



UNIVERSITY OF MALTA  
L-Università ta' Malta

# Fundamentals of Software Testing Assignment

Elise Callus – 265896(M) | Jonathan Borg – 63396(M) | Karl Farrugia – 59796(M)  
CPS3230 – Fundamentals of Software Testing  
B.Sc(Hons) Computing Science

## FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

### Declaration

Plagiarism is defined as “the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines” (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I/ We\*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is ~~my~~/ our\* work, except where acknowledged and referenced.

I/ We\* understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

\* Delete as appropriate.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Jonathan Borg 63396(M)  
Student Name

JBorg  
Signature

Elise Callus 265896(M)  
Student Name

ECallus  
Signature

Karl Farrugia 59796(M)  
Student Name

Karl Farrugia  
Signature

\_\_\_\_\_  
Student Name

\_\_\_\_\_  
Signature

CPS3230  
Course Code

Fundamentals of Software Testing Assignment  
Title of work submitted

07/01/2017  
Date

## TABLE OF CONTENTS

Introduction .....	3
Task 1. Unit testing and Test Driven Development .....	3
Implementation .....	4
Coverage through Unit-Tests .....	4
Test Doubles.....	5
Task 2. Cucumber and Automated Web Testing .....	6
Setup .....	6
Developing Web Applications .....	6
Login – index.jsp.....	6
Account Page – display.jsp.....	6
Error Page – error.jsp.....	7
Withdraw Page – withdraw.jsp.....	7
Cucumber Testing and Automated Web Testing.....	9
AffiliateLogin.feature .....	9
RunCucumberTests.java.....	9
PageTest.java .....	9
First Scenario.....	9
Second Scenario .....	10
Third Scenario .....	11
Fourth Scenario .....	11
Task 3. Model Based Testing.....	12
State Model.....	12
Assumptions:.....	13
Graph .....	13
Further Steps.....	14
Task 4. Performance Testing.....	15
Conclusion.....	17

## INTRODUCTION

The assignment was tackled on one computer with one person coding and two other persons overwriting the code and searching for solutions when getting stuck, and at a later point switching roles. A git repository was created; this repository was used to store the current state of the program. Apart from this, the repository was used to revert changes and keep track of all the progress done on the project. A link to this repository can be found at <https://github.com/jonathanborg/CPS3230-Software-Testing>.

## TASK 1. UNIT TESTING AND TEST DRIVEN DEVELOPMENT

The main objective of this task was to build and unit-test the class diagram provided. This diagram was modified and altered to provide better functionality of the classes and variables, while ensuring maximum test coverage.

Figure 1 shows the new modified class diagram describing the system implemented in our project.

