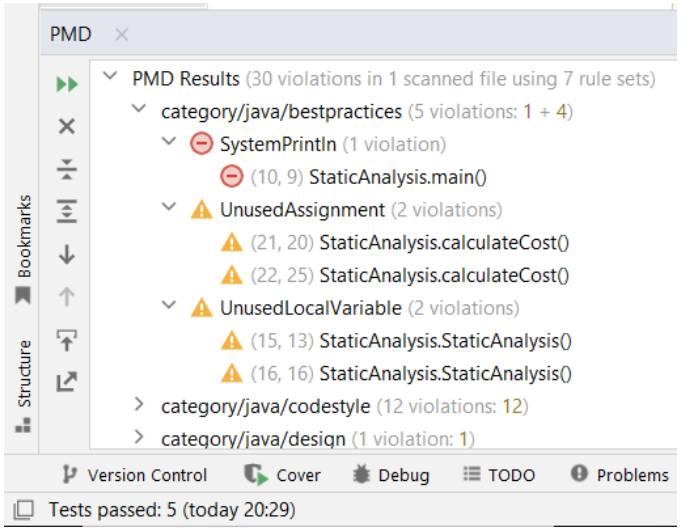


Figure 14: PMD RESULTS 1

Therefore, the StaticAnalysis.java file contains the following two different data flow anomalies:

- **UnusedAssignment – int cost = 0;** – Found 'DD'-anomaly for variable 'cost' (lines '21'-'26') in public static String calculateCost(). As the value of cost = 0 is never used and also it is later redefined on lines 26 and 30 depending on how the decision on line 24 evaluates, this error is thrown as Unused Assignment.
- **UnusedAssignment – String output = "";** – Found 'DD'-anomaly for variable 'output' (lines '22'-'34') in public static String calculateCost(). As the value of output ="" is never used and also it is later redefined to a different string value on lines 34 and 36 depending on how the decision on line 33 evaluates, this error is thrown as Unused Assignment.

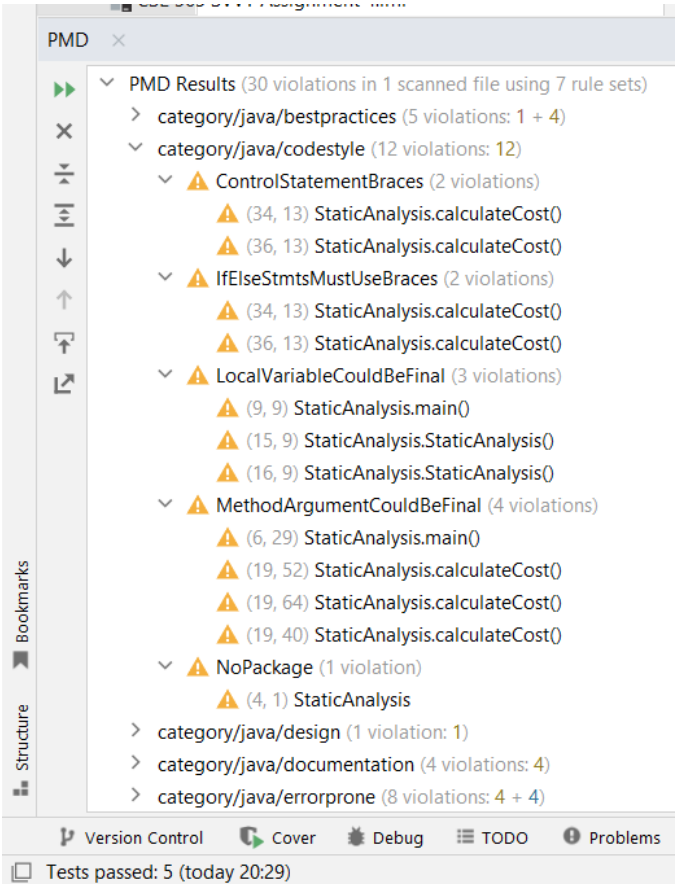


Figure 15: PMD RESULTS 2

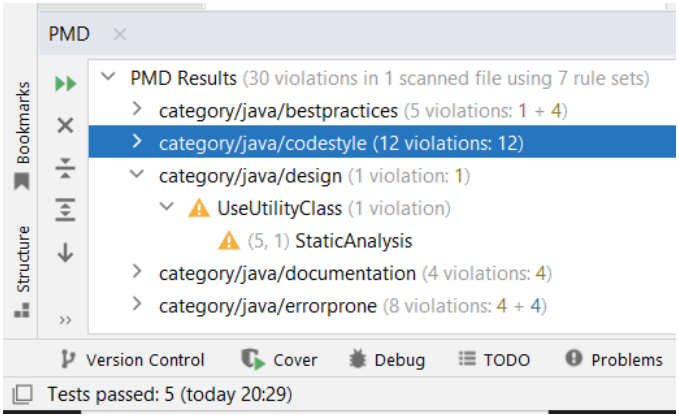


Figure 16: PMD RESULTS 3

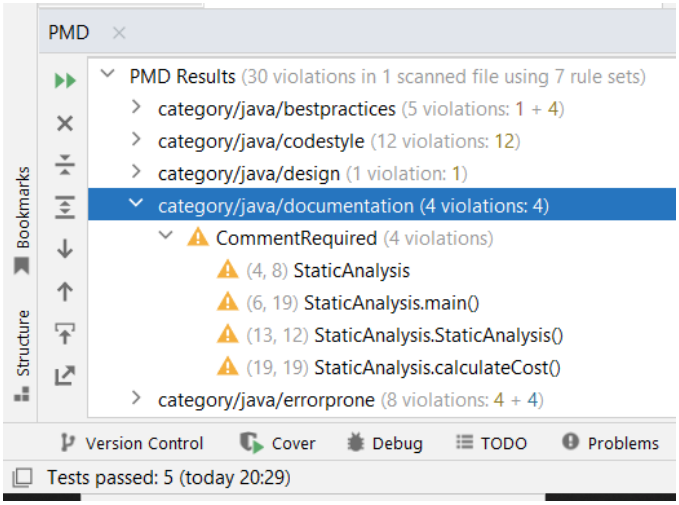


Figure 17: PMD RESULTS 4

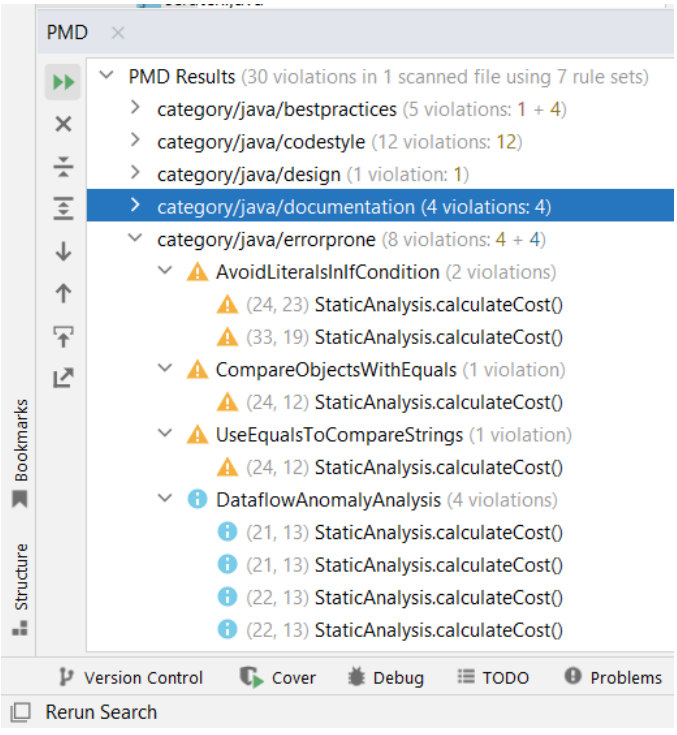


Figure 18: PMD RESULTS 5

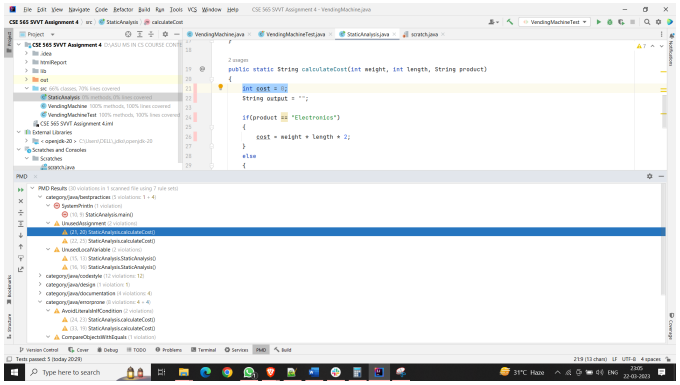


Figure 19: Data Anamoly 1 - UnusedAssignment - int cost = 0;

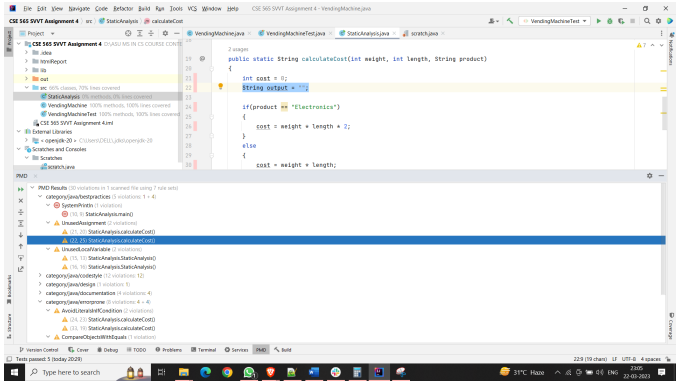


Figure 20: Data Anamoly 2 - UnusedAssignment - String output = "";

Lessons Learned

This project has provided an opportunity to explore and evaluate additional testing and analysis techniques and their associated tools for improving software development effectiveness and efficiency. From this project, several key lessons have been learned that can be beneficial in future software development projects.

Firstly, selecting an appropriate Integrated Development Environment (IDE) is crucial to achieving the desired testing and analysis outcomes. The choice of IDE should consider the task's scope, purpose, and available resources. In this project, IntelliJ IDEA and its built-in Code Coverage Runner are practical tools for measuring code coverage in a single file, with the latter being less resource-intensive. By utilizing online resources and dependable articles, I gained a deeper understanding of the tool's scope and purpose and its different types of code coverage analysis.

Secondly, I learned that developing a comprehensive set of test cases is a time-consuming task that requires careful consideration of the code's structure and logic. Creating a set of comprehensive test cases is essential to achieve the desired code coverage. During the project, I had to create test cases to cover all 24 lines/statements and the 15 decision branches of the VendingMachine.java code file. It took mul-

tiple attempts to develop a set of five test cases that achieved the desired code coverage of 100% statement coverage and at least 90% decision coverage. I also learned the importance of documenting each test case's scope and coverage in the report to demonstrate the effectiveness of the testing process.

Thirdly, I discovered the importance of static source code analysis in identifying potential bugs or issues in the code. I have learned that static source code analysis tools such as PMD can help identify coding issues that manual code reviews may miss. These tools can be helpful in improving code quality and reducing the number of bugs in software projects.

This project has provided a valuable learning experience and an opportunity to apply testing and analysis techniques to real-world code and gain practical experience with testing and analysis tools, creating comprehensive test cases, and evaluating tools' effectiveness and efficiency. The lessons learned from this project can benefit future software development projects, enabling developers to create high-quality software effectively and efficiently.

Assignment 5

This project aims to demonstrate the use of graphical user interface (GUI) testing techniques and tools. GUI testing is a software testing technique that focuses on the user interface of an application. In this project, we will select an application with a graphical user interface, make changes to the GUI, and use a GUI test automation tool to test the application. This project has two tasks.

For Task 1, this project uses a graphical user interface (GUI) testing technique and a GUI testing tool for a software application. The project requires creating any application with a GUI, which can be coded in any programming language and has at least three GUI pages with a flow from one page to another. Each page should have GUI elements such as images, buttons, labels, text boxes, and two of the following: scroll-bar, check-boxes, radio button, slider, list, and drawing elements. Two versions of the application should be created, where the first version will serve as a baseline for the second version, which will include at least three changes in the GUI, such as the size, location, and orientation of the GUI elements. Additionally, a change in the flow of pages, the link from one page to another, will also be included in the second version.

For Task 2, this project requires selecting a GUI test automation tool to record test cases and report the test results. Test cases should be developed to test the existence of GUI elements, the correctness of their location and size, the content of the GUI element, and the correctness of the link from one page to another. The test cases developed for the first version of the GUI will be executed for the second version, and the test results will be reported, highlighting the number of test cases that passed or failed.

As part of this project, a report will provide a detailed description of the application with screenshots of the GUI for version one and version two and a description of the selected GUI test automation tool, including its features, scope, and area of usage. The test cases developed for the two versions of the GUI will be explained, highlighting the coverage of the test cases. The test results after the execution of the test cases will be explained, with references to screenshots in the report. Additionally, the report will include an assessment of the GUI testing tool in terms of its usability, coverage, and features. Overall, this project aims to showcase the importance of GUI testing and demonstrate the use of GUI testing tools for ensuring the quality and reliability of software applications with a graphical user interface.

Work Completed

I have made a web application using HTML and CSS for a user to register, log in, and give a review of my application. To begin with, I salvaged the underlying code framework for a login form from a GitHub repository[23] and then duplicated the same form to create the two additional pages for sign-up and review. Following this, I have changed each page to have GUI elements such as images, buttons, labels, text boxes, and two of the following: scroll-bar, check-boxes, radio button, slider, list, and drawing elements to meet the project requirements.

The first page in my application is the Sign Up form, where the user can register by providing specific information such as their username, email, phone number, gender, age, and password for the account. Per the project requirements, each page should have GUI elements such as images, buttons, labels, text boxes, and any of the following: scroll-bar, check-boxes, radio button, slider, list, and drawing elements. Therefore, the Sign-Up form has a small user avatar image at the top and the title "Create Account" just below the image. The sign-up form includes text input boxes for username, email, phone number, password, and confirm password fields. The form contains a group of radio buttons rendered as a list for the gender field, allowing the user to select their sex, and a slider for the age field, allowing the user to set their age by sliding the rounded element left and right. There are also two buttons on the sign-up form: The sign-Up Button and the Login Button. Now, to implement the navigation flow on this page, I have used the two created buttons mentioned previously. When the user clicks on the Sign-Up button, the application will create an account for the user and reroute them to the login page immediately so that the user can log in to the application and leave feedback on the review form page. Similarly, when the user clicks on the Login Button, the application will reroute them to the login page immediately so that the user can log in to the application and leave feedback on the review form page.

