CPS3230 – Assignment 2 Chakotay Incorvaia 358199(M)

Fundamentals of Software Testing

Semester 1

2019/2020

B.Sc.IT(Hons) Software Development



Contents

[Plagiarism Form: 3](#_Toc27817759)

[Assignment Github Page: 4](#_Toc27817760)

[Task 1: 4](#_Toc27817761)

[Step definitions file: 6](#_Toc27817762)

[Design Decisions: 6](#_Toc27817763)

[Task 2: 8](#_Toc27817764)

[FSM Model: 8](#_Toc27817765)

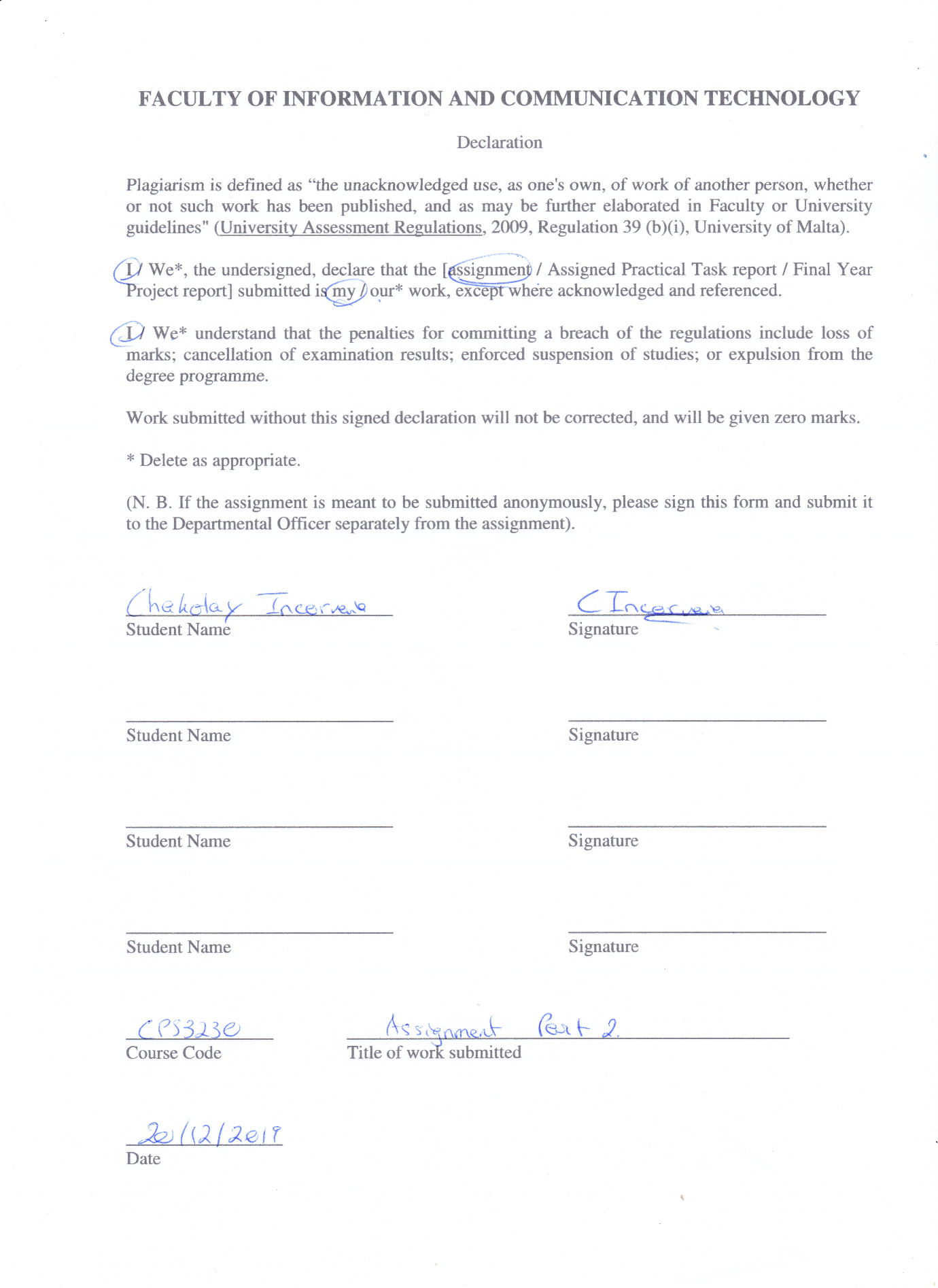
[Test Results: 10](#_Toc27817766)

[GreedyTester 10](#_Toc27817767)

[LookaheadTester 10](#_Toc27817768)

[RandomTester 10](#_Toc27817769)

# Plagiarism Form:



# Assignment Github Page:

The examiner should be already added: <https://github.com/CHAKOTAY99/CPS3230_Assignment_2>

# Task 1:

For my assignment I decided to use the website of SimarkSupplies, a supplier and reseller of computer components and gaming related accessories. Their website is very reliable as it is constantly up to date and does not see any major stylistic changes which might affect the development of the tests throughout the assignment lifetime. Seasonal changes such as those for Black Friday and Christmas sales only change a minor part of the website, with fields being renamed but keep the same identifiers and class names. In preparation for such scenarios, the tests have been written in a way in which stylistic changes are disregarded.

