**Course: Software Testing**

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

|  |  |
| --- | --- |
| Group #: |  |
| Student Names: |  |
|  |

**Table of Contents**

[1 Unit testing plan 2](#_Toc159269567)

[1.1 List of the methods under test 2](#_Toc159269568)

[1.2 The chosen black-box test technique(s) and why you have decided to choose them 2](#_Toc159269569)

[1.3 How the team plans to organize their JUnit test suites (based on Appendix C in lab doc) 2](#_Toc159269570)

[2 Designing the unit test-cases using black-box test-case design techniques 2](#_Toc159269571)

[2.1 SUT Class DataUtilities 2](#_Toc159269572)

[2.1.1 calculateColumnTotal(Values2D data, int column) 2](#_Toc159269573)

[2.1.2 calculateRowTotal(Values2D data, int row) 3](#_Toc159269574)

[2.1.3 createNumberArray(double[] data) 3](#_Toc159269575)

[2.1.4 createNumberArray2D(double[][] data) 3](#_Toc159269576)

[2.1.5 getCumulativePercentages(KeyedValues data) 3](#_Toc159269577)

[2.2 SUT Class Range 3](#_Toc159269578)

[2.2.1 Name of the method under test from Class Range 3](#_Toc159269579)

[2.2.2 Name of the method under test from Class Range 3](#_Toc159269580)

[2.2.3 Name of the method under test from Class Range 4](#_Toc159269581)

[2.2.4 Name of the method under test from Class Range 4](#_Toc159269582)

[2.2.5 Name of the method under test from Class Range 4](#_Toc159269583)

[3 Output of test suite execution: Include a screenshot of test suite execution in JUnit showing their Pass/Fail/Error status (showing the names of test methods) 4](#_Toc159269584)

[4 Based on the list of failed test cases (failures) in the previous section above, discuss the possible faults leading to those failures 5](#_Toc159269585)

[5 How the team work/effort was divided and managed 5](#_Toc159269586)

[5.1 How the team work/effort of the lab was managed and divided 5](#_Toc159269587)

[5.2 Writing the lab report 5](#_Toc159269588)

[5.3 Lessons learned from your teamwork in this lab 5](#_Toc159269589)

[6 Difficulties/ challenges encountered, overcoming them, and lessons learned 6](#_Toc159269590)

[6.1 Difficulties/ challenges encountered 6](#_Toc159269591)

[6.2 How did you overcome the above difficulties/ challenges? 6](#_Toc159269592)

[6.3 “Technical” Lessons learned 6](#_Toc159269593)

[7 Comments/feedback on the lab and lab document itself 6](#_Toc159269594)

[7.1 Did you find the lab a useful learning experience? How it helped you learn the new testing topics 6](#_Toc159269595)

[7.2 Was the lab document easy to follow? 6](#_Toc159269596)

[7.3 About time budget? (Was there too much/too little time for this lab?) 6](#_Toc159269597)

[7.4 Please provide your comments on how to improve the lab work and lab document 6](#_Toc159269598)

**General advice for writing high-quality lab reports:**

* Ensure following technical writing advice and resources on the internet. Do a Google search for:
  + <https://www.google.com/search?q=technical+writing+for+software+engineers>
  + And <https://www.google.com/search?q=technical+writing+for+students>
* Avoid common mistakes of technical writing ([google.com/search?q=technical+writing+mistakes](https://www.google.com/search?q=technical+writing+mistakes)), such as very long paragraphs, etc.

# URL of your project in GitHub.com

www.github.com/…

# Introduction

Write an introduction to your lab work. Your lab report is a “technical report”. The introduction section of a technical report should specify the context of the report. It should specify the purpose, objectives of the project, and an overview of the work done. There are various online resources on how to write the introduction section of a technical report: <https://www.google.com/search?q=technical+report+%22introduction%22>

# Unit testing plan

The unit testing plan should include the followings.

## List of the methods (units) under test

Class Range:

* …
* …

Class DataUtilities:

* …
* …

## Why equivalence-classes partitioning and boundary-value testing (BVA) are the good choices of for black-box test-case design for methods of the two classes under test

## How the team plans to organize their JUnit test suites (based on Appendix C in lab doc)

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

* You need to include in your report all the details and steps of the test-case design method technique(s). For example, we learned in the lectures that the equivalence classing technique has three steps:
  + 1-Identify the input domain (of method/function parameters):
  + 2-Equivalence classing of method input(s): shall be presented "visually", like done in the lectures
  + 3-Combining equivalence classes of the different inputs, using the multi-dimensional approach, i.e., Strong Equivalence-Class Testing (SECT)
* Note that the above details should be included for “each” method under test, in each of the sections below
* You should not include any test code in this section, but should only include the “design” of the test cases using your chosen test-case design method technique(s).