The second page of the application form is the Login Form, where the user can log in by entering their username, email, and password. Per the project requirements, each page should have GUI elements such as images, buttons, labels, text boxes, and any of the following: scroll-bar, check-boxes, radio button, slider, list, and drawing elements. Therefore, similar to the sign-up form, the login form has a small user avatar image at the top and a title "Login" just below the image. Apart from the image, title, and input fields for login credentials, the login form has four other components: Scroll Box with a Label, Radio Button with a Label, Sign Up Button, and the Login Button. The scroll box

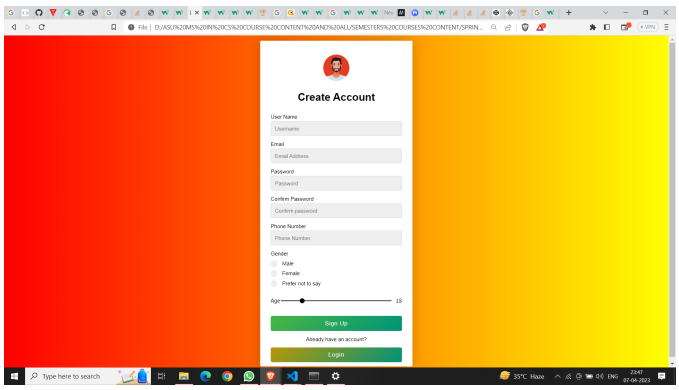


Figure 21: Form Page 1 : Registration / Sign Up Form

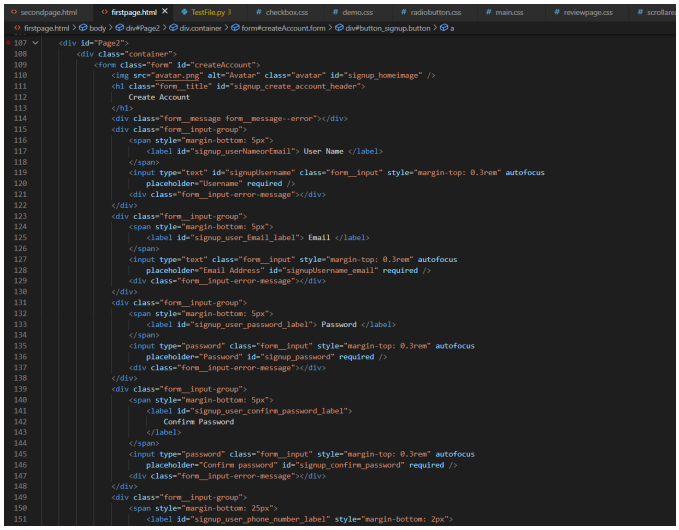


Figure 22: Code Part 1 for the Sign Up Form

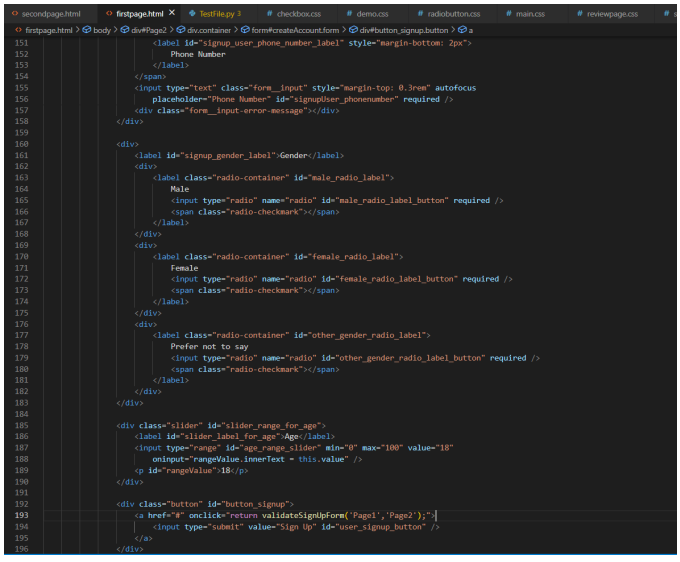


Figure 23: Code Part 2 for the Sign Up Form

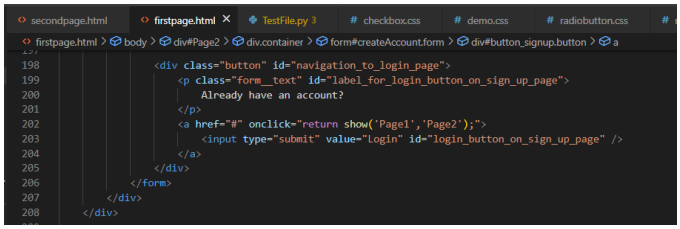


Figure 24: Code Part 3 for the Sign Up Form

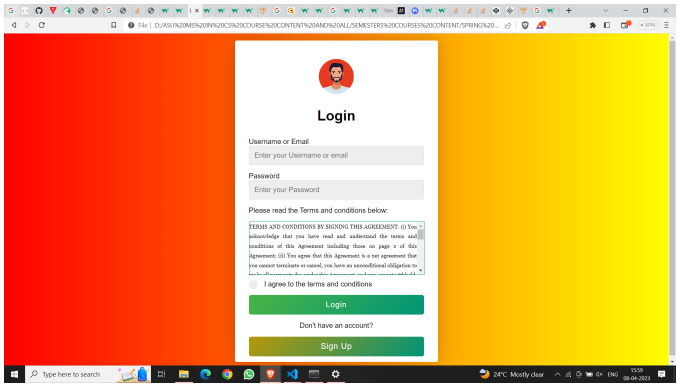


Figure 25: Form Page 2 : Login Form

is configured to match the form's width and a predefined height, allowing the user to quickly scroll through the terms and conditions before logging in to the application. A label "Please read the terms and conditions below" just above the scroll box makes the user aware of the information specified within the scroll area. This radio button is included in the form because the user can only log in if he or she agrees to the application's terms and conditions. To do so, the user must select the radio button with the label "I agree to the terms and conditions," which informs the application that the user has accepted the terms and conditions of the application. The Sign-Up button allows the user to navigate to the Sign-Up form page to create an account for themselves if they do not have an existing account. In contrast, if the user clicks the login button, the application validates the form to see if they have entered their credentials and checked the radio button, and if these validations pass, the application will proceed to the review form page, where the user can provide feedback.

The third page of this application is a review form. I created a review page after being inspired by the comments section of the ASU Canvas Grades Section. In this feedback form, I ask the user to rate this web application based on its UI, functionality implementation, and content. Per the project requirements, each page should have GUI elements such as images, buttons, labels, text boxes, and any of the following: scroll-bar, check-boxes, radio button, slider, list, and drawing elements. Therefore, similar to the first two forms, the feedback form has a relevant image at the top and the title "We appreciate your review!" just below the image. Nevertheless, the feedback form also has a subtitle, "Your review will help me to improve in my future assign-

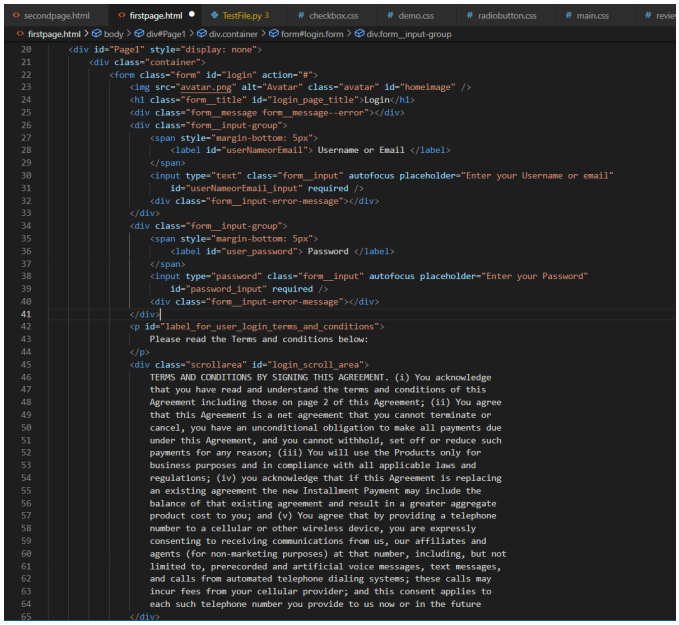


Figure 26: Code Part 1 for the Login Form

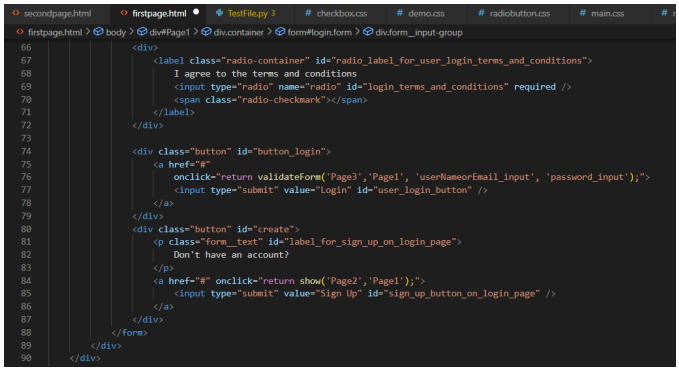


Figure 27: Code Part 2 for the Login Form

ments," written below the title to inform the user why this review or feedback is important to us. Furthermore, again, like the first two forms, this feedback form too has two input text boxes for the username and email fields. The form then has a kind of rating header label, "Please rate your service experience for the following parameters," It asks the user to rate the applications for the following factors as they see fit. I have included 5 rating sections where each section has different GUI elements taking the rating from the user. The first two rating sections ask the user to rate the application for the following factors "Selection of the testing tool for this assignment?" and "Assessment of the submitted report in terms of the set of grammar or formatting errors, information about the selected testing tool and the description of the Test Cases executed as part of this assignment?". Both these rating sections contain a star rating format, which corresponds to the usage of drawing elements requirement of the GUI. The third rating section asks the user to rate the application for the following factor "How did you like the

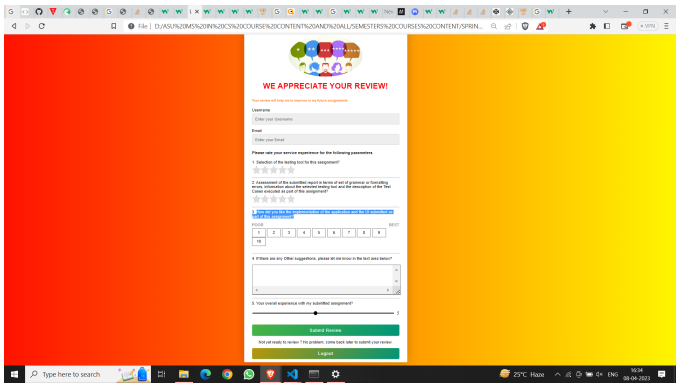


Figure 28: Form Page 3 : Feedback / Review Form

implementation of the application and the UI submitted as part of this assignment?". Therefore, for this section, I have implemented a scale rating format that contains a group of radio buttons rendered side by side. The fourth rating section consists of a simple text area box in which the user can provide personal feedback on what they liked about the application and what could have been done better regarding the web application as a whole, the test cases developed, and the submitted report. Furthermore, for the last rating section, I have included a slider that asks the user to rate their overall experience with the web application, test cases developed, and report submitted on a scale of 1 to 5. Finally, to finish the Review form, I had to include it in the navigation flow to fulfill the GUI requirements. Therefore, I have added two buttons: Submit Review Button and the Logout Button, when the user can click the Submit Review button after providing all the feedback inputs. The review is submitted, the user is logged out when you click this button, and the application returns to the login page. Similarly, If the user still needs to be ready to leave feedback, he or she can click on the Logout button, redirecting the user to the login page, where he or she can log in once ready to leave feedback.

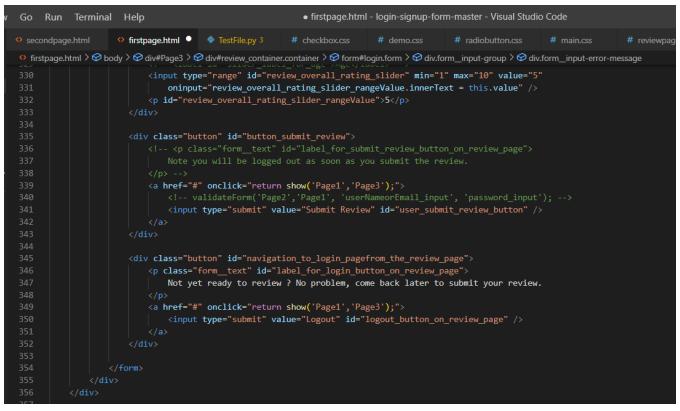


Figure 29: Code Part 4 for the Feedback Form

Moving on to the next phase of Task 1, using the first version of the application created above as a baseline for the second version, I have made at least three changes in the

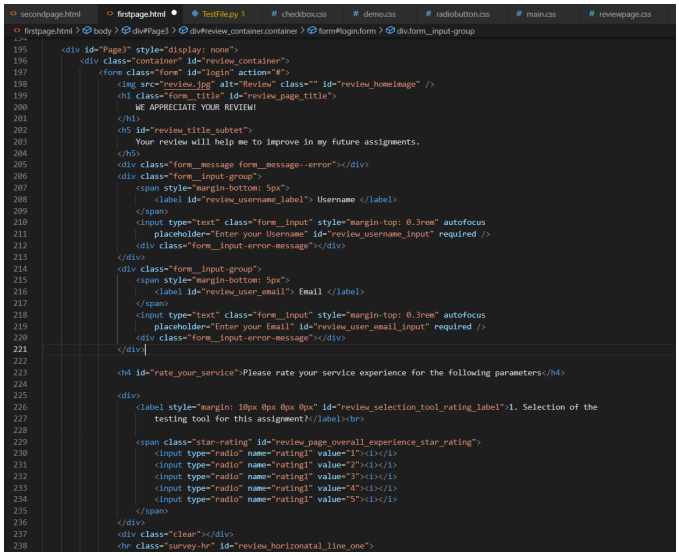


Figure 30: Code Part 1 for the Feedback Form

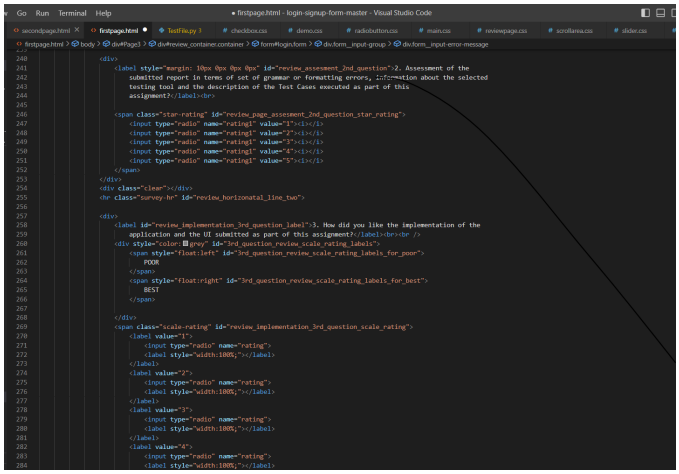


Figure 31: Code Part 2 for the Feedback Form

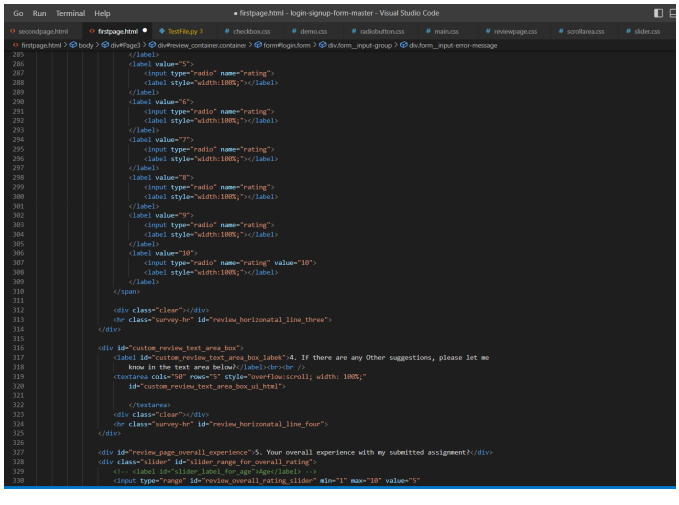


Figure 32: Code Part 3 for the Feedback Form

GUI, such as the size, location, and orientation of the GUI elements. Additionally, I have changed the flow of pages, the link from one page to another. I have mentioned the changes I have made on each page in detail below.