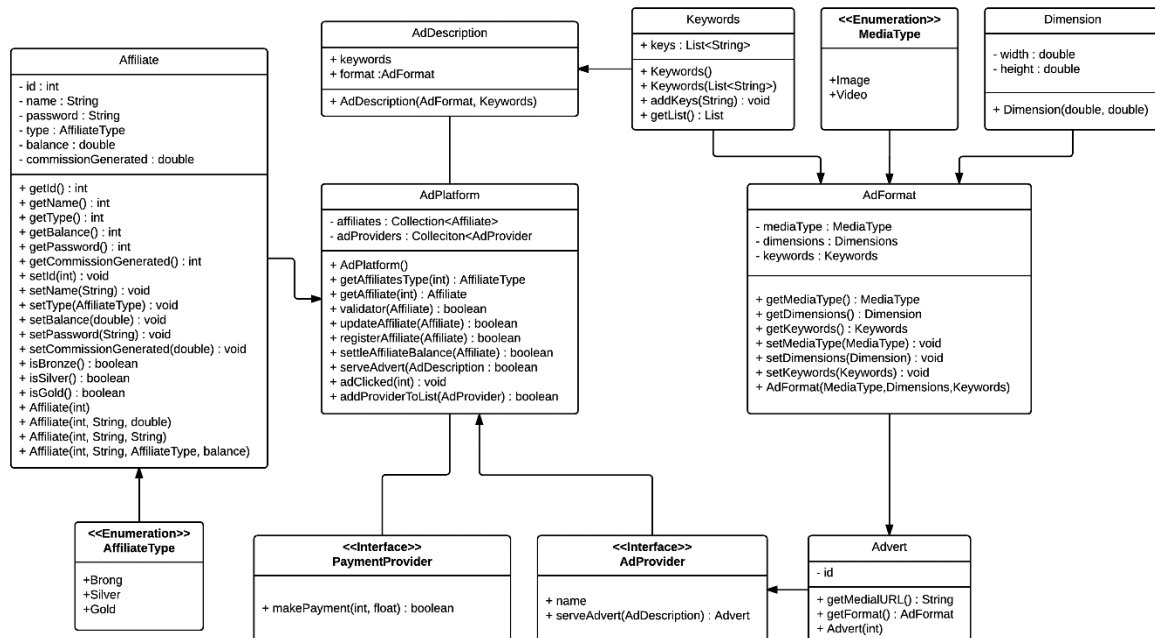


Figure 1 The Modified class diagram of the system implemented

## Implementation

This class diagram varies from the original class diagram in the following ways.

- The constructors, setters and getters were added in Affiliate and AdFormat classes.
- Dimension class was created, which was used within AdFormat. This class was used to indicate the height and width of an advert.
- Keywords class was created to store a list of keywords, which would be used to identify the description and format of an advert.
- Significant number of changes to the AdPlatform class.

The AdPlatform class was modified from the original design as to make implementation of method and classes easier, alongside the testing itself.

- AdPlatform constructor.
- getAffiliatesType, which finds the type of an affiliate given by its ID.
- getAffiliate, which retrieves an affiliate from the collection.
- validator, used for task two, which checks if an affiliate matches a pre-existent affiliate according to its ID and password.
- updateAffiliate, which is used to update the information of an affiliate, such as its balance or its type.
- registerAffiliate, used to add a new affiliate to the collection.
- settleAffiliateBalance, used to settle an affiliate's balance only if the affiliate has more than €4.99 and gets the commission upon settling his balance. The commission is set according to the affiliate type. Apart from that, an affiliate does not lose his type upon settling his balance. This means if an affiliate reached gold, he will not go down to bronze upon withdrawal but will keep his title of gold.
- serveAdvert, used to check if an advert with the same description exists.
- adClicked, increments the affiliates' balance by €0.50 and updates the affiliates' type if required.
- addProviderToList, simply adds a new adProvider to the collection.

## Coverage through Unit-Tests

All the unit tests for the system were implemented within the AdPlatformTest class. This class stores a total of 30 tests, with these tests 100% coverage was obtained within the classes and methods. On the other hand, 96% line coverage was obtained, the only two classes which lacked 100% line coverage were the AdPlatform class and the Affiliate class.

AdPlatform obtained a total of 94.4% line coverage, the only lines not covered in the unit tests were whenever a for loop had an if statement inside of it and the loop was just being used to retrieve something from one of the collections.

The affiliate class obtained a total of 98% line coverage, the only line which was not covered lies within the override method. The equals method was overrode so that the affiliates could be compared against each other through the use of this function, which checked an affiliate by its ID.

Using these unit tests, maximum statement and branch coverage were obtained. The values indicated above were obtained using IntelliJ IDEs' built in code, method and line coverage. No test patterns were used to implement the system.

### Test Doubles

To obtain maximum branch coverage, test doubles were used. Most of the test doubles used vary by one line only, example registering an affiliate one or registering an affiliate twice, as shown below.

All the test doubles are indicated within the comments of each test inside AdPlatformTest class

```
@Test//adding an element for the 1st time
public void registerAffiliateTrue() throws Exception {
    Affiliate aff = new Affiliate(1,"Client", AffiliateType.GOLD,5);
    assertEquals(true,adPlatform.registerAffiliate(aff));
}

@Test//adding an element for the second time to a collection
public void registerAffiliateFalse() throws Exception {
    Affiliate aff = new Affiliate(1,"Client", AffiliateType.GOLD,5);
    adPlatform.registerAffiliate(aff);
    //adding same element twice
    assertEquals(false,adPlatform.registerAffiliate(aff));
}
```

[ all classes ] [ cps.uom.edu ]

#### Coverage Summary for Package: cps.uom.edu

Package	Class, %	Method, %	Line, %
cps.uom.edu	100% (9/ 9)	100% (50/ 50)	96.9% (155/ 160)

