**Course: Software Testing**

**Lab. Report #3 – White-box testing and code coverage**

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

# URL of your project in GitHub.com

[www.github.com/…](http://www.github.com/…)

# General advice for writing high-quality lab reports:

* Ensure to read and learn from advice on technical writing 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.
* Obviously, all the pre-provided hints in this template SHOULD BE removed once students finished writing their report

# 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>

# Test plan for white-box unit testing

Explain how you plan to conduct white-box unit testing, which means designing test cases based on code coverage

Reminder:

* Test-case design means designing test cases “on paper”, using the techniques learned in lectures and in tabular form, BEFORE coding them in Java JUnit
* Test-case development means using the designed test-cases from the previous step to test methods in Java JUnit (NOTE: a test case shall be developed as one test method in JUnit)

# Description of how you have designed additional test cases (test methods) to improve code coverage for methods from the SUT

This section has the following parts.

## Summary table of test-suite size from in lab3 compared to lab2

Find more details in the lab doc

## Discussing details of design and development of additional test cases, for two randomly-chosen methods under test from each of the two classes under test

Follow the following procedure.

### Range class under test

For the Range class, even if you have developed tests for many of its methods, just choose randomly, as examples, two of its methods under test and fill out the following info (tables) for each of them. Do the same for the DataUtilities class.

We provide an example below. But that should be removed from your report before submission, and it cannot be used as one of your example cases …!

#### AN EXAMPLE as a hint to students:

|  |  |  |
| --- | --- | --- |
| BEFORE making the coverage improvement | Screenshot of code coverage of the method (BEFORE making the coverage improvement) - **Line coverage** |  |
| Screenshot of code coverage of the method (BEFORE making the coverage improvement) - **Branch coverage** |  |
| AFTER making the coverage improvement | Screenshot of code coverage of the method (AFTER making the coverage improvement) - **Line coverage** |  |
| Screenshot of code coverage of the method (AFTER making the coverage improvement) - **Branch coverage** |  |
| Discussions of how you designed the new test case(s) to increase code coverage for this method. Hint: Recall from lectures that, to increase code coverage, we analyze the code coverage of a method, and find out which parts (lines or branches) are not yet covered (tested), and determine the input(s) to the method which would cover those uncovered parts. | | Upon inspection of the coverage data (see above), we observed that, in the Range constructor, the true branch of the if statement is not covered. To cover that, we designed a new test called testConstructorInvalidInputs, which provides invalid inputs to the constructor (as shown in the code below), and when we run the new test suites, the uncovered if branch and the lines under it are now covered. |
| Screenshot of test-code of the new test cases, that you developed to increase coverage: | |  |

#### Method under test 1: method\_name

|  |  |
| --- | --- |
| Screenshot of code coverage of the method (BEFORE making the coverage improvement) -- IMPORTANT: MAKE FORMATTING CHANGES (ADD NEW COLUMNS / ROWS TO THIS TABLE) LIKE THE ABOVE DFIRST METHOD TO HAVE BOTH LINE AND DECISION COVERAGE SCREENSHOTS | A brown rectangle with a white border  Description automatically generated |
| Screenshot of code coverage of the method (AFTER making the coverage improvement) | A brown rectangle with a white border  Description automatically generated |
| Discussions of how you designed the new test case(s) to increase code coverage for this method. Hint: Recall from lectures that, to increase code coverage, we analyze the code coverage of a method, and find out which parts (lines or branches) are not yet covered (tested), and determine the input(s) to the method which would cover those uncovered parts.  Then, we design and develop new additional test cases (methods) to actually cover those uncovered parts. |  |
| Screenshot of test-code of the new test cases, developed to increase coverage: | A brown rectangle with a white border  Description automatically generated |

#### Method under test 2: method\_name

|  |  |
| --- | --- |
| Screenshot of code coverage of the method (BEFORE making the coverage improvement) -- IMPORTANT: MAKE FORMATTING CHANGES (ADD NEW COLUMNS / ROWS TO THIS TABLE) LIKE THE ABOVE DFIRST METHOD TO HAVE BOTH LINE AND DECISION COVERAGE SCREENSHOTS |  |
| Screenshot of code coverage of the method (AFTER making the coverage improvement) |  |
| Discussions of how you designed the new test case(s) to increase code coverage for this method. Hint: Recall from lectures that, to increase code coverage, we analyze the code coverage of a method, and find out which parts (lines or branches) are not yet covered (tested), and determine the input(s) to the method which would cover those uncovered parts.  Then, we design and develop new additional test cases (methods) to actually cover those uncovered parts. |  |
| Screenshot of test-code of the new test cases, developed to increase coverage: |  |

### DataUtilities class under test

#### Method under test 1: method\_name

|  |  |
| --- | --- |
| Screenshot of code coverage of the method (BEFORE making the coverage improvement) |  |
| Screenshot of code coverage of the method (AFTER making the coverage improvement) |  |
| Discussions of how you designed the new test case(s) to increase code coverage for this method. Hint: Recall from lectures that, to increase code coverage, we analyze the code coverage of a method, and find out which parts (lines or branches) are not yet covered (tested), and determine the input(s) to the method which would cover those uncovered parts.  Then, we design and develop new additional test cases (methods) to actually cover those uncovered parts. |  |
| Screenshot of test-code of the new test cases, developed to increase coverage: |  |