The Sign Up form is the first page of the application where the user can register their account by providing specific information such as their username, email, phone number, gender, age, and password. The first changes made to this form are as follows:

- **Image** : Initially, the sign-up form had the small user avatar image at the top and a title "Create Account" just below the image. But in this updated GUI, two changes have been made:
  - **Location of the Image** : The title "Create Account" has been pushed to the top, and the image has been pushed to be placed below the title.
  - **Size of the image** : The height and width CSS attributes have been updated to make the image more prominent, as shown in the image below.
- **Orientation of the Input Text Box Fields** : All fields were initially implemented in a downward flow. However, four input fields, namely the username, email, password, and confirm password, have been placed in groups of two in the updated GUI.
- **Width of the form**: The form's width was set initially to 400px, but in the updated GUI, the width of the entire form is set to 600px, which is why the form's width is visible in the image attached below.

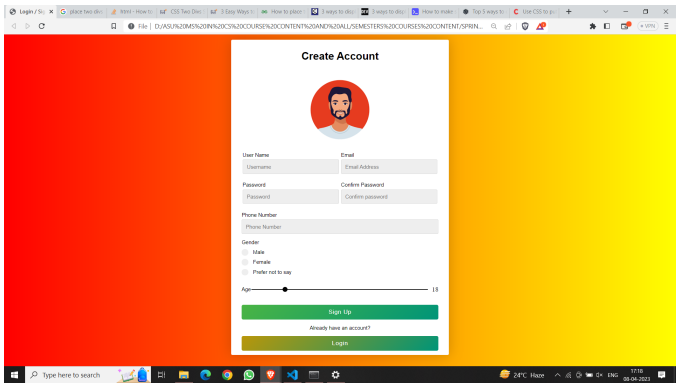


Figure 33: Updated Page 1 : Registration / Sign Up Form

The Login Form is the second page of the application form, where the user can log in by entering their username or email address and password. The changes made to this form are as follows:

- **Image** : Initially, the login form had the small user avatar image at the top and a title "Login" just below the image. However, in this updated GUI, two changes have been made:
  - **Location of the Image** : The title "Login" has been pushed to the top and the image has been pushed to be placed below the title.

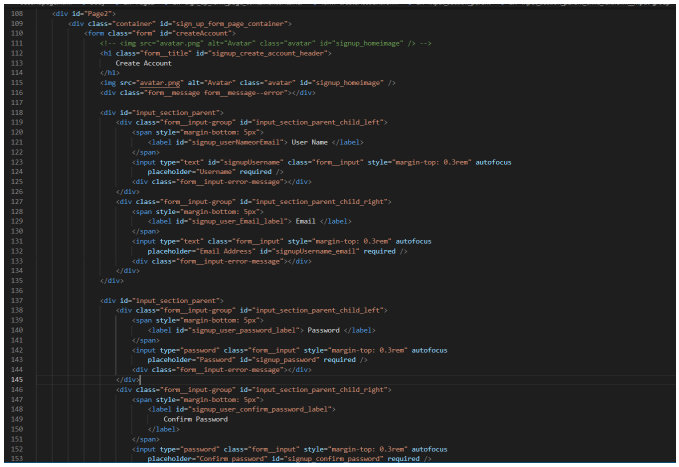


Figure 34: Code Part 1 for the Updated Sign Up Page

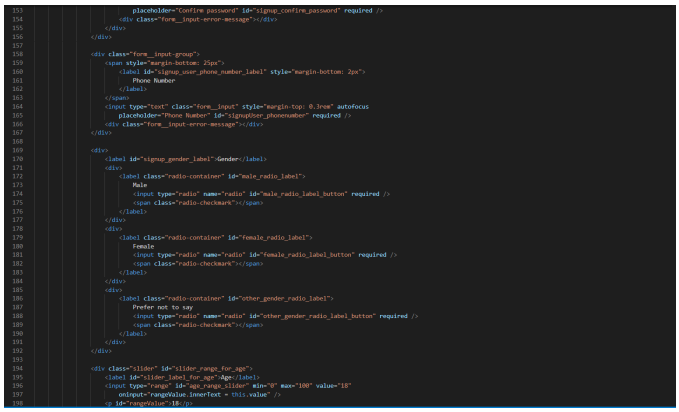


Figure 35: Code Part 2 for the Updated Sign Up Page

Figure 36: Code Part 3 for the Updated Sign Up Page

  - **Size of the image** : As shown in the image below, the height and width CSS attributes have been updated to make the image larger.

- **Spacing between the Labels and Input Text Boxes** : I have added a “margin-top” CSS property to the input field text boxes to add some spacing between them and their respective labels.

- **Scroll Area Box** : The height of the Scroll Area Box has increased compared to its initial height.

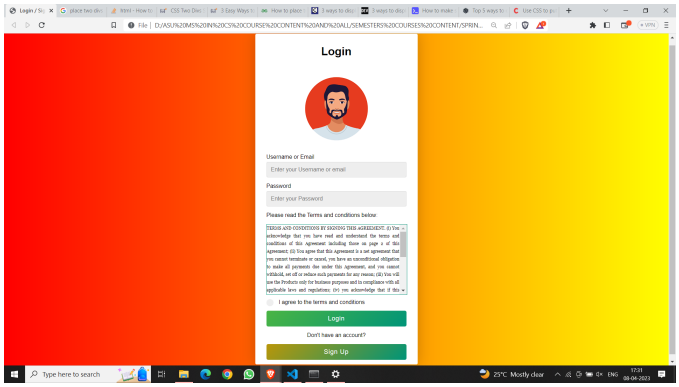


Figure 37: Updated Page 2 : Login Form

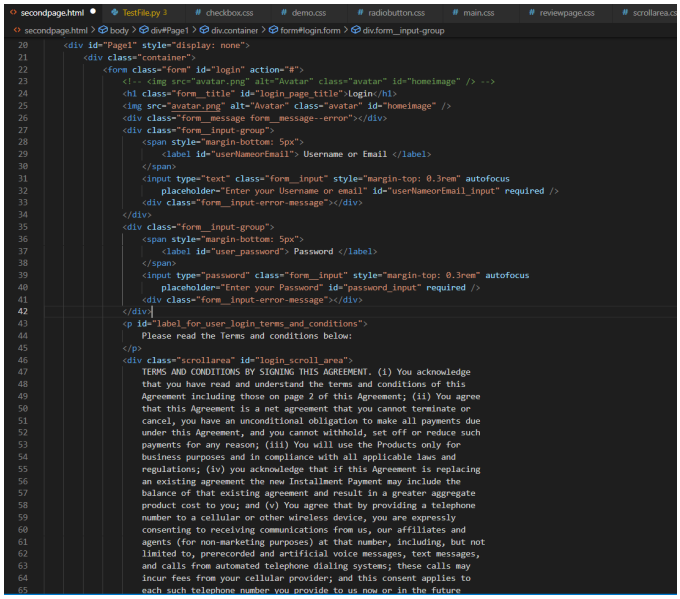


Figure 38: Code Part 1 for the Updated Login Page

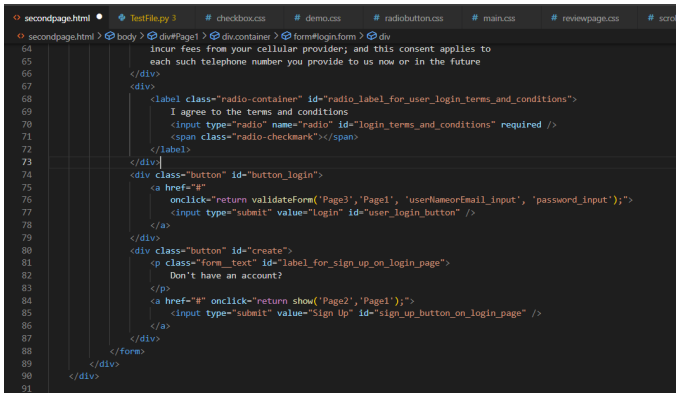


Figure 39: Code Part 2 for the Updated Login Page

The third page of this application is a review form. I created a review page after being inspired by the comments section of the ASU Canvas Grades Section. In this feedback form, I ask the user to rate this web application based on its UI, functionality implementation, and content. This form has undergone the following modifications:

- **Third Rating Section**: I have done three changes in this section.
  - **Interchanged the labels**; I have interchanged the positions of the POOR and the BEST labels.
  - Furthermore, to match the labels, I have also reordered the rating values from 10 - 1 instead of 1 - 10.
- **Text Area Box** : I increased the number of rows for the text area box, which increased its height compared to its initial height.
- **Flow Change**: Since it was stated in the assignment description that one flow change was required, instead of navigating to the Login page as was implemented initially, when the user clicks on the Logout button at the bottom, the application will now navigate to the Sign Up page.

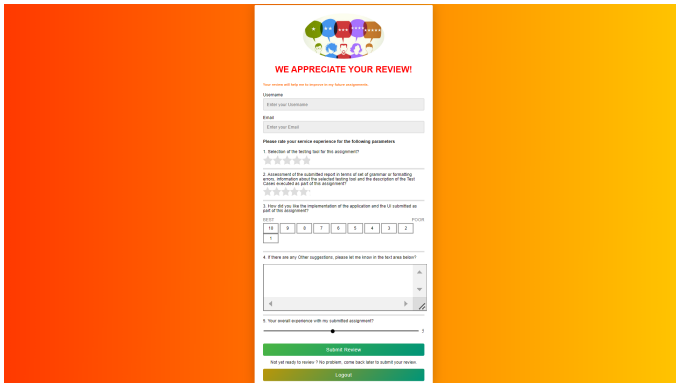


Figure 40: Updated Page 3 : Feedback / Review Form

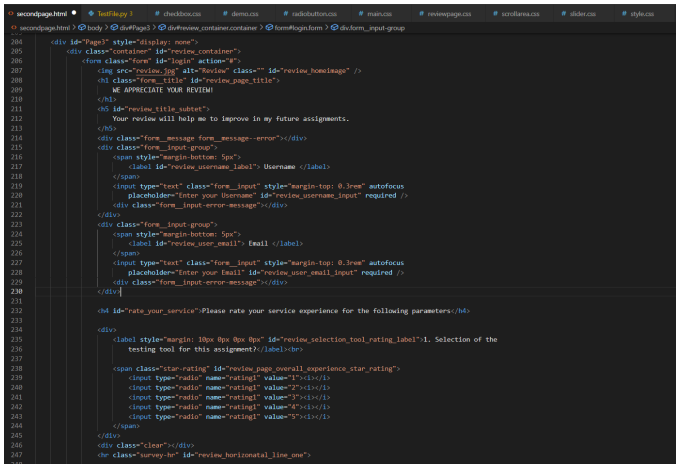


Figure 41: Code Part 1 for the Updated Review Form

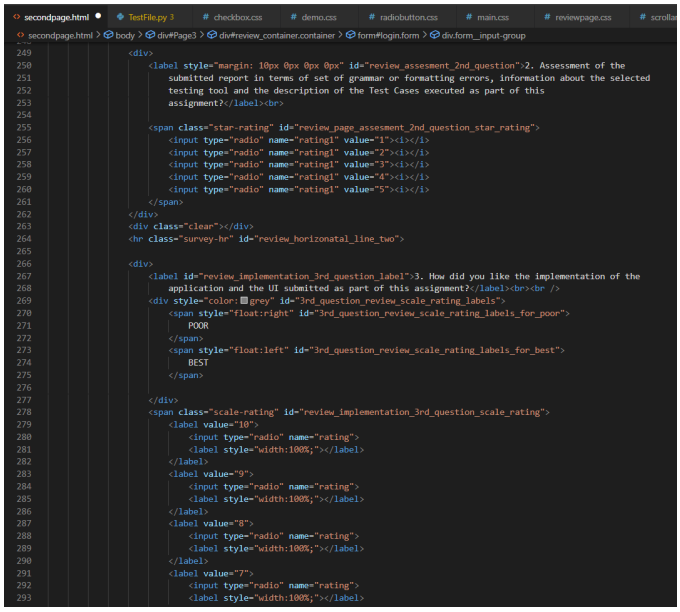


Figure 42: Code Part 2 for the Updated Review Form

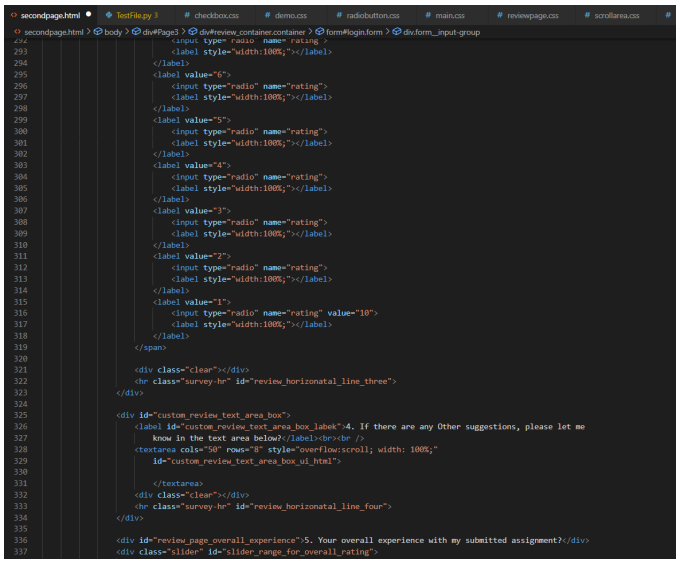


Figure 43: Code Part 3 for the Updated Review Form

For Task 2 of the project, selecting a suitable GUI test automation tool was required to record test cases and report the test results. I have scoured the internet for articles[24, 25, 26, and 27] and information about the different GUI test automation tools available in the market. After extensive research on different GUI test automation tools available in the market, it was concluded that Selenium would be the best choice for this project.