Class ▲	Class, %	Method, %	Line, %
AdDescription	100% (1/ 1)	100% (1/ 1)	100% (4/ 4)
AdFormat	100% (1/ 1)	100% (7/ 7)	100% (14/ 14)
AdPlatform	100% (1/ 1)	100% (10/ 10)	94.4% (67/ 71)
Advert	100% (1/ 1)	100% (3/ 3)	100% (5/ 5)
Affiliate	100% (1/ 1)	100% (20/ 20)	98% (48/ 49)
AffiliateType	100% (1/ 1)	100% (2/ 2)	100% (2/ 2)
Dimension	100% (1/ 1)	100% (1/ 1)	100% (4/ 4)
Keywords	100% (1/ 1)	100% (4/ 4)	100% (9/ 9)
MediaType	100% (1/ 1)	100% (2/ 2)	100% (2/ 2)

generated on 2017-01-05 17:23

Figure 2 IntelliJ autogenerated file displaying coverage summary

## TASK 2. CUCUMBER AND AUTOMATED WEB TESTING

### Setup

The tools that were used to accomplish this task were; the Apache Tomcat, which was used to launch the JSP coded website as a localhost, and the Selenium WebDriver, which was used to access the launched localhost website, and automatically run certain actions and compare the results obtained from those actions to expected results.

### Developing Web Applications

The task required the creation of a login page for affiliates, **index.jsp**, and account page for affiliates, **display.jsp**. To cater for all the requirements another two pages were created, the error page, **error.jsp**, and the withdraw page, **withdraw.jsp**.

#### Login – index.jsp

On launching the login page two prompt boxes are displayed. These request the affiliate's id and password. A submit button must be clicked after filling the particulars. On submission the login page forwards the collected data to **display.jsp** through the use of forms, where the relevant checks are made to verify the affiliate id and password are correct.

#### Account Page – display.jsp

The account page receives the information from the login page by requesting the value of a specified variable using `request.getParameter("VARIABLE_NAME")`. A request is made for the id and the password. On gathering this information the id and password are registered as an affiliate which is then compared to the registered affiliates within `AdPlatform()`. If the entered affiliate details match the details of a registered affiliate, the affiliate's details (id, name and balance are displayed), and the affiliate is saved globally to ensure that future redirections to the page will not log out the user, a redirection may occur on pressing an Ad or trying to withdraw funds. The id, name and balance are tagged using `class="VARIABLE_NAME"` which makes it possible to test for further on.

The Ad Click and Withdraw are implemented as buttons which redirect to the display page itself, and trigger their relevant if statement in which their code will be executed. The Ad button will result in calling the `adClicked` method, which credits the affiliate's balance by €0.50, and displays the new balance instead of the old balance while updating the value of the balance variable. The Withdraw button will result in storing the initial balance, performing the `settleAffiliateBalance` method within the `AdPlatform`, storing if it was successful or not, and storing the new balance, which will remain unchanged if credit is less than 5, or be set to 0 otherwise. After these actions the session is redirected to **withdraw.jsp** where these variables are then displayed accordingly.

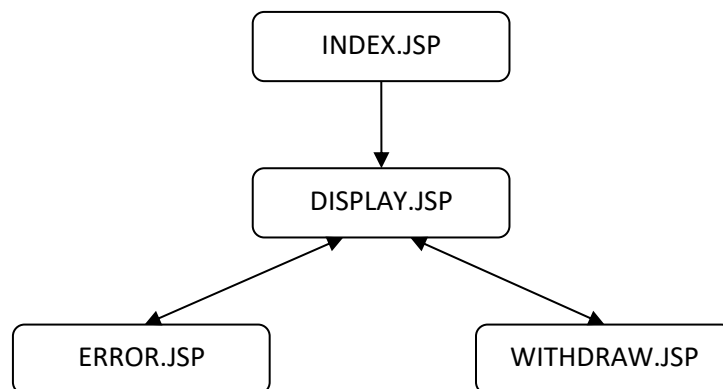
Should the entered details not match any registered affiliate the system redirects the page to an error page, **error.jsp**.

### Error Page – error.jsp

The error page is essentially the index login page with the difference that on entering an error message pops up using `alert("ID or password are incorrect")` as an error message. On closing the message that popped up the user may proceed to re-enter the details, which are then forwarded to the **display.jsp** for verification.

