**Course: Software Testing**

**Lab. Report #2 – Automated Requirements-Based API Unit Testing using JUnit**

|  |  |
| --- | --- |
| Group #: | G63 |
| Student Names: | Aaron Watters |
| Lewis Creelman |

# Unit testing plan

For the testing of the ‘JfreeChart’ software we as a group decided to use Pairwise testing as our main black box testing technique. There are a couple of reasons for this choice; firstly, using pairwise testing will allow us to reduce the number of test cases that we have to create as pairwise is not an exhaustive testing of the system but rather a testing of combination of parameters that are given to the functions we are testing.

Secondly, it gives a good in between of SECT testing and WECT testing in that we can provide an effective fault detection solution whilst also maintain a smaller more manageable size of test cases that we must monitor and record.

The process we used for deriving test cases is as follows:

1. We identify the use case scenarios for the system.
2. We identify test cases based on the use cases we have found.
3. For each of each test cases we identified we would identify the conditions that are required for the use case to execute successfully.
4. We would assign data values that we need to execute the test cases and add those values to the unit tests. We would then record the results of these Junit tests for the purposes of this lab document.

What follows is the organization of the Junit test suites for our unit tests:

for the two main classes of Range.java and DataUtilities.java we have 2 test classes: RangeTest.java and DataUtilitiesTest.java. Each of these classes will contain the test methods that have been created for the system use cases. For example, if we were testing the combine function in the Range.java class we have a test\_combine\_correct method which runs the combine method using correct data; we would then have a test\_combine\_incorrect method which runs the combine method to use incorrect data to test if the function fails when it is supposed to.

# Designing the unit test-cases using black-box test-case design techniques

Note: You should not include any test code in this section, but only the design of the test cases using the above methods, before coding them.

# Output of test suite execution

## Screenshot of test-suite execution in JUnit (showing the names of test methods)

See the lab document for an example

## List of failed test cases, and the possible defects based on that information

Text…

# How the team work/effort was divided and managed

## How the team work/effort of the lab was managed and divided

* You can say for example discuss which sections / parts of the lab was done by who…
* And also discuss the meetings that you had to plan and run the lab work
* Etc.

## Writing the lab report

Fill up the following table to specify who wrote what part of the lab document:

|  |  |
| --- | --- |
| **Lab-report section** | **Written by** |
| 1- Introduction | Student A |
| 2-.. |  |
| … |  |

## Lessons learned from your teamwork in this lab

Text

# Difficulties/ challenges encountered, overcoming them, and lessons learned

This section has the following sub-sections.

## Difficulties/ challenges encountered

Text…

## How did you overcome the above difficulties/ challenges?

Text…

## Lessons learned

Text…

# Comments/feedback on the lab and lab document itself

This section has the following sub-sections.

## Did you find the lab a useful learning experience? How it helped you learn the new testing topics

Text…

## Was the lab document easy to follow?

Text…

## About time budget? (Was there too much/too little time for this lab?)

Text…

## Please provide your comments on how to improve the lab work and lab document

Text…