Selenium is a popular open-source library for automating web browsers, including web application testing, scraping, and repetitive task automation. The Selenium tool provides a set of libraries and tools to simulate user interac-

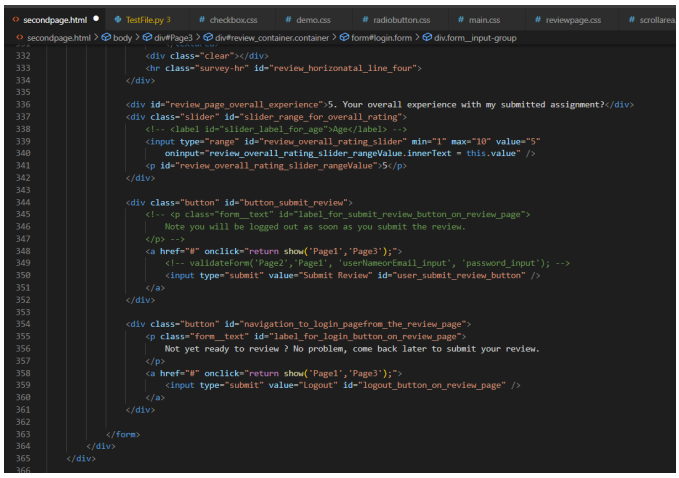


Figure 44: Code Part 4 for the Updated Review Form

tions with web browsers, which can help test web applications and other tasks such as web scraping and web automation. The key features of Selenium, such as programming language support, browser support, fast test execution, parallelism, and record and playback features, were considered. However, the most attractive factor of Selenium was the Selenium Web-Driver. Selenium Web-Driver is a tool for automating web application testing using a programmatic interface to control a web browser. This means that developers and testers can use Selenium Web-Driver to create functional tests that automate browser actions like clicking buttons, filling out forms, and navigating between pages, as well as a set of APIs for interacting with web browsers like Chrome, Firefox, and Safari. Therefore, Selenium Web-Driver was an ideal tool for testing this GUI application as the GUI contains elements such as images, buttons, labels, text boxes, scroll-bar, check-boxes, radio buttons, slider, list, and drawing elements.

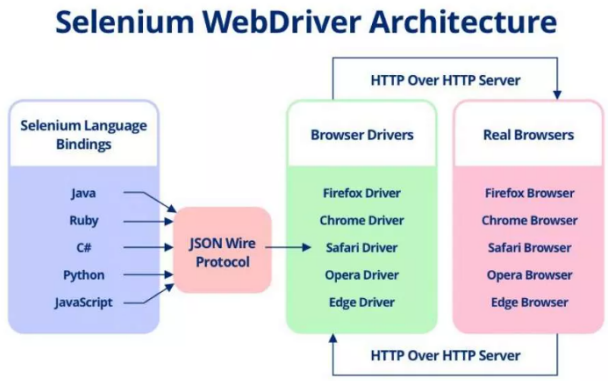


Figure 45: Selenium Web-Driver architecture

To fully understand the Selenium Web-Driver tool, the official documentation[28] and other reliable websites[29, 32, and 33] were consulted, and the scope, purpose, and usage of Selenium Web-Driver were studied. It was found that the scope of Selenium Web-Driver is to automate functional tests that verify the behavior of a web application, while the primary purpose of Selenium with Python is to automate web testing by controlling the browser and making assertions about the web application's behavior. Finally, after a thorough understanding of the tool, a working example of Selenium Web-Driver was included in the report with a detailed explanation of how the tool works and how it meets its scope and primary purpose. This example demonstrates how Selenium Web-Driver can automate the user login process with a web browser and verify the expected behavior of the web application. We can run this test by using Selenium Web-Driver to automate it repeatedly to ensure that the login functionality is working correctly without manual testing.

In order to perform UI testing using Selenium, it was necessary to set up the appropriate tools and dependencies. This involved installing Selenium, an IDE, and a browser driver. Firstly, I visited the official download page[30] for Selenium and selected Python as the appropriate language binding for this project. The reason for selecting Python was because it is a popular and influential programming language, and it has excellent support for Selenium testing. With Python selected, I downloaded the appropriate version of Selenium for my system.

Next, I ensured an IDE was installed for writing and executing the test cases. I used Visual Studio Code (VSCode) for this project, which I had already installed on my system. IDE was chosen because it is a robust and feature-rich text editor that offers a range of features to aid development, such as syntax highlighting, code completion, and debugging tools for various programming languages, including Python. It also provides an extensive range of plugins and extensions that can enhance the development process, such as auto-complete and debugging features.

Finally, I needed a browser driver to drive real browsers with automated UI testing scripts. As mentioned in the previous section, different browser drivers are specific to different browsers. However, for this project, I will only test this project in Chrome. The reason for selecting Chrome-Driver was that it is an open-source tool that provides excellent support for automating the testing of web applications and works seamlessly with Selenium Web-Driver. Therefore, I downloaded Chrome-Driver from the official website[31].

With all the dependencies installed, I created the Python file for testing. Following all the steps in detail mentioned in the official documentation[28] and other reliable websites[34, 35, and 36] to setup the project for testing, I first imported all the required dependencies within the Python file, including Selenium and Chrome-Driver. I then loaded the two HTML files containing the two versions of the GUI application we had created earlier. I could interact with the GUI elements by loading these files into the script and performing tests using the Selenium Web-Driver. With the initial setup completed, I was able to begin creating test cases following each every step mentioned in the official documentation[28] and other reliable websites[29, 34, 35, and 36] and to cover all aspects of the GUI application, which includes its correct location, the size of the element, the

content of the GUI element, and the link's correctness (flow from one page to another).

As a result of these guidelines, I primarily created three types of Test Cases.

- **Category 1**: This category of test cases checks for the **existence, location and the size** of the element using either it's unique ID or CLASS_NAME.
  - The command **"l = browser.find_element(By.ID, "signup_homeimage")"** checks for the existence of the element.
  - The commands **"location = l.location"**, **" x = location.get("x")"** and **" y = location.get("y")"** retrieve the x and y coordinates of the element.
  - The commands **"size = l.size"**, and **"w, h = size["width"], size["height"]"** retrieve the width and the height of the element.
- **Category 2**: This category of test cases checks for the **Content of the GUI element** using either it's unique ID or CLASS_NAME.
  - This is possible with the help of the command **"s = l.text"**. This command is used to get the label content of the GUI element.
- **Category 3**: This category of test cases checks for the **Flow of the GUI** from one link to another using either it's unique ID or CLASS_NAME.
  - This is implemented by using the command **"browser.find_element(By.ID, "user_signup_button").click()"**.
  - The application is then navigated to the next page after clicking a button, so an additional code line is implemented to check for an element present on the next page to validate whether or not the application has indeed navigated.

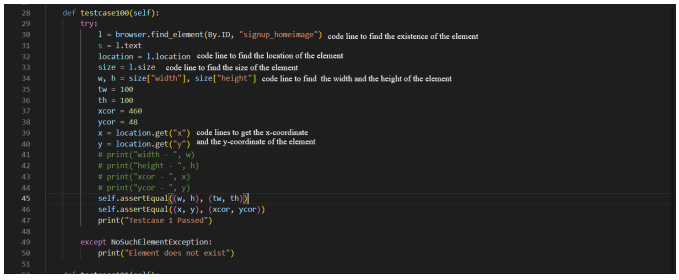


Figure 46: Code Example of the Test Case Category 1

Figure 47: Code Example of the Test Case Category 2

To end the report, as per the project requirements, I have provided a detailed evaluation of Selenium and the Selenium

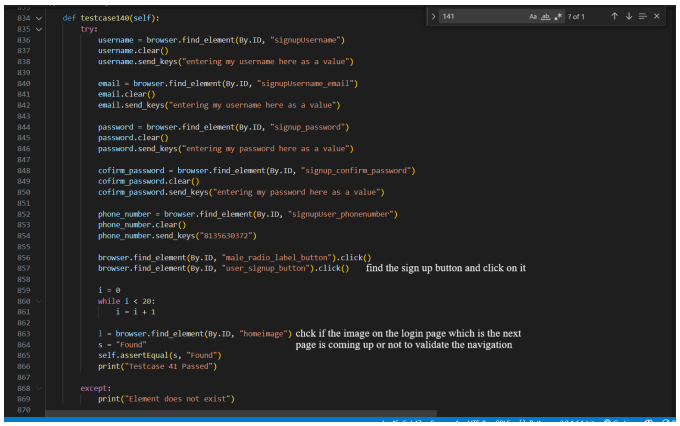


Figure 48: Code Example of the Test Case Category 3

Web-Driver tool in terms set of features and functionalities provided, type of coverage, reuse of test cases, test results produced, ease of usage, and type of GUI elements that can be tested. After working with Selenium and the Selenium Web-Driver tool firsthand for this project, I have gained a deeper understanding of the tool and its capabilities.

In terms of features and functionalities, Selenium provides a wide range of features for automating web application testing, such as browser automation, web element interaction, and various assertion methods for verifying expected behavior. The Selenium Web-Driver tool, in particular, provides a set of APIs for interacting with web browsers and simulating user interactions with web elements. Additionally, Selenium supports multiple programming languages, making it a versatile tool for test automation.

Regarding the type of coverage, Selenium Web-Driver is ideal for functional testing and can be used to verify the behavior of web applications. Selenium also supports unit testing frameworks, making it suitable for integration and regression testing.

One of the significant benefits of using Selenium Web-Driver is the ability to reuse test cases. As Selenium Web-Driver is a programmatic interface, test cases can be written in a modular and reusable way, allowing them to be run on multiple browsers and operating systems. This feature of Selenium makes it an efficient tool for web application testing, saving time and resources by reducing the effort required in creating new test cases for each browser.

Selenium produces comprehensive test results, including pass/fail status, screenshots of failures, and log files performance metrics. The test results produced by Selenium are highly detailed and informative, making it easy to interpret and analyze, as Selenium provides detailed logs and reporting capabilities. These results provide insight into the application's behavior and can be used to identify and diagnose issues in the web application. The test results are provided in various formats, such as XML, HTML, and JSON, making it easier for testers to analyze and interpret the results. Additionally, Selenium provides visual confirmation of test steps, making it easier to track test progress and locate errors.

In terms of ease of usage, Selenium Web-Driver has a rel-

atively steep learning curve, especially for beginners unfamiliar with programming languages, as it depends on the user's experience with programming languages. However, the official Selenium documentation provides a comprehensive guide on how to use the tool. Additionally, the Selenium community provides ample resources for troubleshooting and support. Therefore, once familiar with the tool, writing test cases can be straightforward and efficient.

Selenium Web-Driver can test various GUI elements, including images, buttons, labels, text boxes, scroll-bars, check-boxes, radio buttons, sliders, lists, and drawing elements. This makes it a versatile tool for testing various web applications.

Based on my experience with the tool, Selenium Web-Driver is a powerful and versatile tool for automating web application testing. Its extensive features and functionalities, type of coverage, ability to reuse test cases, complete test results, ease of usage, and type of GUI elements that can be tested make it an efficient and effective tool for web application testing.

Number of Developed Test Cases

For the whole application, I have developed a total of 109 test cases based on the initial GUI.

Test Cases Results for the Version-1 GUI

All the 109 test cases have successfully passed after a successful execution of the test cases.

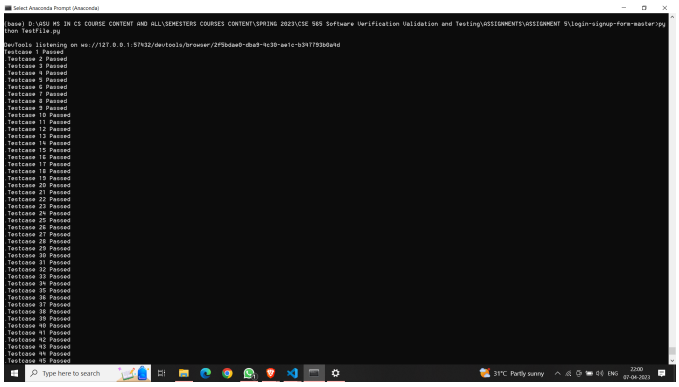


Figure 49: Results of Test Cases on the Version-1 GUI - 1

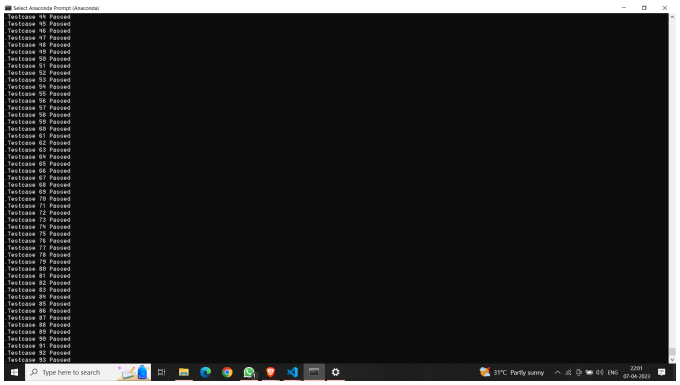


Figure 50: Results of Test Cases on the Version-1 GUI - 2

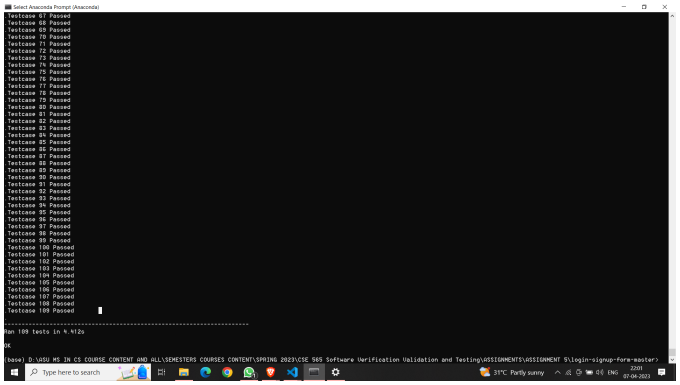


Figure 51: Results of Test Cases on the Version-1 GUI - 3

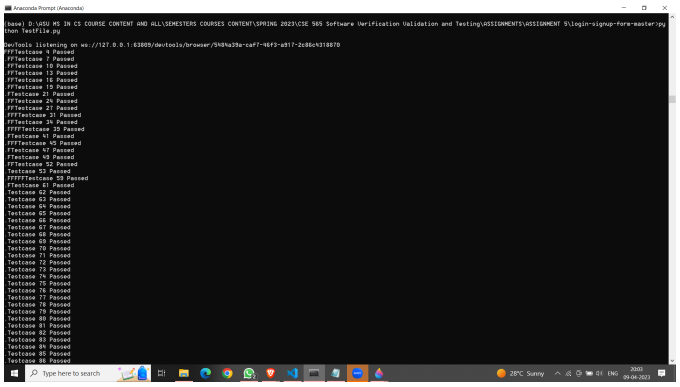


Figure 52: Results of Test Cases on the Version-2 GUI - 1

Test Cases Results for the Version-2 GUI

Out of the developed 109 test cases, 55 test cases failed when executed on the updated GUI.