### Withdraw Page – withdraw.jsp

The withdraw page retrieves the old balance, message and new balance from the **display.jsp** page and displays them accordingly. The variables are once more tagged using `class="VARIABLE_NAME"` to make it possible to test for further on. A submit button is implemented in the withdraw page to redirect back to the account page, since the affiliate is still saved as a global variable the affiliate does not need to re-sign in after withdrawing.





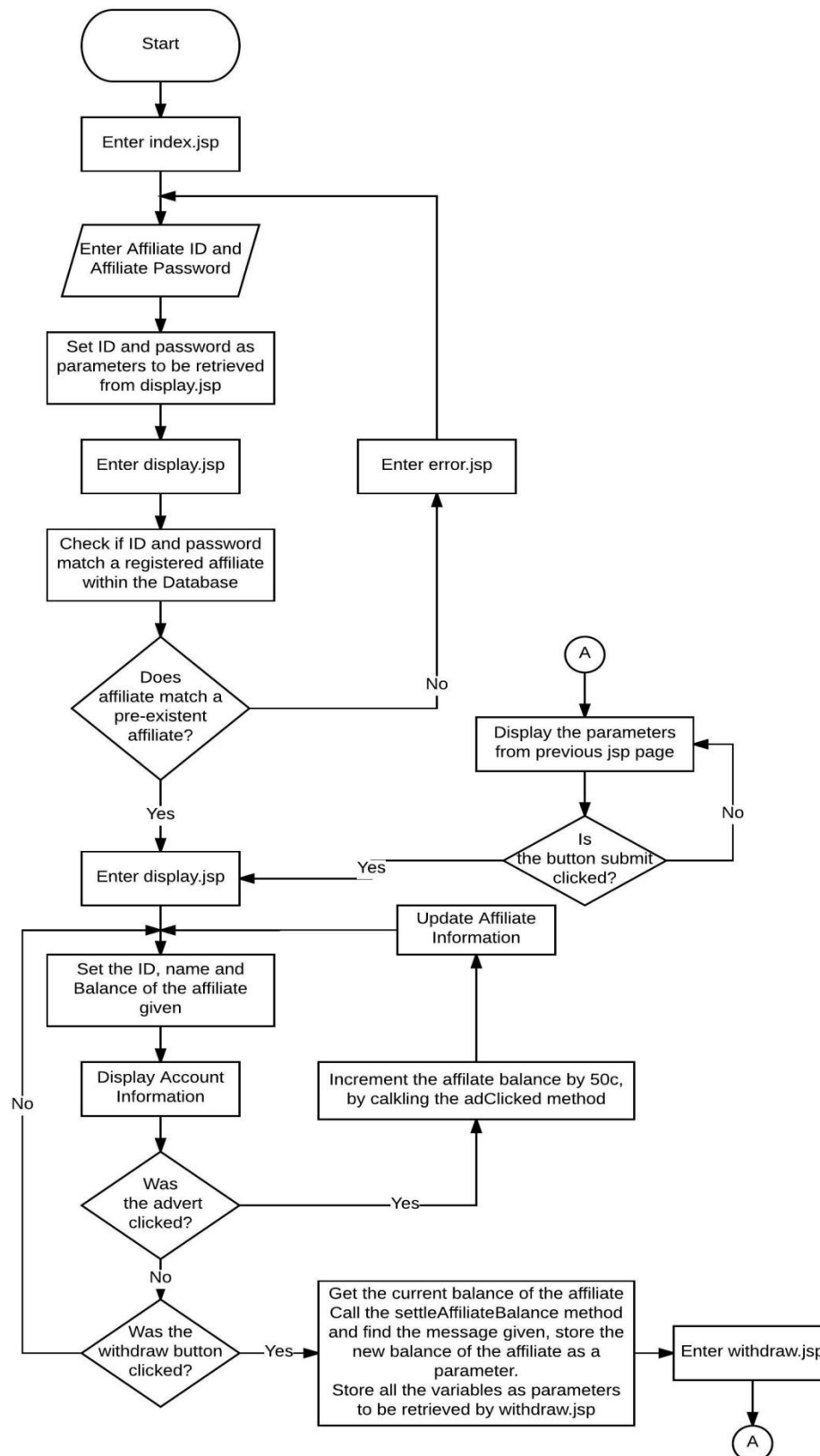


Figure 3 The flowchart representing the process flow of the web pages

## Cucumber Testing and Automated Web Testing

### AffiliateLogin.feature

In this class the 4 features to be tested for are declared. This feature is then accessed and run from RunCucumberTests.java and PageTest.java.

