**SENG 438 - Software Testing, Reliability, and Quality**

**Lab. Report #2 – Requirements-Based Test Generation**

|  |  |
| --- | --- |
| Group #: | 19 |
| Student Names: | Eric Renno |
| Ryan Sommerville |
| Quinn Ledingham |
| Kaumil Patel |

# Introduction

The purpose of this lab is to explore how to use Junit tests to thoroughly test the methods of a particular class in order to ensure that the class functions as it should. To do this, the Junit class allows you to create a test function and compares the correct result with the result that the function actually returns. Although the process is straightforward, accurately testing a class requires a thorough analysis of boundary cases and other necessary test cases for each method. Additionally,

# Detailed description of unit test strategy

// including the input partitions you have designed

The test was performed using the Eclipse IDE. First, java files were created for the test methods using a shortcut on Eclipse. This will create a method in the test classes for each method to be tested in the classes to be tested. Next, test cases were invented for each method so that boundary cases are covered. Once this is done, the test cases are coded. If necessary, a Mocker class is used as a substitute for inputs where we don’t have access to the actual class. After all the test cases are coded, the tests are run and the results recorded.

# Test cases developed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Method | Test Function | Expected Result/ Actual Result | Passes |
| DataUtilities | calculateColumnTotal( Values2D data, int column) |  |  |  |
|  | calculateRowTotal(Values2D data, int column) |  |  |  |
|  | clone(double [][] source) |  |  |  |
|  | createNumberArray(double[] data) |  |  |  |
|  | createNumberArray2D( double[][] data) |  |  |  |
|  | equal(double[][] a, double[][] b) |  |  |  |
|  | getCumulativePercentages( KeyedValues data) |  |  |  |
| Range | combine(Range range1, Range range2) | combineTestIntersect()  Input: range1: 0 to 10  range2: 5 to 15. | range.lower = 0  range.upper = 15 | Pass |
|  |  | combineTestNoOverlap()  Input: range1: 0 to 10  range2: 15 to 20 | range.lower = 0  range.upper = 20 | Pass |
|  |  | combineTestNull()  Input: range1 = NULL  range2: 0 to 10 | range.lower = 0  range.upper = 10 | Pass |
|  | getLowerBound() | getLowerBoundTest()  Input: Range = -10 to 10 | -10 | Pass |
|  | getUpperBound() | getUpperBoundTest()  Input: Range = -10 to 10 | 10 | Pass |
|  | constrain(double value) | constrainTestInsideRange()  Input: value = 3, Range = 2 to 7 | 3 | Pass |
|  |  | constrainTestOutsideRangeAbove()  Input: value = 8, Range = 2 to 7 | 7 | Pass |
|  |  | constrainTestOutsideRangeBelow()  Input: value = 1, Range = 2 to 7 | 2 | Pass |
|  |  | constrainTestOnLower()  Input: value = 2, Range = 2 to 7 | 2 | Pass |
|  |  | constrainTestOnUpper()  Input: value = 7, Range = 2 to 7 | 7 | Pass |

// write down the name of the test methods and classes. Organize the based on the source code method // they test. identify which tests cover which partitions you have explained in the test strategy section //above

# How the team work/effort was divided and managed

Ryan Sommerville: Formatted and put together most of the report.

Eric Renno:

Quinn Ledingham:

Kaumil Patel:

# Difficulties encountered, challenges overcome, and lessons learned

Text…

# Comments/feedback on the lab itself

Text…