Understanding of the Results of the Test Cases for the Updated GUI

- **Test Cases with respect to the Sign Up Page**: For the Sign-Up page, 41 test cases have been created. Because of the changes made to this page, such as changing the position of the image and title and changing the format of the form, all of the test cases about location and size, totaling 28 cases, have failed. Twelve of the thirteen test cases passed because they were checking for the content of the GUI elements, and since I had not changed any of the element's content, all of them passed. Finally, the last test case that passed was about the GUI flow, and it passed because the flow from the Sign Up page to the Login page was not hampered or modified in this case.
- **Test Cases with respect to the Login Page**: For the Login page, 20 test cases have been created because of the changes made to this page, such as changing the position of the image and title, as well as adding some space

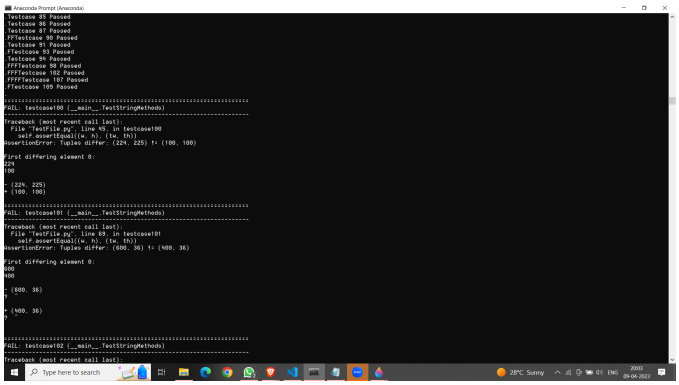


Figure 53: Results of Test Cases on the Version-2 GUI - 2

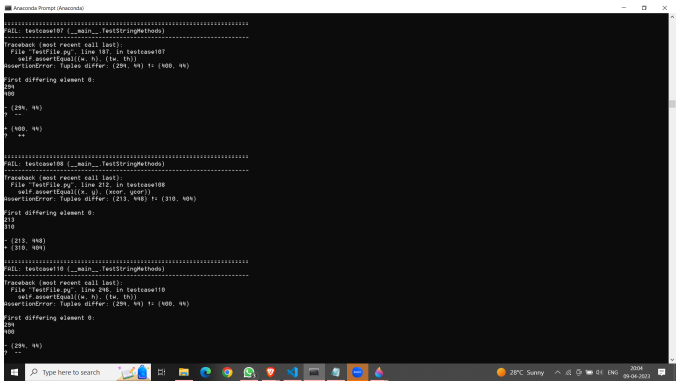


Figure 55: Results of Test Cases on the Version-2 GUI - 4

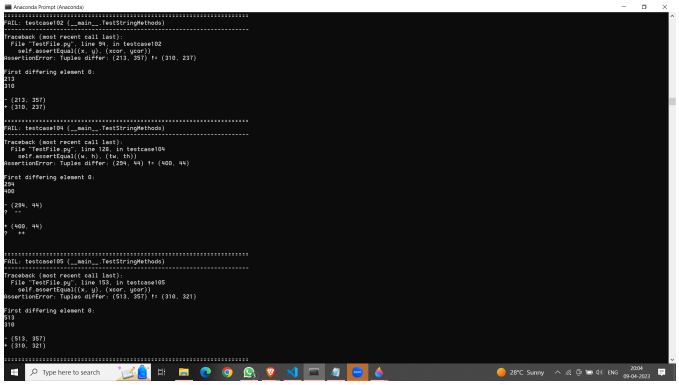


Figure 54: Results of Test Cases on the Version-2 GUI - 3

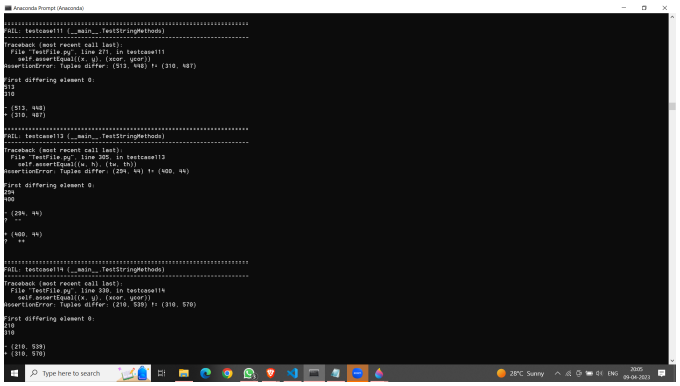


Figure 56: Results of Test Cases on the Version-2 GUI - 5

between the labels and fields, all of the test cases about location and size, totaling 13 cases, have failed. Six of the seven test cases passed because they were checking for the content of the GUI elements, and since I had not changed any of the element's content, all of them passed. Finally, the last test case that passed was about the GUI flow, and it passed because the flow from the Login page to the Feedback form page was not hampered or modified in this case.

- **Test Cases with respect to the Feedback Form Page**: The Feedback Form page has 49 test cases. However, because I did not change the position of the image at the top of this page, all subsequent location and size test cases did not fail and instead passed. The failed test cases are associated with the "POOR" and "BEST" labels, which were relocated. Following these test cases, the text area box test case failed because its height was increased. As a result, some of the following 11 test cases concerning location and size failed since I have changed the flow where when the user submits the feedback form, the application is rerouted to the sign-up page instead of the login page per the initial implementation.

As a result, we have 55 failing test cases, most related to location and size, except for one that failed due to a change in the application's flow. Only two of the 54 passed test cases check for the flow of the GUI application, while the majority

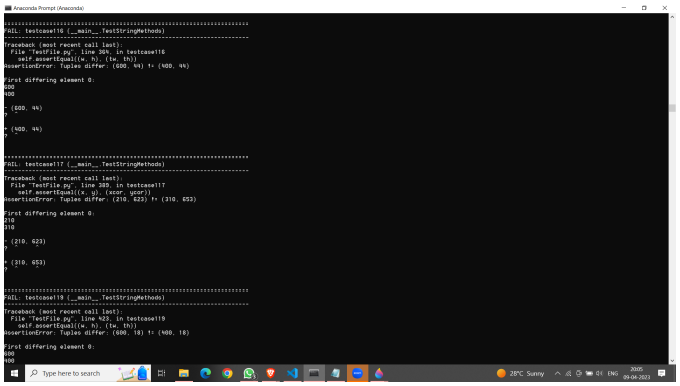


Figure 57: Results of Test Cases on the Version-2 GUI - 6

of the 54 passed test cases check for the content of the GUI elements.