### RunCucumberTests.java

Launches all Cucumber tests within PageTest.java.

### PageTest.java

Four scenarios shall be executed within these Cucumber tests.

1. Successful Affiliate Login
2. Unsuccessful Affiliate Login
3. Account Admin Page Contents
4. Withdrawals

Before the tests began executing the selenium WebDriver was declared to be used.

```
Scenario: Successful Affiliate Login
  Given I am an affiliate trying to log in
  When I login using valid credentials
  Then I should be taken to my account admin page

Scenario: Unsuccessful Affiliate Login
  Given I am an affiliate trying to log in
  When I login using invalid credentials
  Then I should see an error message
  And I should remain on the login page
```

These two test shall be using the same **GIVEN**, and hence it was declared only once for both scenarios. The Given declares the affiliate and stores it in the AdPlatform.

### First Scenario

**WHEN** I login using valid credentials

The driver opens the localhost website, finds where the data needs to be entered and fills the correct relevant data. After filling the data the driver finds and clicks the submit button. The AssertEquals is performed on the correct id and password, for which a true result is expected.

**THEN** I should be taken to my account admin page

An AssertEquals is performed on the current Url brought by the driver being equal to "http://localhost:xxxx/display.jsp", the expected result is true. After the AssertEquals the driver closes the site and quits its process.

### Second Scenario

#### **WHEN** I login using invalid credentials

The driver opens the localhost website, finds where the data needs to be entered and fills it with incorrect data. After filling the data the driver finds and clicks the submit button. The AssertEquals is performed on the incorrect id and password, for which a false result is expected.

#### **THEN** I should see an error message

The driver checks if an alert message has popped up, if it has a flag is set to true. The AssertEquals is performed on this flag and it is expected to be true.

#### **AND** I should remain on the login page

An AssertEquals is once more performed upon the current URL brought by the driver being equal to "http://localhost:xxxx/error.jsp", the expected result is true. The **error.jsp** is essentially the login page after an error message. After the AssertEquals the driver closes the site and quits its process.

#### Scenario: Account Admin Page Contents

**Given** I am a logged in affiliate

**When** I visit my account admin page

**Then** I should see my balance

**And** I should see a button allowing me to withdraw my balance

#### Scenario: Withdrawals

**Given** I am a logged in affiliate

**And** my balance is <balance>

**When** I try to withdraw my balance

**Then** I should see a message indicating <message-type>

**And** my new balance will be <new-balance>

Once more the two tests shall be using the same **GIVEN**, and hence it was declared only once for both scenarios. The Given declares the affiliate and stores it in the AdPlatform and then proceeds to use the driver to send the login data with the entered affiliates id and password, and click on the submit button.

### Third Scenario

**WHEN** I visit my account admin page

The AssertEquals is performed on entering the correct id and password, the expected result is true.

**THEN** I should see my balance

Several AdClicks are performed and afterwards an AssertEquals is performed to confirm that the balances actually match, the expected result is true. The balance is gathered from the tagged variable using `findElement(By.className("balance"))`.

**AND** I should see a button allowing me to withdraw my balance

An AssertEquals is performed on the Withdraw button being present by measuring its size, the expected answer is true.

### Fourth Scenario

**AND** my balance is <balance>

Once more several AdClicks are performed and afterwards an AssertEquals is performed to confirm that the balances actually match, the expected result is true.

**WHEN** I try to withdraw my balance

The driver clicks on the withdraw button and an AssertEquals is executed on the current URL being equal to "http://localhost:xxxx/withdraw.jsp", expected result is true.

**THEN** I should see a message indicating <message-type>

An AssertEquals is performed on the webpage to check that an error message has been displayed and matches the expected one, expected result it true.