From a technical perspective, ‘www.simarksupplies.com’ is much easier to test in comparison to other websites. This is because it follows good development principles, with many of the fields containing unique id’s, class names and other identifiers which make them easy to distinguish from other properties. In rare cases in which this was not the case, *XPath* had to be used in order to identify the unidentifiable properties.

The website also meets all the criteria in order to qualify for use in the assignment:

* Logging in
* Logging out
* Searching for products
* Add products into shopping cart
* Remove Products from shopping cart
* Checkout Products from shopping cart
* Modify quantities of products on order or shopping cart

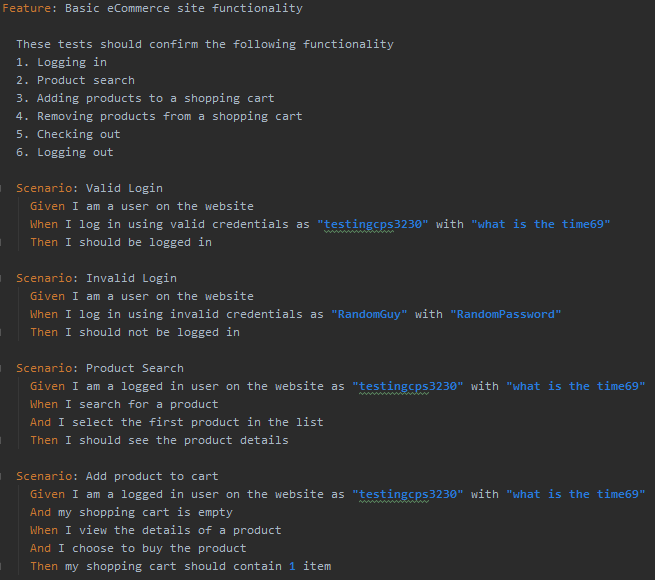
The images showcase the contents of the feature file.

Figure - First Four Scenarios

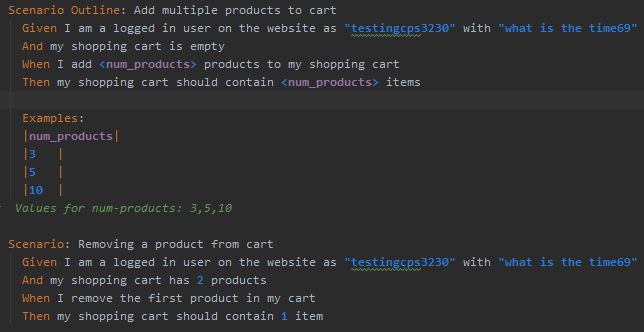


Figure - Final Four Scenarios

## Step definitions file:

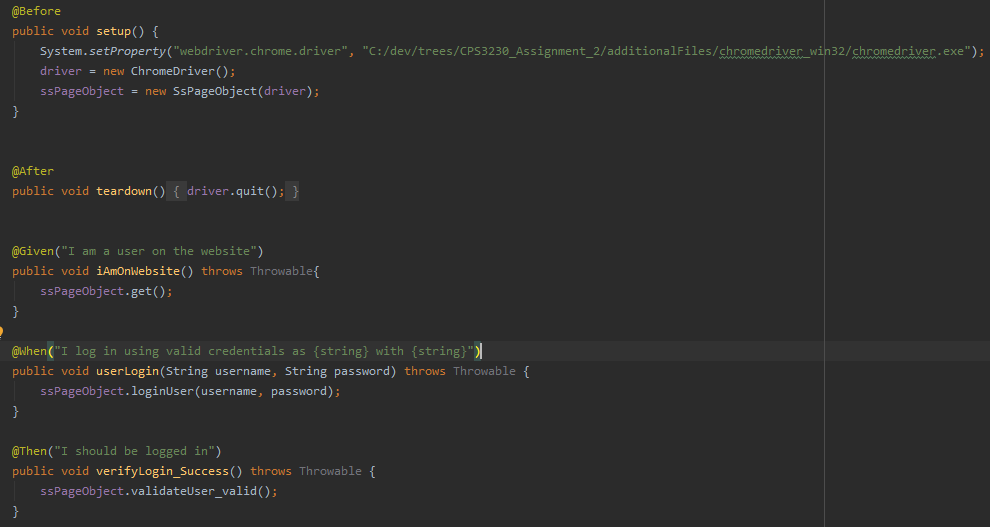


Figure - Sample of step definitions

The step definitions file contains which methods are run step by step for the scenarios. For example, logging in the website, validation after the login and the teardown which is called at the end of the run. Each method also has its respective annotations such as @Given, @When and @Then, for the scenario that they satisfy.