#### Method under test 2: method\_name

|  |  |
| --- | --- |
| Screenshot of code coverage of the method (BEFORE making the coverage improvement) |  |
| Screenshot of code coverage of the method (AFTER making the coverage improvement) |  |
| Discussions of how you designed the new test case(s) to increase code coverage for this method. Hint: Recall from lectures that, to increase code coverage, we analyze the code coverage of a method, and find out which parts (lines or branches) are not yet covered (tested), and determine the input(s) to the method which would cover those uncovered parts.  Then, we design and develop new additional test cases (methods) to actually cover those uncovered parts. |  |
| Screenshot of test-code of the new test cases, developed to increase coverage: |  |

# Showing that the coverage threshold is achieved for each class

For this section, four screenshots, one for Range, and one for DataUtilities) from the code coverage results showing the coverage “bars” with % values should be provided in the subsections below.

Each screenshot should be like:

A screenshot of a computer

Description automatically generated

Note: the levels should clearly be above the given thresholds.

## Range class

### Line-coverage screenshot:

BEFORE adding new test cases for increasing coverage:

…screenshot…

AFTER adding new test cases for increasing coverage:

…screenshot…

### Branch-coverage screenshot:

BEFORE adding new test cases for increasing coverage:

…screenshot…

AFTER adding new test cases for increasing coverage:

…screenshot…

## DataUtilities class

### Line-coverage screenshot:

BEFORE adding new test cases for increasing coverage:

…screenshot…

AFTER adding new test cases for increasing coverage:

…screenshot…

### Branch-coverage screenshot:

BEFORE adding new test cases for increasing coverage:

…screenshot…

AFTER adding new test cases for increasing coverage:

…screenshot…

# Output of test suite execution:

Include a screenshot of test suite execution in JUnit showing their Pass/Fail/Error status, and the top-bar numbers, such as:

A screenshot of a computer

Description automatically generated

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

# Comparison on the advantages and disadvantages of black-box testing versus white-box testing

**Using your examples and experiences learned in labs 2 and 3. Be as specific as possible.**

## Advantages of black-box testing (requirements-based test generation)

## Disadvantages of black-box testing (requirements-based test generation)

## Advantages of white-box testing (coverage-based test generation)

## Disadvantages of white-box testing (coverage-based test generation)

# Manual data-flow coverage calculations for the method Range.constrain(double)

## Step 1 -Identifying code “blocks” in a tabular form

Several examples of this exercise were done in the lectures.

|  |  |
| --- | --- |
| **Code block number** | **Lines of code in the code block** |
| ,,, |  |
|  |  |
|  |  |
|  |  |

## Step 2 -Designing the CFG using code block numbers

## Step 3 - Identifying the variables’ def / use from the CFG, in a tabular form

Use the tabular form from the lectures

## Step 4- Identifying the definition-clear-use paths, in a tabular form

Use the tabular form from the lectures

## Step 5- Calculating data-flow coverage ratios (percentage values) for a given test case or test suite, on all the definition-clear paths

# Manual mutation testing

## Mutation 1

### Mutation Operator used

### Provide the full code of the mutant method under test, and highlight the mutated line of code

### Results of running the latest test suite on the mutant, and discuss whether you need to design / add a new test case to distinguish the mutant

Include the following table, as provided shown in the lab document:

**Status of running each test case on Mutant1:**

|  |
| --- |
|  |

## Mutation 2

### Mutation Operator used

### Provide the full code of the mutant (mutated) method under test, and highlight the mutated line of code

### Results of running the latest test suite on the mutant, and discuss whether you need to design / add a new test case to distinguish the mutant

Include the following table, as provided shown in the lab document:

**Status of running each test case on Mutant2:**

|  |
| --- |
|  |

## Mutation 3

### Mutation Operator used

### Provide the full code of the mutant (mutated) method under test, and highlight the mutated line of code

### Results of running the latest test suite on the mutant, and discuss whether you need to design / add a new test case to distinguish the mutant

Include the following table, as provided shown in the lab document:

**Status of running each test case on Mutant3:**

|  |
| --- |
|  |

## Mutation 4

### Mutation Operator used

### Provide the full code of the mutant (mutated) method under test, and highlight the mutated line of code

### Results of running the latest test suite on the mutant, and discuss whether you need to design / add a new test case to distinguish the mutant

Include the following table, as provided shown in the lab document:

**Status of running each test case on Mutant4**

|  |
| --- |
|  |

## Mutation 5

### Mutation Operator used

### Provide the full code of the mutant (mutated) method under test, and highlight the mutated line of code

### Results of running the latest test suite on the mutant, and discuss whether you need to design / add a new test case to distinguish the mutant

Include the following table, as provided shown in the lab document:

**Status of running each test case on Mutant5**

|  |
| --- |
|  |

## Summary table of all mutants

Include the following table, as provided shown in the lab document:

**Summary table of all mutants**

|  |
| --- |
|  |

# Team work

## How the team work/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.**

# Technical difficulties/ challenges encountered, overcoming them, and lessons learned

## Technical 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.**

# Comments/feedback on the lab and lab document itself

This section has the following sub-sections.

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

Text…

## Was the lab document easy to follow?

Text…

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

Text…