**AND** my new balance will be <new-balance>

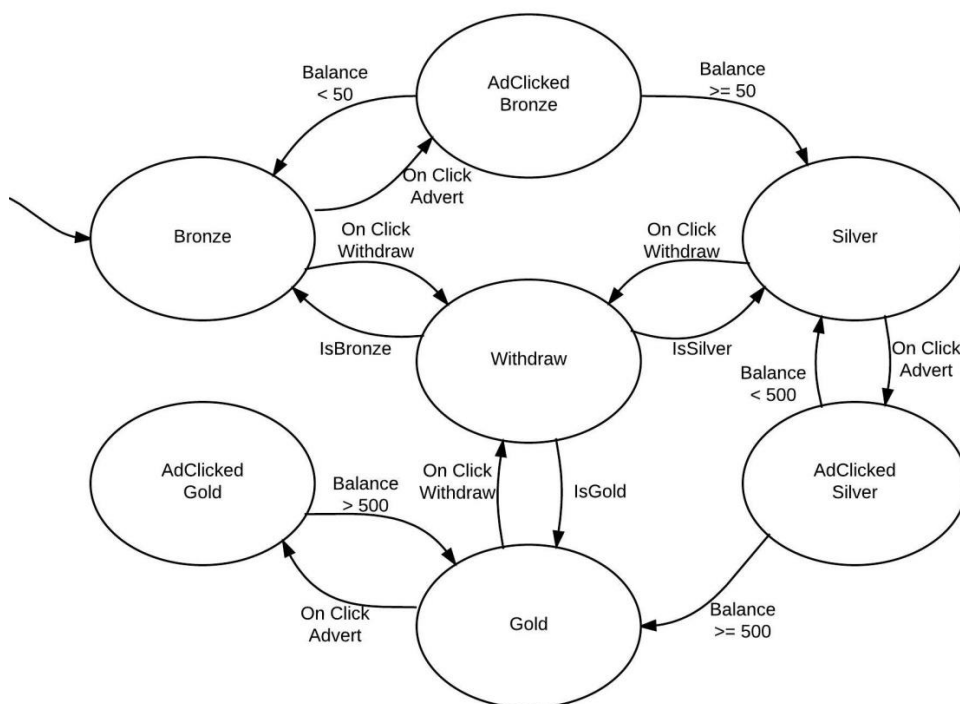
An AssertEquals is performed on the webpage to check that the new correct balance is displayed, this result is compared to the user's balance and it is expected to be true. The driver then proceeds to close the webpage and quits its process.

### TASK 3. MODEL BASED TESTING

For the web application to support simple ad delivery, a button called Ad Clicked was implemented. When this button is clicked, the `adClicked(int)` will be called. This method updates the balance and affiliate type.

#### State Model

The following model describes how the web aspect of an ad delivery from one AdProvider should function.



**Figure 4 Graphical Notation of the State Model**

The model states can be found in `AffiliateStates.java` while the model test cases can be run by running `AffiliateModel.java`.

The affiliate starts off with an initial balance of 0 having an affiliate-type of Bronze. This implies that the affiliate starts off in the bronze state in the model above.

Once AdClick happens, the current state is AdBronze. In the test case, it is asserted that the affiliate type is that of Bronze. The function `adClicked(int)` was applied on the Ad Platform. In this method, 50c is added to the balance and a check whether it should be upgraded to Silver takes place. If the affiliate's updated type is bronze, the current state is set to state Bronze while if the new type is Silver, the new state is Silver. It is asserted that the affiliate type matches the model's type.

Once the balance reaches 50 euro, the current state is set to Silver. After this transition, once an ad is clicked, the new state is set to AdSilver. An assertion is done to check whether the model's type and affiliate's type are both silver. The `adClicked(int)` function is called and the balance and affiliate type are updated. The new affiliate type is read and moves accordingly into the other states.

Once 500 euros are reached in the balance, the new affiliate type is set to Gold. An assertion is done to check whether the model's type and affiliate's type are the same. Once the Ad clicked button is pressed, the new state is set to adGold. An assertion is done to check whether the affiliate is of type gold. The function `adClicked(int)` is called and the current state is set to Gold.

When in states Bronze, Silver and Gold, one can press the button withdraw. The new state is set to 'Withdraw'. The balance is set to 0 if larger or equal to 5euro. This is done in the function `settleAffiliateBalance(Affiliate)`. However the affiliate type remains the same as it was before the transition. This implies that if the affiliate was of type silver, his balance would be set to 0 but he would still have the affiliate type of silver. Assertions were made to check that after the withdrawal, the affiliate's type remains the same.