## Design Decisions:

The scenarios with edited in order to include parameters which are run by the methods such as the login credentials. They were also edited to include the amounts of products that is to be run in **Scenario 5**.

The chosen website had a minor issue with the login/registration and logout functions. Their placement on the webpage were in the same location, only changing when their respective action was run in order to display the alternative. The link was also the first element located on the webpage, which made finding it easier. Thus, it was decided that when the login or logout function was required, a check had first to be made in order to check the status of the user, so that the link can be identified, before the action is run.

In scenarios in which the resulting list from a search query had to be tested, the test query was given a generic name so that it would return more than one product. This is so that the manipulation of the results list could be possible. In these cases, the first result was taken.

The chosen website had two implementations of the shopping cart. One was a dedicated shopping cart page which must be used in order to proceed to checkout. This page had no apparent way in which a product can be removed from the list, instead its value must be changed to zero and have the list updated via an “Update Cart” button. Alternatively, one could also use the shopping cart found on the right side of the screen of the website. This version of the shopping cart enabled the user to quickly remove items via a dedicated button, but it did not allow them on the other hand to change the quantity of the product. In the case of Task 1, the latter option was used because it reflects the most common user behaviour, whilst in Task 2 both methods were found useful and were more detailed.

For more accurate tests, the cart is emptied at the end of each test and the user is also made sure to be logged out.

Scenario 5 was interpreted to include separate products of sizes 3, 5 and 10. For this an array was created which contains the names of 10 different products which can be search for and added into the cart.

# Task 2:

## FSM Model:

A screen shot of a smart phone

Description automatically generated

Figure 4 - FSM model

The model was made to represent typical user behaviour and is made to be quite detailed. The user can perform certain actions from any state for most of the model. I believe this is the best representation of typical user behaviour. For example, the user can search from any state, regardless if the user is logged in or logged out. The same exists for accessing the shopping cart and many other actions such as logging in from any point in the application – representing users which fill up their shopping cart before logging in. There are certain limitations in the tests such as only allowing the test to checkout if the user is both logged in and in the shopping cart or allowing a user to add a product to the cart only from the resulting search list. In order for the test to come to a conclusion, the user cannot continue after reaching the checkout state and the test is forced to end.

The amount of states in the FSM model was changed into a list of enums. Each enum will represent a state such as being logged in, logged out, being in the product details, the resulting list from a search query, being in the shopping cart and finally checking out.

Two classes were created, the ‘WebsiteModel’ and ‘WebsiteModelTest’. ‘WebsiteModelTest’ is tasked with running the actual test with the setup and teardown which must be applied. Whilst ‘WebsiteModel’ contains all the transitions, guards and action methods. These methods are in control of the logic in the test.

A class was created which contains the actual methods which are to be called during the tests, called ‘SsSystem’. They are used in the transitions to achieve objectives such as logging in, logging out, purchasing a product and many more.

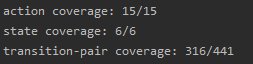
The website homepage did not contain and clickable products, and so only the other headings were used. In cases of seasonal sales, the website “Weekly Offers” section gets renamed to the sale name such as “Black Friday Sale” or “Christmas Offers”. To counter this change, the headings are searched for by index and not by name, so that these changes do not affect the tests. The same principle applies to checking the product categories, once clicked, the website will return a results list. This is because each search, category or headings list in the website, all return a resulting products list. Some results lists had to be differentiated for validation because of different heading names, namely “New Products” and “Weekly Offers”, which do not use the same standard principles like the other headings.

By default, the user is assumed to be logged out, which acts as a starting point for the test. In order to access some other states without having to be in the initial logged in and logout out states, methods had to be made. These methods ensure that the user is either logged in or logged out without being in the initial states, allowing the test to carry on from other states such as the shopping cart or the search result list.

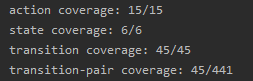
Lastly, there are three different types of tests to run, these are greedy test, lookahead test and random test. Each test runs differently due to different goals and goes through each transition and state at their own pace and manner and have been tested for this assignment. The greedy test is useful to give priority to any unexplored path, meaning that the test decides which at the state it is to transition to another state. The lookahead test does a lookahead of several transitions, by default it does 3 transitions, to unexplored paths whereas the random test implements a random walk through the model. At the end the greedy test was decided to be the final test to use in such a model, this is since it results in the highest pair-transition coverage and explores each transition given by the user.

## Test Results:

### GreedyTester



### LookaheadTester



### RandomTester