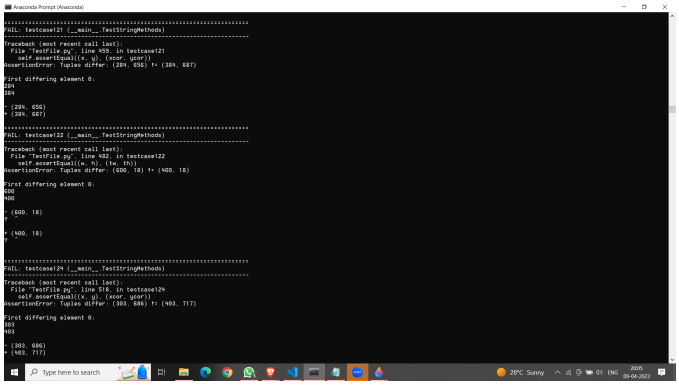


Figure 58: Results of Test Cases on the Version-2 GUI - 7

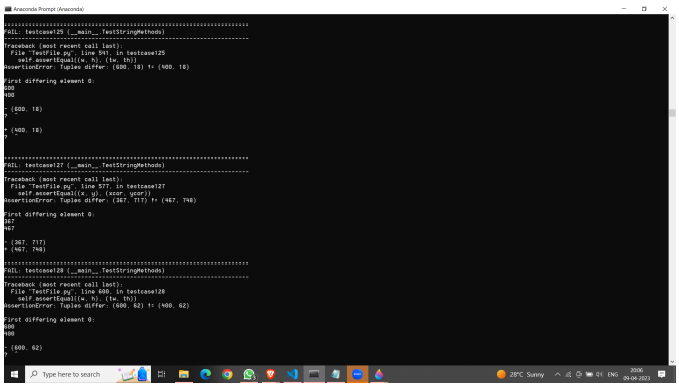


Figure 59: Results of Test Cases on the Version-2 GUI - 8

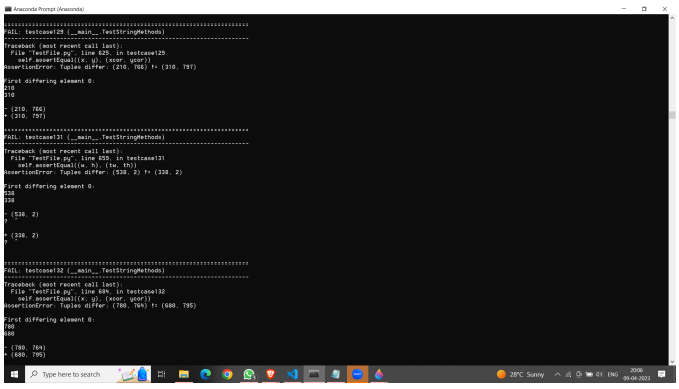


Figure 60: Results of Test Cases on the Version-2 GUI - 9

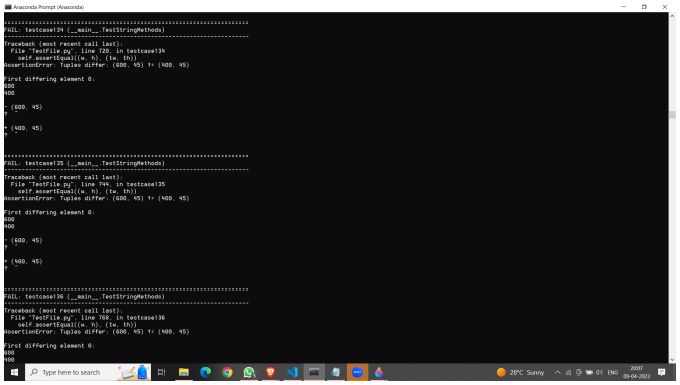


Figure 61: Results of Test Cases on the Version-2 GUI - 10

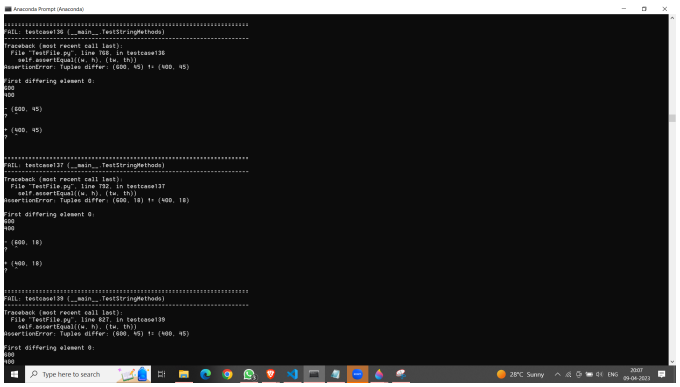


Figure 62: Results of Test Cases on the Version-2 GUI -11

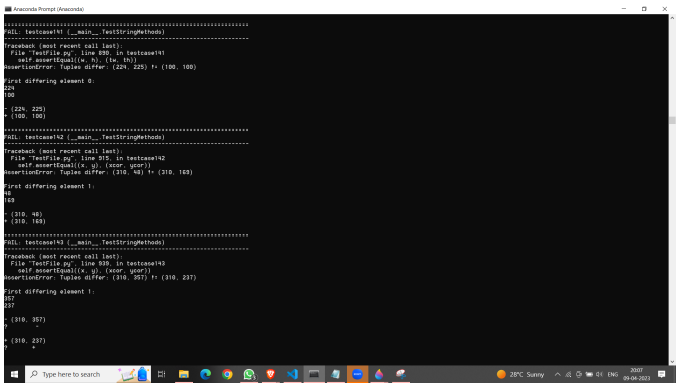


Figure 63: Results of Test Cases on the Version-2 GUI - 12

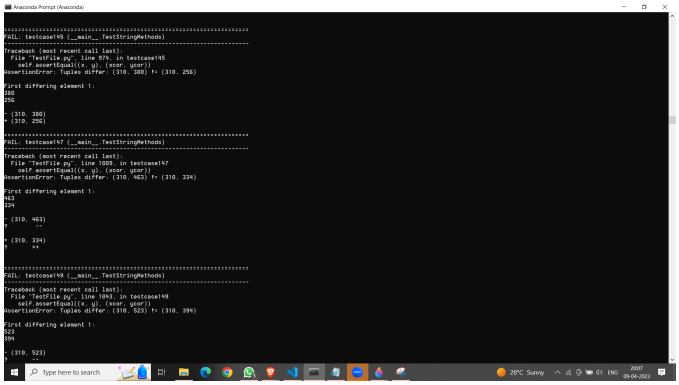


Figure 64: Results of Test Cases on the Version-2 GUI - 13

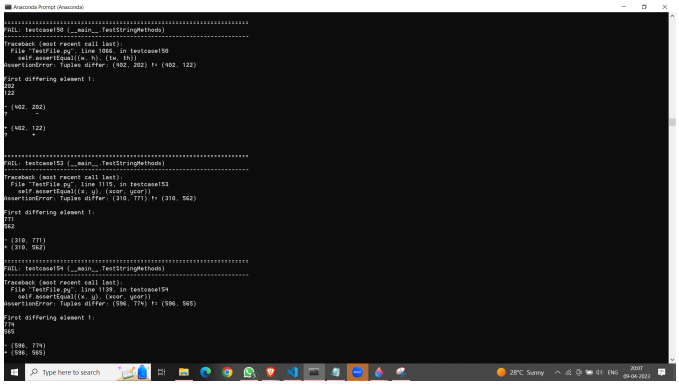


Figure 65: Results of Test Cases on the Version-2 GUI - 14

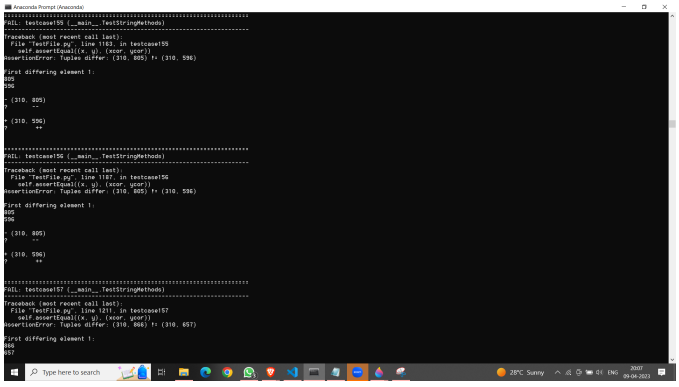


Figure 66: Results of Test Cases on the Version-2 GUI - 15

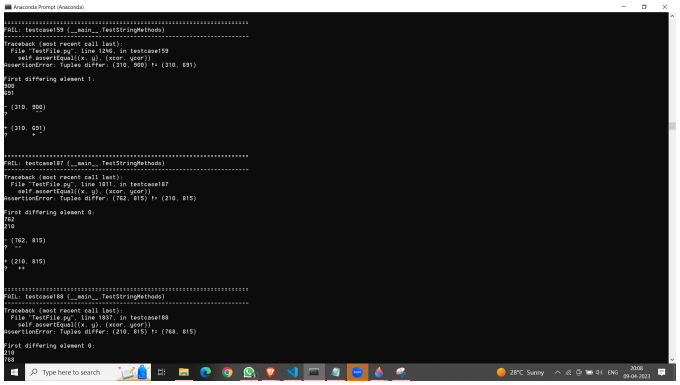


Figure 67: Results of Test Cases on the Version-2 GUI - 16

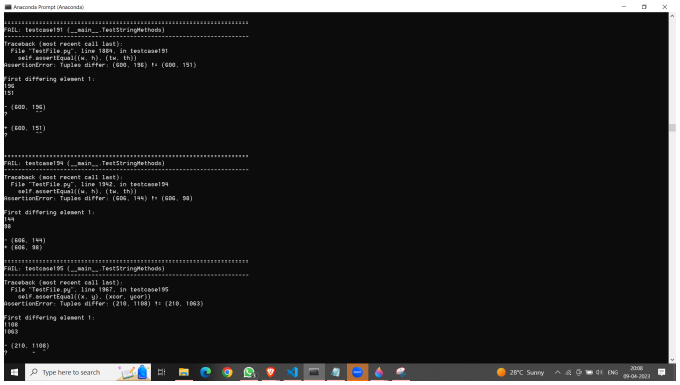


Figure 68: Results of Test Cases on the Version-2 GUI - 17

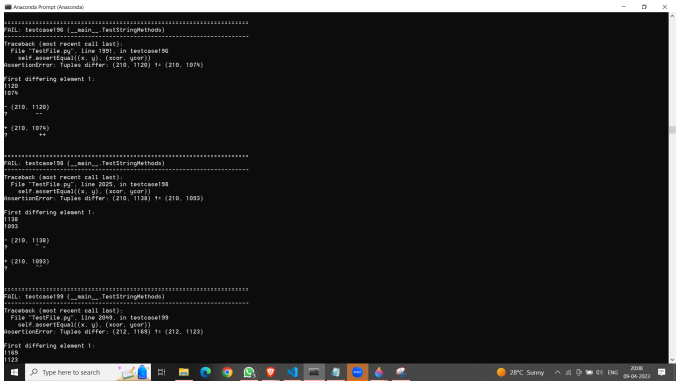


Figure 69: Results of Test Cases on the Version-2 GUI - 18

References

- Unit testing
- Unit Test Frameworks by Paul Hamill”
- What is Unit Testing and Why Developer Should Learn It
- Why Is Unit Testing Important in Software Development?
- Jest Framework
- Unit Tutorial — Testing Framework for Java
- What are the differences between Jest and JUnit?
- Heap Sort Algorithm
- Jest Crash Course - Unit Testing in JavaScript
- Jest Tutorial: Complete Guide to Jest Testing
- WHAT IS DESIGN OF EXPERIMENTS (DOE)?
- Pairwise Testing Or All-Pairs Testing Tutorial With Tools And Examples
- Pairwise Testing Guide: How To Perform Pairwise Testing
- EXTENDED COMBINATORIAL TESTING USING GRAPH ALGORITHMS AND APACHE SPARK
- Czerwonka (2006)
- PICT tool from Microsoft
- Pairwise Pict Online
- Pairwise Independent Combinatorial Testing
- Code coverage
- PMD Source Code Analyzer Project
- Collecting Unit Test Coverage in IntelliJ IDEA
- How To Use PMD Code Analyzer With IntelliJ
- Sample Code for the assignment
- 40 Best UI Testing Tools And Techniques
- 11 Best Automated UI Testing Tools In 2022
- 19 BEST Automation Testing Tools (2023)
- 9 Best GUI Testing Tools in Software Testing
- WebDriver
- A Complete Guide to Selenium UI testing
- Selenium Offical Downloads
- Chrome-Driver Offical Downloads Selenium Webdriver
- Selenium Selenium & Selenium WebDriver
- Selenium WebDriver Tutorial - A Comprehensive Guide to WebDriver Automation Selenium with Python
- The Selenium Browser Automation Project
- Prof. James Collofello, CSE 565 Lecture Videos, 2023
- Prof. Ayca Tuzmen, CSE 565 Lectures, 2023
- CSE 565 - Software Verification Validation and Testing - Assignment 1 - Unit testing & Unit testing Frameworks by Kedar Sai Nadh Reddy Kanchi

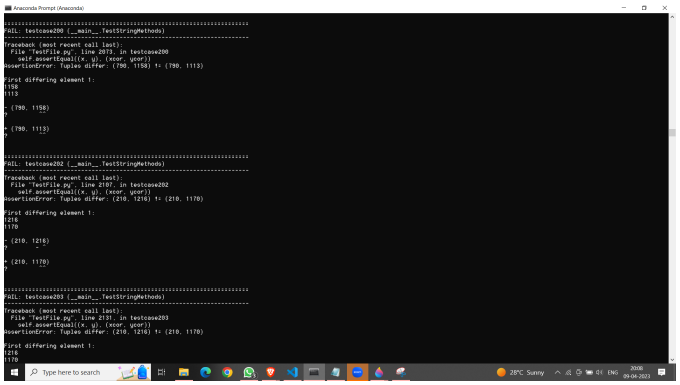


Figure 70: Results of Test Cases on the Version-2 GUI - 19

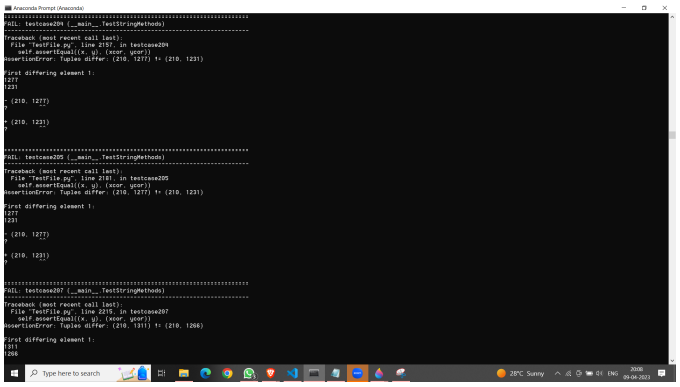


Figure 71: Results of Test Cases on the Version-2 GUI - 20

- CSE 565 - Software Verification Validation and Testing - Assignment 3 - Design of Experiments & Pairwise Testing by Kedar Sai Nadh Reddy Kanchi
- CSE 565 - Software Verification Validation and Testing - Assignment 4 - Decision Code Coverage by Kedar Sai Nadh Reddy Kanchi
- CSE 565 - Software Verification Validation and Testing - Assignment 5 - Graphical User Interface Testing by Kedar Sai Nadh Reddy Kanchi

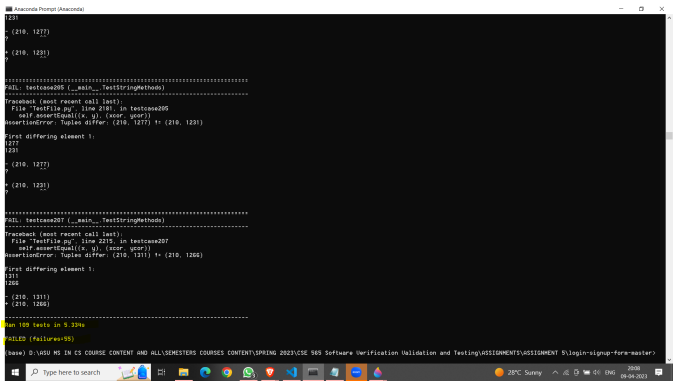


Figure 72: Results of Test Cases on the Version-2 GUI- 21

```javascript
// JavaScript program for implementation of Heap Sort

const heapsort_algorithm = {

    heapsort_code: function(arr) {
        var N = arr.length;

        // Build heap (rearrange array)
        for (var i = Math.floor(N / 2) - 1; i >= 0; i--)
            heapify(arr, N, i);

        // One by one extract an element from heap
        for (var i = N - 1; i > 0; i--) {
            // Move current root to end
            var temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
        return arr;
    }


}

// To heapify a subtree rooted with node i which is an index in arr[].
// n is size of heap
const heapify = (arr, N, i) => {

    var largest = i; // Initialize largest as root
    var l = 2 * i + 1; // left = 2*i + 1
    var r = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < N && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < N && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        var swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        // Recursively heapify the affected sub-tree
        heapify(arr, N, largest);
    }
}

module.exports = heapsort_algorithm;

```

```
const heapsort = require("./heapsort");

test('heapsort test case 1', () => {
    expect(heapsort.heapsort_code([12, 11, 13, 5, 6, 7])).toEqual([ 5, 6, 7, 11, 12, 13 ]);
});

test('heapsort test case 2', () => {
    expect(heapsort.heapsort_code([4,1,3,9,7])).toEqual([ 1,3,4,7,9 ]);
});

test('heapsort test case 3', () => {
    expect(heapsort.heapsort_code([10,9,8,7,6,5,4,3,2,1])).toEqual([ 1, 2,3,4,5,6,7,8,9,10 ]);
});

test('heapsort test case 4', () => {
    expect(heapsort.heapsort_code([])).toEqual([]);
});

test('heapsort test case 5', () => {
    expect(heapsort.heapsort_code(["banana", "apple", "mango", "pineapple", "orange"])).toEqual(["apple", "banana", "mango", "orange", "pineapple"]);
});

test('heapsort test case 6', () => {
    expect(heapsort.heapsort_code(["CAB", "ACB", "BCA", "ABC", "CBA", "BAC"])).toEqual(["ABC", "ACB", "BAC", "BCA", "CAB", "CBA"]);
});

test('heapsort test case 7', () => {
    expect(heapsort.heapsort_code([5,3,3,1,5,3,2])).toEqual([1,2,3,3,3,5,5]);
});

test('heapsort test case 8', () => {

 expect(heapsort.heapsort_code(['5','3','3','1','5','3','2'])).toEqual(['1','2','3','3','3','5','5']);
});

test('heapsort test case 9', () => {
    expect(heapsort.heapsort_code(["M4dkbH6NjJ", "1XSRJpa60H", "hxWwGbcMJM", "PjMgcuKxsO", "AWVQ1I3pZs", "6I4vODbmk8", "LjeMBnEqr1", "lnJnShEBE6", "RJzCi0UcRE", "IOvB93CeP9"])).
    toEqual(["1XSRJpa60H", "6I4vODbmk8", "AWVQ1I3pZs", "IOvB93CeP9", "LjeMBnEqr1", "M4dkbH6NjJ", "PjMgcuKxsO", "RJzCi0UcRE", "hxWwGbcMJM", "lnJnShEBE6"]);
});

test('heapsort test case 10 - checking to fail', () => {
    expect(heapsort.heapsort_code([4,1,3,9,7])).toEqual([ 1,7,4,3,9 ]);
});
```

Current scope: all classes | <empty package name>

Coverage Summary for Package: <empty package>

Package	Class, %	Method, %	Branch, %	Line, %
<empty package>	66.7% (2/3)	72.7% (8/11)	75% (15/20)	70% (35/50)

Class	Class, %	Method, %	Branch, %	Line, %
StaticAnalysis	0% (0/1)	0% (0/3)	0% (0/4)	0% (0/15)
VendingMachine	100% (1/1)	100% (2/2)	93.8% (15/16)	100% (24/24)
VendingMachineTest	100% (1/1)	100% (6/6)		100% (11/11)

generated on 2023-03-22 20:30

Static Source Code Analysis for the StaticAnalysis Code using the PMD tool - Assignment 4

PMD report

Problems found

#	File	Line	Problem
1	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	4	All classes, interfaces, enums and annotations must belong to a named package
2	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	4	Class comments are required
3	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	5	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
4	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	6	Parameter 'args' is not assigned and could be declared final
5	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	6	Public method and constructor comments are required
6	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	9	Local variable 'value' could be declared final
7	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	10	System.out.println is used
8	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	13	Public method and constructor comments are required
9	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	15	Avoid unused local variables such as 'weight'.
10	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING	15	Local variable 'weight' could be declared final

	2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java		
11	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	16	Avoid unused local variables such as 'length'.
12	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	16	Local variable 'length' could be declared final
13	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	19	Parameter 'length' is not assigned and could be declared final
14	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	19	Parameter 'product' is not assigned and could be declared final
15	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	19	Parameter 'weight' is not assigned and could be declared final
16	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	19	Public method and constructor comments are required
17	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	21	Found 'DD'-anomaly for variable 'cost' (lines '21'-'26').
18	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	21	Found 'DD'-anomaly for variable 'cost' (lines '21'-'30').
19	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	21	The initializer for variable 'cost' is never used (overwritten on lines 26 and 30)
20	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	22	Found 'DD'-anomaly for variable 'output' (lines '22'-'34').