A greedy algorithm was used to test this model. The reset probability was set to a low number so that resets would not frequently occur.

#### Assumptions:

- Once an affiliate reaches a higher affiliate type, it remains with the higher type. For example, if an affiliate is silver, he can only stay in silver or reach gold. He cannot downgrade to bronze.
- It is assumed that no sign out is needed for any affiliate.

#### Graph

The graph shown in figure 5 was obtained by driving the values obtained in the table below. The probability of the model was set to 0.001, meaning it had that chance of entering a different state from the one it is in. Apart from that, it had the chance of 1% of clicking withdraw, as to try and maximise the number of times the `adClicked` occurred and allowing an affiliate to reach the state of Gold.

<u>State Coverage Obtained</u>	<u>Test Cases Generated</u>
2	100
4	500
4	1000
5	10000
7	10000
7	50000
7	100000

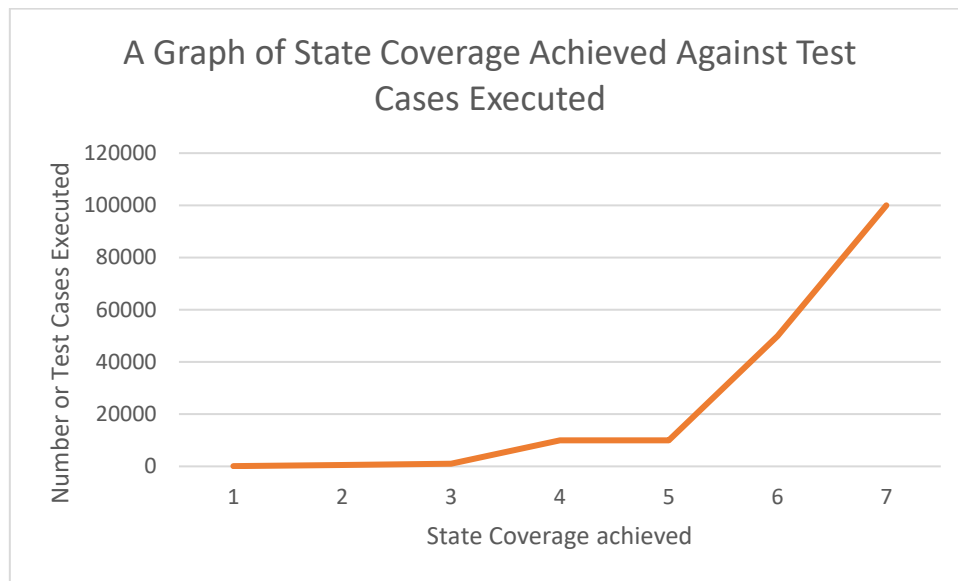


Figure 5 Illustrates a graph of the state coverage against the number of tests executed

### Further Steps

To reduce the amount of test cases required to reach 100% coverage, a random number was created in the withdraw test case. If the random number was equal to 5, the account was settled. This was done since the balance was settled too often when the Withdraw state was reached and the states such as Silver and Gold were not reached. This was causing us not to reach 100% state coverage. Once the random number generator was created, 100% state coverage was reached.