## SUT Class DataUtilities

### calculateColumnTotal(Values2D data, int column)

All the details and steps of your chosen test-case design method technique(s)

For example, if you choose to use equivalence classing, you should include these sub-section:

* 1-Identify the input domain (of method/function parameters):
* 2-Equivalence classing of method input(s): shall be presented "visually", like done in the lectures
* 3-Combining equivalence classes of the different inputs, using the multi-dimensional approach, i.e., Strong Equivalence-Class Testing (SECT)

### calculateRowTotal(Values2D data, int row)

all the details and steps of your chosen test-case design method technique(s)

### createNumberArray(double[] data)

all the details and steps of your chosen test-case design method technique(s)

### createNumberArray2D(double[][] data)

all the details and steps of your chosen test-case design method technique(s)

### getCumulativePercentages(KeyedValues data)

All the details and steps of your chosen test-case design method technique(s)

## SUT Class Range

### Name of the method under test from Class Range

all the details and steps of your chosen test-case design method technique(s)

### Name of the method under test from Class Range

all the details and steps of your chosen test-case design method technique(s)

### Name of the method under test from Class Range

all the details and steps of your chosen test-case design method technique(s)

### Name of the method under test from Class Range

all the details and steps of your chosen test-case design method technique(s)

### Name of the method under test from Class Range

all the details and steps of your chosen test-case design method technique(s)

# Output of test suite execution: Include a screenshot of test suite execution in JUnit showing their Pass/Fail/Error status (showing the names of test methods)

Include a screenshot of execution of the entire test suite in JUnit showing their Pass/Fail/Error status, such as:

A screenshot of a computer

Description automatically generated

**(Note: This is just an example. We are NOT providing the number of test cases for you.)**

# Based on the list of failed test cases (failures) in the previous section above, discuss the possible faults leading to those failures

Use the chain of software dependability threats: error, fault, failure, as learned in the lectures

Provide a sample of maximum 10 case (rows in the table below)

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Failed test case name** | **Discussion of possible error(s) leading to failure** | **Discussion of possible fault(s) leading to failure** |
| **1** |  |  |  |
| **2** | **…** |  |  |
| **…** |  |  |  |
| **10** |  |  |  |

# How the teamwork/effort was divided and managed

## How the teamwork/effort of the lab was managed and divided

* You can say for example discuss which parts of the lab-work (e.g., classes under test, etc.) 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

Only include lessons learned from **your teamwork in this section**. **“Technical”** lessons learned **shall be discussed in another section below.**

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

## Difficulties/ challenges encountered

Text…

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

Text…

## “Technical” Lessons learned

Only include **“technical”** lessons learned from **in this section**. Lessons learned **your teamwork shall be discussed in another section above.**

# JUnit test-code and it quality checklist

Provide comments on each of the following items:

## Checklist for setting up and using GitHub properly

|  |  |
| --- | --- |
| **Item** | **Put √ in each row if it has been done properly** |
| “Collaborator” access given to lab-work graders? The graders need to be able to leave comments in your test code |  |
| The GitHub repo ONLY includes the Java code files that the lab document has asked to be developed, and not any other files (even the project files starting with “.”, etc.) |  |
| Both students should have committed to the project files on GitHub |  |
| Meaningful GitHub account IDs have been chosen, including student name, to ensure that it is easy to identify who has committed what. Account IDs should not have cyphered strings such as xyz\_, ghost2020, etc. |  |
| We should see regular commits to the GitHub repo, during the duration of lab time period (two weeks) |  |

## Checklist for completeness of the test-code

|  |  |  |
| --- | --- | --- |
| **Item** | **Put √** | **How did you check it?** |
| Does the test-code clearly and fully match the test-case design, reported in your lab report? |  |  |
| There are no obvious requirements which have not been tested |  |  |

## Checklist for quality of test-code

|  |  |  |
| --- | --- | --- |
| **Item** | **Put √** | **How did you check it?** |
| High readability and understandability of test-code (Are the JUnit test methods easy to follow, through commenting or style?) |  |  |
| Have the naming conventions of test methods, variables, etc., proposed by the lab doc, been properly followed? |  |  |
| There are no ""error"" instances in JUnit suite's execution output view |  |  |
| Exception handling has been used properly in test-code |  |  |

# 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…