21	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	22	Found 'DD'-anomaly for variable 'output' (lines '22'-'36').
22	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	22	The initializer for variable 'output' is never used (overwritten on lines 34 and 36)
23	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	24	Avoid using Literals in Conditional Statements
24	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	24	Use equals() to compare object references.
25	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	24	Use equals() to compare strings instead of '==' or '!='
26	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	33	Avoid using Literals in Conditional Statements
27	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	34	Avoid using if...else statements without curly braces
28	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	34	This statement should have braces
29	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	36	Avoid using if...else statements without curly braces
30	D:\ASU MS IN CS COURSE CONTENT AND ALL\SEMESTERS COURSES CONTENT\SPRING 2023\CSE 565 Software Verification Validation and Testing\ASSIGNMENTS\Assignment 4 - Decision Code Coverage\CSE 565 SVVT Assignment 4\src\StaticAnalysis.java	36	This statement should have braces

```python
import time
import unittest

from selenium import webdriver
from selenium.common import NoSuchElementException
from selenium.webdriver.common.by import By

# from pytime import time


browser = webdriver.Chrome()

# Please update this to the file location on your System
# browser.get(
#     "file:///D://ASU MS IN CS COURSE CONTENT AND ALL//SEMESTERS COURSES CONTENT//SPRING 2023//CSE 565 Software Verification Validation and Testing//ASSIGNMENTS//ASSIGNMENT 5//login-signup-form-master//firstpage.html"
# )

# This is for the updated file. Please update this to the file location on your System
browser.get("file:///D://ASU MS IN CS COURSE CONTENT AND ALL//SEMESTERS COURSES CONTENT//SPRING 2023//CSE 565 Software Verification Validation and Testing//ASSIGNMENTS//ASSIGNMENT 5//login-signup-form-master//secondpage.html")



class TestStringMethods(unittest.TestCase):
    maxDiff = None

    # test cases for the registration page

    def testcase100(self):
        try:
            l = browser.find_element(By.ID, "signup_homeimage")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 100
            th = 100
            xcor = 460
            ycor = 48
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 1 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 1 Failed")

    def testcase101(self):
        try:
            l = browser.find_element(By.ID, "signup_create_account_header")
```

```
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 36
            xcor = 310
            ycor = 169
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 2 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 2 Failed")

    def testcase102(self):
        try:
            l = browser.find_element(By.ID, "signup_userNameorEmail")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 81
            th = 18
            xcor = 310
            ycor = 237
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 3 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 3 Failed")

    def testcase103(self):
        try:
            l = browser.find_element(By.ID, "signup_userNameorEmail")
            s = l.text
            value = "User Name"
            self.assertEqual(s, value)
            print("Testcase 4 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 4 Failed")

    def testcase104(self):
        try:
            l = browser.find_element(By.ID, "signupUsername")
            s = l.text
```

```
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 44
            xcor = 310
            ycor = 261
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 5 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 5 Failed")

    def testcase105(self):
        try:
            l = browser.find_element(By.ID, "signup_user_Email_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 40
            th = 18
            xcor = 310
            ycor = 321
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 6 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 6 Failed")

    def testcase106(self):
        try:
            l = browser.find_element(By.ID, "signup_user_Email_label")
            s = l.text
            value = "Email"
            self.assertEqual(s, value)
            print("Testcase 7 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 7 Failed")

    def testcase107(self):
        try:
            l = browser.find_element(By.ID, "signupUsername_email")
            s = l.text
            location = l.location
```

```
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 44
            xcor = 310
            ycor = 344
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 8 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 8 Failed")

    def testcase108(self):
        try:
            l = browser.find_element(By.ID, "signup_user_password_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 70
            th = 18
            xcor = 310
            ycor = 404
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 9 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 9 Failed")

    def testcase109(self):
        try:
            l = browser.find_element(By.ID, "signup_user_password_label")
            s = l.text
            value = "Password"
            self.assertEqual(s, value)
            print("Testcase 10 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 10 Failed")

    def testcase110(self):
        try:
            l = browser.find_element(By.ID, "signup_password")
            s = l.text
            location = l.location
            size = l.size
```

```
            w, h = size["width"], size["height"]
            tw = 400
            th = 44
            xcor = 310
            ycor = 427
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 11 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 11 Failed")

    def testcase111(self):
        try:
            l = browser.find_element(By.ID, "signup_user_confirm_password_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 131
            th = 18
            xcor = 310
            ycor = 487
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 12 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 12 Failed")

    def testcase112(self):
        try:
            l = browser.find_element(By.ID, "signup_user_confirm_password_label")
            s = l.text
            value = "Confirm Password"
            self.assertEqual(s, value)
            print("Testcase 13 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 13 Failed")

    def testcase113(self):
        try:
            l = browser.find_element(By.ID, "signup_confirm_password")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
```

```
            tw = 400
            th = 44
            xcor = 310
            ycor = 510
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 14 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 14 Failed")

    def testcase114(self):
        try:
            l = browser.find_element(By.ID, "signup_user_phone_number_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 108
            th = 18
            xcor = 310
            ycor = 570
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 15 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 15 Failed")

    def testcase115(self):
        try:
            l = browser.find_element(By.ID, "signup_user_phone_number_label")
            s = l.text
            value = "Phone Number"
            self.assertEqual(s, value)
            print("Testcase 16 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 16 Failed")

    def testcase116(self):
        try:
            l = browser.find_element(By.ID, "signupUser_phonenumber")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
```

```
            th = 44
            xcor = 310
            ycor = 593
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 17 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 17 Failed")

    def testcase117(self):
        try:
            l = browser.find_element(By.ID, "signup_gender_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 53
            th = 18
            xcor = 310
            ycor = 653
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 18 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 18 Failed")

    def testcase118(self):
        try:
            l = browser.find_element(By.ID, "signup_gender_label")
            s = l.text
            value = "Gender"
            self.assertEqual(s, value)
            print("Testcase 19 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 19 Failed")

    def testcase119(self):
        try:
            l = browser.find_element(By.ID, "male_radio_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 18
```

```
            xcor = 310
            ycor = 684
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 20 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 20 Failed")

    def testcase120(self):
        try:
            l = browser.find_element(By.ID, "male_radio_label")
            s = l.text
            value = "Male"
            self.assertEqual(s, value)
            print("Testcase 21 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 21 Failed")

    def testcase121(self):
        try:
            l = browser.find_element(By.ID, "male_radio_label_button")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 13
            th = 13
            xcor = 384
            ycor = 687
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 22 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 22 Failed")

    def testcase122(self):
        try:
            l = browser.find_element(By.ID, "female_radio_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 18
            xcor = 310
```

```
            ycor = 714
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 23 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 23 Failed")

    def testcase123(self):
        try:
            l = browser.find_element(By.ID, "female_radio_label")
            s = l.text
            value = "Female"
            self.assertEqual(s, value)
            print("Testcase 24 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 24 Failed")

    def testcase124(self):
        try:
            l = browser.find_element(By.ID, "female_radio_label_button")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 13
            th = 13
            xcor = 403
            ycor = 717
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 25 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 25 Failed")

    def testcase125(self):
        try:
            l = browser.find_element(By.ID, "other_gender_radio_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 18
            xcor = 310
            ycor = 745
```

```
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 26 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 26 Failed")

    def testcase126(self):
        try:
            l = browser.find_element(By.ID, "other_gender_radio_label")
            s = l.text
            value = "Prefer not to say"
            self.assertEqual(s, value)
            print("Testcase 27 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 27 Failed")

    def testcase127(self):
        try:
            l = browser.find_element(By.ID, "other_gender_radio_label_button")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 13
            th = 13
            xcor = 467
            ycor = 748
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 28 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 28 Failed")

    def testcase128(self):
        try:
            l = browser.find_element(By.ID, "slider_range_for_age")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 62
            xcor = 310
            ycor = 775
            x = location.get("x")
```

```
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 29 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 29 Failed")

    def testcase129(self):
        try:
            l = browser.find_element(By.ID, "slider_label_for_age")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 28
            th = 18
            xcor = 310
            ycor = 797
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 30 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 30 Failed")

    def testcase130(self):
        try:
            l = browser.find_element(By.ID, "slider_label_for_age")
            s = l.text
            value = "Age"
            self.assertEqual(s, value)
            print("Testcase 31 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 31 Failed")

    def testcase131(self):
        try:
            l = browser.find_element(By.ID, "age_range_slider")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 338
            th = 2
            xcor = 340
            ycor = 805
            x = location.get("x")
            y = location.get("y")
```

```
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 32 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 32 Failed")

    def testcase132(self):
        try:
            l = browser.find_element(By.ID, "rangeValue")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 30
            th = 22
            xcor = 680
            ycor = 795
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 33 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 33 Failed")

    def testcase133(self):
        try:
            l = browser.find_element(By.ID, "rangeValue")
            s = l.text
            value = browser.find_element(By.ID, "age_range_slider").get_attribute(
                "value"
            )
            self.assertEqual(s, value)
            print("Testcase 34 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 34 Failed")

    def testcase134(self):
        try:
            l = browser.find_element(By.ID, "button_signup")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 45
            xcor = 310
            ycor = 852
            x = location.get("x")
```

```
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 35 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 35 Failed")

    def testcase135(self):
        try:
            l = browser.find_element(By.ID, "user_signup_button")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 45
            xcor = 310
            ycor = 852
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 36 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 36 Failed")

    def testcase136(self):
        try:
            l = browser.find_element(By.ID, "navigation_to_login_page")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 45
            xcor = 310
            ycor = 913
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 37 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 37 Failed")

```

```
    def testcase137(self):
        try:
            l = browser.find_element(By.ID,
"label_for_login_button_on_sign_up_page")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 18
            xcor = 310
            ycor = 913
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 38 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 38 Failed")

    def testcase138(self):
        try:
            l = browser.find_element(By.ID,
"label_for_login_button_on_sign_up_page")
            s = l.text
            value = "Already have an account?"
            self.assertEqual(s, value)
            print("Testcase 39 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 39 Failed")

    def testcase139(self):
        try:
            l = browser.find_element(By.ID, "login_button_on_sign_up_page")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 45
            xcor = 310
            ycor = 948
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 40 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 40 Failed")
```

```
833
834     def testcase140(self):
835         try:
836             username = browser.find_element(By.ID, "signupUsername")
837             username.clear()
838             username.send_keys("entering my username here as a value")
839
840             email = browser.find_element(By.ID, "signupUsername_email")
841             email.clear()
842             email.send_keys("entering my username here as a value")
843
844             password = browser.find_element(By.ID, "signup_password")
845             password.clear()
846             password.send_keys("entering my password here as a value")
847
848             cofirm_password = browser.find_element(By.ID,
    "signup_confirm_password")
849             cofirm_password.clear()
850             cofirm_password.send_keys("entering my password here as a value")
851
852             phone_number = browser.find_element(By.ID, "signupUser_phonenumber")
853             phone_number.clear()
854             phone_number.send_keys("8135630372")
855
856             browser.find_element(By.ID, "male_radio_label_button").click()
857             browser.find_element(By.ID, "user_signup_button").click()
858
859             i = 0
860             while i < 20:
861                 i = i + 1
862
863             l = browser.find_element(By.ID, "homeimage")
864             s = "Found"
865             self.assertEqual(s, "Found")
866             print("Testcase 41 Passed")
867
868         except:
869             print("Element does not exist Test Case 41 Failed")
870
871     # test cases for the login page
872
873     def testcase141(self):
874         try:
875             l = browser.find_element(By.ID, "homeimage")
876             s = l.text
877             location = l.location
878             size = l.size
879             w, h = size["width"], size["height"]
880             tw = 100
881             th = 100
882             xcor = 460
883             ycor = 48
884             x = location.get("x")
885             y = location.get("y")
886             # print("width - ", w)
887             # print("height - ", h)
888             # print("xcor - ", x)
889             # print("ycor - ", y)
890             self.assertEqual((w, h), (tw, th))
891             self.assertEqual((x, y), (xcor, ycor))
```

```
            print("Testcase 42 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 42 Failed")

    def testcase142(self):
        try:
            l = browser.find_element(By.ID, "login_page_title")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 36
            xcor = 310
            ycor = 169
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 43 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 43 Failed")

    def testcase143(self):
        try:
            l = browser.find_element(By.ID, "userNameorEmail")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            x = location.get("x")
            y = location.get("y")
            tw = 137
            th = 18
            xcor = 310
            ycor = 237
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 44 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 44 Failed")

    def testcase144(self):
        try:
            l = browser.find_element(By.ID, "userNameorEmail")
            s = l.text
            value = "Username or Email"
            self.assertEqual(s, value)
            print("Testcase 45 Passed")
```

```
952
953         except NoSuchElementException:
954             print("Element does not exist Test Case 45 Failed")
955
956     def testcase145(self):
957         try:
958             l = browser.find_element(By.ID, "userNameorEmail_input")
959             s = l.text
960             location = l.location
961             size = l.size
962             w, h = size["width"], size["height"]
963             tw = 400
964             th = 44
965             xcor = 310
966             ycor = 256
967             x = location.get("x")
968             y = location.get("y")
969             # print("width - ", w)
970             # print("height - ", h)
971             # print("xcor - ", x)
972             # print("ycor - ", y)
973             self.assertEqual((w, h), (tw, th))
974             self.assertEqual((x, y), (xcor, ycor))
975             print("Testcase 46 Passed")
976
977         except NoSuchElementException:
978             print("Element does not exist Test Case 46 Failed")
979
980     def testcase146(self):
981         try:
982             l = browser.find_element(By.ID, "user_password")
983             s = l.text
984             value = "Password"
985             self.assertEqual(s, value)
986             print("Testcase 47 Passed")
987
988         except NoSuchElementException:
989             print("Element does not exist Test Case 47 Failed")
990
991     def testcase147(self):
992         try:
993             l = browser.find_element(By.ID, "password_input")
994             s = l.text
995             location = l.location
996             size = l.size
997             w, h = size["width"], size["height"]
998             tw = 400
999             th = 44
1000            xcor = 310
1001            ycor = 334
1002            x = location.get("x")
1003            y = location.get("y")
1004            # print("width - ", w)
1005            # print("height - ", h)
1006            # print("xcor - ", x)
1007            # print("ycor - ", y)
1008            self.assertEqual((w, h), (tw, th))
1009            self.assertEqual((x, y), (xcor, ycor))
1010            print("Testcase 48 Passed")
1011
```

```
        except NoSuchElementException:
            print("Element does not exist Test Case 48 Failed")

    def testcase148(self):
        try:
            l = browser.find_element(By.ID,
"label_for_user_login_terms_and_conditions")
            s = l.text
            value = "Please read the Terms and conditions below:"
            self.assertEqual(s, value)
            print("Testcase 49 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 49 Failed")

    def testcase149(self):
        try:
            l = browser.find_element(By.ID,
"label_for_user_login_terms_and_conditions")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 18
            xcor = 310
            ycor = 394
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 50 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 50 Failed")

    def testcase150(self):
        try:
            l = browser.find_element(By.ID, "login_scroll_area")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 402
            th = 122
            xcor = 310
            ycor = 429
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 51 Passed")

```

```
        except NoSuchElementException:
            print("Element does not exist Test Case 51 Failed")

    def testcase151(self):
        try:
            l = browser.find_element(By.ID, "login_scroll_area")
            s = l.text
            value = "TERMS AND CONDITIONS BY SIGNING THIS AGREEMENT. (i) You acknowledge that you have read and understand the terms and conditions of this Agreement including those on page 2 of this Agreement; (ii) You agree that this Agreement is a net agreement that you cannot terminate or cancel, you have an unconditional obligation to make all payments due under this Agreement, and you cannot withhold, set off or reduce such payments for any reason; (iii) You will use the Products only for business purposes and in compliance with all applicable laws and regulations; (iv) you acknowledge that if this Agreement is replacing an existing agreement the new Installment Payment may include the balance of that existing agreement and result in a greater aggregate product cost to you; and (v) You agree that by providing a telephone number to a cellular or other wireless device, you are expressly consenting to receiving communications from us, our affiliates and agents (for non-marketing purposes) at that number, including, but not limited to, prerecorded and artificial voice messages, text messages, and calls from automated telephone dialing systems; these calls may incur fees from your cellular provider; and this consent applies to each such telephone number you provide to us now or in the future"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 52 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 52 Failed")

    def testcase152(self):
        try:
            l = browser.find_element(
                By.ID, "radio_label_for_user_login_terms_and_conditions"
            )
            s = l.text
            value = "I agree to the terms and conditions"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 53 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 53 Failed")

    def testcase153(self):
        try:
            l = browser.find_element(
                By.ID, "radio_label_for_user_login_terms_and_conditions"
            )
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 18
            xcor = 310
            ycor = 562
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
```

```
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 54 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 54 Failed")

    def testcase154(self):
        try:
            l = browser.find_element(By.ID, "login_terms_and_conditions")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 13
            th = 13
            xcor = 596
            ycor = 565
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 55 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 55 Failed")

    def testcase155(self):
        try:
            l = browser.find_element(By.ID, "button_login")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 45
            xcor = 310
            ycor = 596
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 56 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 56 Failed")

    def testcase156(self):
        try:
            l = browser.find_element(By.ID, "user_login_button")
            s = l.text
            location = l.location
```

```
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 45
            xcor = 310
            ycor = 596
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 57 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 57 Failed")

    def testcase157(self):
        try:
            l = browser.find_element(By.ID, "label_for_sign_up_on_login_page")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 400
            th = 18
            xcor = 310
            ycor = 657
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 58 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 58 Failed")


    def testcase158(self):
        try:
            l = browser.find_element(By.ID, "label_for_sign_up_on_login_page")
            s = l.text
            value = "Don't have an account?"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 59 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 59 Failed")

    def testcase159(self):
        try:
            l = browser.find_element(By.ID, "sign_up_button_on_login_page")
            s = l.text
            location = l.location
            size = l.size
```

```
            w, h = size["width"], size["height"]
            tw = 400
            th = 45
            xcor = 310
            ycor = 691
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 60 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 60 Failed")

    def testcase160(self):
        try:
            username = browser.find_element(By.ID, "userNameorEmail_input")
            username.clear()
            username.send_keys("entering my username here as a value")

            password = browser.find_element(By.ID, "password_input")
            password.clear()
            password.send_keys("entering my password here as a value")

            browser.find_element(By.ID, "login_terms_and_conditions").click()
            browser.find_element(By.ID, "user_login_button").click()

            l = browser.find_element(By.ID, "review_homeimage")
            s = "Found"
            self.assertEqual(s, "Found")
            print("Testcase 61 Passed")

        except:
            print("Element does not exist Test Case 61 Failed")

    def testcase161(self):
        try:
            l = browser.find_element(By.ID, "review_homeimage")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 301
            th = 168
            xcor = 359
            ycor = 48
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 62 Passed")