## TASK 4. PERFORMANCE TESTING

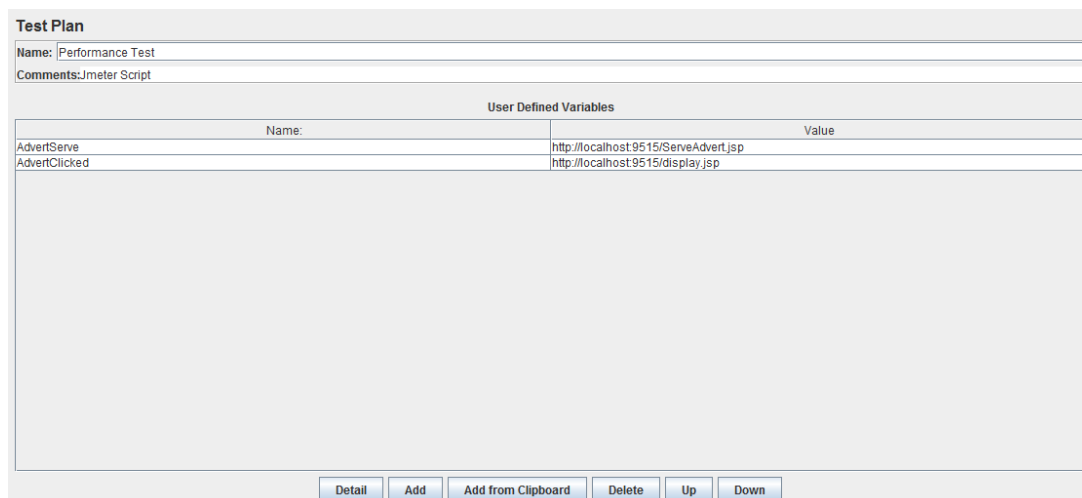
JMeter was used as a performance testing tool for testing out website. The downside of JMeter is that it cannot perform clicks, therefore all clicks had to be catered for by hardcoding them in the system. JMeter however can simulate traffic and thus it was used to perform 2000 serves over a ramp period of 2.5 second and another test of 5000 serves over a ramp period of 2.5 seconds. The 5000 serves does not guarantee success every time. Therefore for guaranteed performance it is assumed that on our machine we can handle a maximum 4000 serves. Our machine specifications are, 2.8 Ghz core i5 machine with 8GB RAM and Windows 10, the response time should not exceed 1 second.

The figure displays two screenshots of the JMeter Thread Group configuration interface. Both screenshots show the 'Thread Group' configuration for 'Localhost Visitors'. The top screenshot is configured for 2000 threads, a 2.5 second ramp-up period, and a loop count of 1. The bottom screenshot is configured for 5000 threads, a 2.5 second ramp-up period, and a loop count of 1. Both screenshots include a status bar at the bottom showing '0' errors, a warning icon, and the thread count '2000 / 2000' or '5000 / 5000'.

Figure 6 The setting of Thread Groups and ramp up

Our two test sites (<http://localhost:xxxx/display.jsp> and <http://localhost:xxxx/ServeAdvert.jsp>) were added in the user-defined variables as AdvertClicked and AdvertServe respectively. The AdvertClick and AdvertServe were both implemented at the same serve count instead of having Advert Click at 10% count of AdvertServe, this was done so both performance tests could be run at the same time using JMeter.





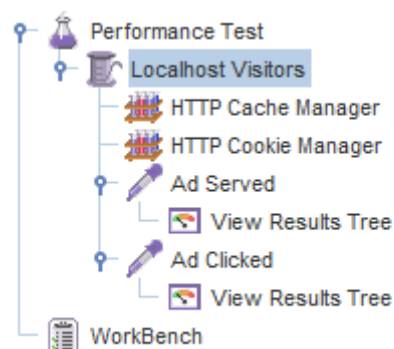
The screenshot shows the 'Test Plan' configuration window in JMeter. It includes fields for 'Name' (Performance Test) and 'Comments' (Jmeter Script). Below these is a table for 'User Defined Variables'.

Name:	Value
AdvertServe	http://localhost9515/ServeAdvert.jsp
AdvertClicked	http://localhost9515/display.jsp

At the bottom of the window are buttons: Detail, Add, Add from Clipboard, Delete, Up, and Down.

**Figure 7 Declaration of Sites to be tested for**

Ad clicking needed to be hard coded in display.jsp itself by performing the click every time the page was entered. The serve advert simply serves the advert from adPlatform and displays it on a page called ServeAdvert.jsp, which was created in order to be able to perform this performance test. A thread group was created where the thread stops if an error occurs. A cache manager and cookie manager were added to the group of threads.



**Figure 8 The Environment Test Setup**

An HTTP request was created for both cases, the requests simulates the traffic to the site. A listener that represents the results as a tree was also added to both cases as to be able to check the results from the traffic flow. The number of Affiliates that can be held in our collection was tested for in IntelliJ itself and was found to be the max number of a collection (max\_int).

## CONCLUSION

All tasks of the assignment are seen to be completed successfully to the best of our ability, meeting all the requirements specified. Well thorough planning has been put into all the parts of the assignment and these could be noted within the documentation itself. The details of the GitHub repository can be found once more hereunder.

- GitHub repository URL: <https://github.com/jonathanborg/CPS3230-Software-Testing>