```

```
        except NoSuchElementException:
            print("Element does not exist Test Case 62 Failed")

    def testcase162(self):
        try:
            l = browser.find_element(By.ID, "review_page_title")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 36
            xcor = 210
            ycor = 237
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 63 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 63 Failed")

    def testcase163(self):
        try:
            l = browser.find_element(By.ID, "review_page_title")
            s = l.text
            value = "WE APPRECIATE YOUR REVIEW!"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 64 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 64 Failed")

    def testcase164(self):
        try:
            l = browser.find_element(By.ID, "review_title_subtet")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 16
            xcor = 210
            ycor = 305
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 65 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 65 Failed")
```

```
1354
1355    def testcase165(self):
1356        try:
1357            l = browser.find_element(By.ID, "review_title_subtet")
1358            s = l.text
1359            value = "Your review will help me to improve in my future assignments."
1360            self.assertEqual(s.strip(), value.strip())
1361            print("Testcase 66 Passed")
1362        except NoSuchElementException:
1363            print("Element does not exist Test Case 66 Failed")
1364
1365    def testcase166(self):
1366        try:
1367            l = browser.find_element(By.ID, "review_username_label")
1368            s = l.text
1369            location = l.location
1370            size = l.size
1371            w, h = size["width"], size["height"]
1372            tw = 74
1373            th = 18
1374            xcor = 210
1375            ycor = 344
1376            x = location.get("x")
1377            y = location.get("y")
1378            # print("width - ", w)
1379            # print("height - ", h)
1380            # print("xcor - ", x)
1381            # print("ycor - ", y)
1382            self.assertEqual((w, h), (tw, th))
1383            self.assertEqual((x, y), (xcor, ycor))
1384            print("Testcase 67 Passed")
1385
1386        except NoSuchElementException:
1387            print("Element does not exist Test Case 67 Failed")
1388
1389    def testcase167(self):
1390        try:
1391            l = browser.find_element(By.ID, "review_username_label")
1392            s = l.text
1393            value = "Username"
1394            self.assertEqual(s.strip(), value.strip())
1395            print("Testcase 68 Passed")
1396        except NoSuchElementException:
1397            print("Element does not exist Test Case 68 Failed")
1398
1399    def testcase168(self):
1400        try:
1401            l = browser.find_element(By.ID, "review_username_input")
1402            s = l.text
1403            location = l.location
1404            size = l.size
1405            w, h = size["width"], size["height"]
1406            tw = 600
1407            th = 44
1408            xcor = 210
1409            ycor = 367
1410            x = location.get("x")
1411            y = location.get("y")
1412            # print("width - ", w)
1413            # print("height - ", h)
```

```
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 69 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 69 Failed")

    def testcase169(self):
        try:
            l = browser.find_element(By.ID, "review_user_email")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 40
            th = 18
            xcor = 210
            ycor = 427
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 70 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 70 Failed")

    def testcase170(self):
        try:
            l = browser.find_element(By.ID, "review_user_email")
            s = l.text
            value = "Email"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 71 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 71 Failed")

    def testcase171(self):
        try:
            l = browser.find_element(By.ID, "review_user_email_input")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 44
            xcor = 210
            ycor = 450
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
```

```
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 72 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 72 Failed")

    def testcase172(self):
        try:
            l = browser.find_element(By.ID, "rate_your_service")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 18
            xcor = 210
            ycor = 515
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 73 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 73 Failed")

    def testcase173(self):
        try:
            l = browser.find_element(By.ID, "rate_your_service")
            s = l.text
            value = "Please rate your service experience for the following
parameters"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 74 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 74 Failed")

    def testcase174(self):
        try:
            l = browser.find_element(By.ID, "review_selection_tool_rating_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 358
            th = 18
            xcor = 210
            ycor = 555
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
```

```
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 75 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 75 Failed")

    def testcase175(self):
        try:
            l = browser.find_element(By.ID, "review_selection_tool_rating_label")
            s = l.text
            value = "1. Selection of the testing tool for this assignment?"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 76 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 76 Failed")

    def testcase176(self):
        try:
            l = browser.find_element(
                By.ID, "review_page_overall_experience_star_rating"
            )
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 175
            th = 35
            xcor = 210
            ycor = 578
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 77 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 77 Failed")

    def testcase177(self):
        try:
            l = browser.find_element(By.ID, "review_horizonatal_line_one")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 2
            xcor = 210
            ycor = 627
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
```

```
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 78 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 78 Failed")

    def testcase178(self):
        try:
            l = browser.find_element(By.ID, "review_assesment_2nd_question")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 564
            th = 54
            xcor = 210
            ycor = 639
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 79 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 79 Failed")

    def testcase179(self):
        try:
            l = browser.find_element(By.ID, "review_assesment_2nd_question")
            s = l.text
            value = "2. Assessment of the submitted report in terms of set of grammar or formatting errors, information about the selected testing tool and the description of the Test Cases executed as part of this assignment?"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 80 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 80 Failed")

    def testcase180(self):
        try:
            l = browser.find_element(
                By.ID, "review_page_assesment_2nd_question_star_rating"
            )
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 175
            th = 35
            xcor = 210
            ycor = 699
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
```

```
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 81 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 81 Failed")

    def testcase181(self):
        try:
            l = browser.find_element(By.ID, "review_horizonatal_line_two")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 2
            xcor = 210
            ycor = 748
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 82 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 82 Failed")

    def testcase182(self):
        try:
            l = browser.find_element(By.ID,
"review_implementation_3rd_question_label")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 573
            th = 36
            xcor = 210
            ycor = 760
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 83 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 83 Failed")

    def testcase183(self):
        try:
```

```
            l = browser.find_element(By.ID,
"review_implementation_3rd_question_label")
            s = l.text
            value = "3. How did you like the implementation of the application and
the UI submitted as part of this assignment?"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 84 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 84 Failed")

    def testcase184(self):
        try:
            l = browser.find_element(
                By.ID, "review_implementation_3rd_question_scale_rating"
            )
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 35
            xcor = 210
            ycor = 838
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 85 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 85 Failed")

    def testcase185(self):
        try:
            l = browser.find_element(By.ID, "review_horizonatal_line_three")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 2
            xcor = 210
            ycor = 902
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 86 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 86 Failed")

```

```
    def testcase186(self):
        try:
            l = browser.find_element(By.ID,
"3rd_question_review_scale_rating_labels")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 0
            xcor = 210
            ycor = 815
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 87 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 87 Failed")

    def testcase187(self):
        try:
            l = browser.find_element(
                By.ID, "3rd_question_review_scale_rating_labels_for_poor"
            )
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 47
            th = 18
            xcor = 210
            ycor = 815
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 88 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 88 Failed")

    def testcase188(self):
        try:
            l = browser.find_element(
                By.ID, "3rd_question_review_scale_rating_labels_for_best"
            )
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
```

```
            tw = 42
            th = 18
            xcor = 768
            ycor = 815
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 89 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 89 Failed")

    def testcase189(self):
        try:
            l = browser.find_element(
                By.ID, "3rd_question_review_scale_rating_labels_for_poor"
            )
            s = l.text
            value = "POOR"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 90 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 90 Failed")

    def testcase190(self):
        try:
            l = browser.find_element(
                By.ID, "3rd_question_review_scale_rating_labels_for_best"
            )
            s = l.text
            value = "BEST"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 91 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 91 Failed")

    def testcase191(self):
        try:
            l = browser.find_element(By.ID, "custom_review_text_area_box")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 151
            xcor = 210
            ycor = 914
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
```

```
            print("Testcase 92 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 92 Failed")

    def testcase192(self):
        try:
            l = browser.find_element(By.ID, "custom_review_text_area_box_labek")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 571
            th = 18
            xcor = 210
            ycor = 914
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 93 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 93 Failed")

    def testcase193(self):
        try:
            l = browser.find_element(By.ID, "custom_review_text_area_box_labek")
            s = l.text
            value = "4. If there are any Other suggestions, please let me know in
the text area below?"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 94 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 94 Failed")

    def testcase194(self):
        try:
            l = browser.find_element(By.ID, "custom_review_text_area_box_ui_html")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 606
            th = 98
            xcor = 210
            ycor = 950
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 95 Passed")
```

```
1945
1946        except NoSuchElementException:
1947            print("Element does not exist Test Case 95 Failed")
1948
1949    def testcase195(self):
1950        try:
1951            l = browser.find_element(By.ID, "review_horizonatal_line_four")
1952            s = l.text
1953            location = l.location
1954            size = l.size
1955            w, h = size["width"], size["height"]
1956            tw = 600
1957            th = 2
1958            xcor = 210
1959            ycor = 1063
1960            x = location.get("x")
1961            y = location.get("y")
1962            # print("width - ", w)
1963            # print("height - ", h)
1964            # print("xcor - ", x)
1965            # print("ycor - ", y)
1966            self.assertEqual((w, h), (tw, th))
1967            self.assertEqual((x, y), (xcor, ycor))
1968            print("Testcase 96 Passed")
1969
1970        except NoSuchElementException:
1971            print("Element does not exist Test Case 96 Failed")
1972
1973    def testcase196(self):
1974        try:
1975            l = browser.find_element(By.ID, "review_page_overall_experience")
1976            s = l.text
1977            location = l.location
1978            size = l.size
1979            w, h = size["width"], size["height"]
1980            tw = 600
1981            th = 18
1982            xcor = 210
1983            ycor = 1074
1984            x = location.get("x")
1985            y = location.get("y")
1986            # print("width - ", w)
1987            # print("height - ", h)
1988            # print("xcor - ", x)
1989            # print("ycor - ", y)
1990            self.assertEqual((w, h), (tw, th))
1991            self.assertEqual((x, y), (xcor, ycor))
1992            print("Testcase 97 Passed")
1993
1994        except NoSuchElementException:
1995            print("Element does not exist Test Case 97 Failed")
1996
1997    def testcase197(self):
1998        try:
1999            l = browser.find_element(By.ID, "review_page_overall_experience")
2000            s = l.text
2001            value = "5. Your overall experience with my submitted assignment?"
2002            self.assertEqual(s.strip(), value.strip())
2003            print("Testcase 98 Passed")
2004        except NoSuchElementException:
```

```
            print("Element does not exist Test Case 98 Failed")

    def testcase198(self):
        try:
            l = browser.find_element(By.ID, "slider_range_for_overall_rating")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 62
            xcor = 210
            ycor = 1093
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 99 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 99 Failed")

    def testcase199(self):
        try:
            l = browser.find_element(By.ID, "review_overall_rating_slider")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 576
            th = 2
            xcor = 212
            ycor = 1123
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 100 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 100 Failed")

    def testcase200(self):
        try:
            l = browser.find_element(By.ID,
"review_overall_rating_slider_rangeValue")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 20
            th = 22
```

```
            xcor = 790
            ycor = 1113
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 101 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 101 Failed")

    def testcase201(self):
        try:
            l = browser.find_element(By.ID,
"review_overall_rating_slider_rangeValue")
            s = l.text
            value = "5"
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 102 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 102 Failed")

    def testcase202(self):
        try:
            l = browser.find_element(By.ID, "button_submit_review")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 45
            xcor = 210
            ycor = 1170
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 103 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 103 Failed")

    def testcase203(self):
        try:
            l = browser.find_element(By.ID, "user_submit_review_button")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 45
            xcor = 210
```

```
            ycor = 1170
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 104 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 104 Failed")

    def testcase204(self):
        try:
            l = browser.find_element(
                By.ID, "navigation_to_login_pagefrom_the_review_page"
            )
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 45
            xcor = 210
            ycor = 1231
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 105 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 105 Failed")

    def testcase205(self):
        try:
            l = browser.find_element(By.ID,
"label_for_login_button_on_review_page")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 18
            xcor = 210
            ycor = 1231
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
```

```
            print("Testcase 106 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 106 Failed")

    def testcase206(self):
        try:
            l = browser.find_element(By.ID,
"label_for_login_button_on_review_page")
            s = l.text
            value = "Not yet ready to review ? No problem, come back later to
submit your review."
            self.assertEqual(s.strip(), value.strip())
            print("Testcase 107 Passed")
        except NoSuchElementException:
            print("Element does not exist Test Case 107 Failed")

    def testcase207(self):
        try:
            l = browser.find_element(By.ID, "logout_button_on_review_page")
            s = l.text
            location = l.location
            size = l.size
            w, h = size["width"], size["height"]
            tw = 600
            th = 45
            xcor = 210
            ycor = 1266
            x = location.get("x")
            y = location.get("y")
            # print("width - ", w)
            # print("height - ", h)
            # print("xcor - ", x)
            # print("ycor - ", y)
            self.assertEqual((w, h), (tw, th))
            self.assertEqual((x, y), (xcor, ycor))
            print("Testcase 108 Passed")

        except NoSuchElementException:
            print("Element does not exist Test Case 108 Failed")

    def testcase208(self):
        try:
            username = browser.find_element(By.ID, "review_username_input")
            username.clear()
            username.send_keys("entering my username here as a value")

            email = browser.find_element(By.ID, "review_user_email_input")
            email.clear()
            email.send_keys("entering my email here as a value")

            textarea = browser.find_element(
                By.ID, "custom_review_text_area_box_ui_html"
            )
            textarea.clear()
            textarea.send_keys("entering my suggestions here as a value")

            browser.find_element(By.ID, "user_submit_review_button").click()

            l = browser.find_element(By.ID, "homeimage")
```

```
2240             s = "Found"
2241             self.assertEqual(s, "Found")
2242             print("Testcase 109 Passed")
2243 
2244         except:
2245             print("Element does not exist Test Case 109 Failed")
2246 
2247 
2248 
2249 if __name__ == "__main__":
2250     unittest.main()
2251 
```