**SSW 567 - Assignment 5**

**Group 5**

Miguel Camacho

Ed Chang

Harmony Sullivan

**Assignment Description**

The objective of this assignment is to unit testing your initial program (s) (or another if you really want to) using the techniques presented in the lecture. As part of that, you are to use a unit test tool, run a static analyzer, run a code coverage tool, and analyze your test cases vs. the equivalence classes and boundary conditions – and write/run appropriate test cases. You are to write up the results in an organized and hopefully interesting way that demonstrates what you did and what you learned and any interesting results. As always, start with a summary at the top, and follow with more detailed results.

Part 1

If you haven’t yet, Implement Junit or one of the C++ unit test tools (suggest trying parasoft tool for C++) for your triangle (or other) program and run all of your tests using it.

Keep track of the amount of time it takes to implement your test tool and run your tests.

Part 2

Find and run a static analyzer and code coverage tool against your program and current set of test cases. Keep track of time to implement and run tests. Fix bugs and write new tests as appropriate. Document results.

Part 3

Define equivalence classes and boundaries for triangle program.

Make sure you have

* External equivalence classes and boundaries for both Inputs and outputs
* Internal equivalence classes and boundaries

Map your existing tests to those classes. Analyze/Categorize the tests you have so far. (are you testing more than you need to? Less than you need to?)Add/delete tests as appropriate. Rerun your code coverage tool after you have added the new tests. Compare/Analyze results. Feel free to include a commentary on the tools.

Hope you enjoy this.

**Summary of Results**

The different testing methods all offered pros and cons. There is a pro of quickly running many tests in Junit, without having to manually run your program many times. Developers have control over input and can write many tests with small variations in the input. However, there are some issues that may make it necessary to write code in a more structured manner in order to use Junit properly. Our code that was in the main function was not getting tested.

We liked the code coverage tool because it was very visual and had excellent features such as the merge feature. It allowed an additional set of tests to be ran with additional test cases without having to rerun all tests. Code coverage seems important, but developers should also ensure they are listening to their customer/users and testing the scenarios that they envision for the program.

Equivalence classes and boundary conditions allowed the team see all possible scenarios for the triangle program and ensure that each scenario had a test case to be tested. Setting up the boundary conditions where pretty simple with:

* triangle sides greater than 0
* only numeric characters allowed
* checking to make sure the sum of two sides is greater than 1 side
* checking to see if the triangle is a right angle

**Part 1**

There is a limitation to testing each method separately with Junit. Some methods may depend on others when running the program as a whole. The “valid triangle” checks are all found in a method called isValid and not checked within each other method. When the main function is run, isValid is run first before any others are allowed. But when running methods on their own these checks are not done. This has to be taken into consideration when choosing input values. We may consider adding these checks to each method in the future, but the way the main function is currently written, it is not necessary. The good thing about Junit is developers have control over their inputs.

13 Junit methods were run to test all test cases. Small syntax errors were found and fixed and in the end there were 0 errors.

It took approximately 5 hours to get the Junit tests all set up and ran.

Test cases include those from week 2 as well as new ones from week 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Input** | **Expected Results** | **Junit method name** |
| HES-1 | Invalid side length | 0,1,1 | message stating side must be greater than 0 | testIsValid() |
| HES-2 | Correct scalene, right-angle triangle | 3,4,5 | message stating this is a scalene right-angle triangle | testScaleneAndRight() |
| HES-3 | Invalid values for input | a, 4, $ | message stating that invalid characters were entered | testIsValid2() |
| TEST-1 | isoceles triangle | 5,6,6 | isoceles triangle | testIso() |
| TEST-2 | scalene and right-angle | 3,4,5 | scalene and right-angle | testScaleneAndRight() |
| TEST-3 | Invalid Input | 5,6, A | Invalid Input | testIsValid3() |
| Test Case 1  Invalid Input:  Side Length is Zero | SIDE EQUALS 0 | Side A Length = 3  Side B Length = 4  Side C Length = 0 | Error Message:  Invalid Input | testIsValid4() |
| Test Case 2  Invalid Input:  Side Length is Negative | NEGATIVE VALUE | Side A Length = 3  Side B Length = 4  Side C Length = -5 | Error Message:  Invalid Input | testIsValid5() |
| Test Case 3  No Input:  Side Length not Entered | NO VALUE | Side A Length = 3  Side B Length = 4  For Side C Length prompt, hit Enter | Error Message:  Invalid Input | testIsValid6() |
| Test Case 4  Invalid Input:  Side Length has Large Value | LARGE VALUE | Side A Length = 3  Side B Length = 4  Side C Length = 2,147,483,647 | Error Message: Invalid Input | testIsValid7() |
| Wk4-1 | Determine if a triangle is a right triangle, within 1%. Testing requirement 4b. | 3.01, 4.01, 5.01 | [a^2 + b^2 must be greater than or equal to 99% of c^2 or less than or equal to 101% of c^2. So if a^2 + b^2 is greater than 0.99\*c^2 and less than 1.01\*c^2.]  **This is a right triangle, scalene.** | testScaleneAndRight2() |
| Wk4-2 | Determine if a triangle with all sides equal is equilateral. Testing requirement 5a. | 8,8,8 | It is not a right and an equilateral (missing print statement in requirements) | testEqAndRight() |
| Wk4-3 | Print Right Isosceles Triangle. Testing requirement 6a. | 1,1,1.4142 | Print “Right Isosceles Triangle” | testIsoAndRight() |
| Wk4-5 | Print Scalene Triangle. Testing Requirement 6e. | 7,8,9 | Print “Scalene Triangle” | testSca() |

An example Junit method from our tests:

@Test

public void testSca() {

// Test Case Wk4-5

Triangle2 tester = new Triangle2(7,8,9);

boolean expected = true;

boolean actual = tester.sca();

assertEquals("Triangle with sides 7,8,9 is a scalene triangle", expected, actual);

}

**Part 2**

**FindBug (static analysis)**

We used ‘FindBug’ as a static analysis tool. This tool did not take long to install in Eclipse, approximately 20 minutes. This tool finds bugs in the static code; therefore test cases were not ran

.

This found one bug in our code:

*[Bug: Test for floating point equality in Triangle2.isRight(double, double, double)*

*This operation compares two floating point values for equality. Because floating point calculations may involve rounding, calculated float and double values may not be accurate. For values that must be precise, such as monetary values, consider using a fixed-precision type such as BigDecimal. For values that need not be precise, consider comparing for equality within some range, for example:*

*if ( Math.abs(x - y) < .0000001 ). See the Java Language Specification, section 4.2.4.*

*Rank: Of Concern (15),*

*confidence: High*

*Pattern: FE\_FLOATING\_POINT\_EQUALITY*

*Type: FE,*

*Category: STYLE (Dodgy code)*

*At Triangle2.java:[line 75]*

*In method Triangle2.isRight(double, double, double)*

*Another occurrence at Triangle2.java:[line 76]*

*Another occurrence at Triangle2.java:[line 77]*

*File: C:/Users/harmony.sullivan/Documents/SSW-567/eclipse-code/Triangle2/src/Triangle2.java]*

The method (isRight) has actually been rewritten in week 4. I had just left the original method in the class, but don’t call it in the main method. Therefore, this bug was already taken care of by the new method isRight2, which covers numerical precision. I can go ahead and remove the isRight method. Therefore no bugs are found.

**EclEmma (code coverage)**

We used the ‘EclEmma’ code coverage tool. The install into Eclipse was also not very difficult, approximately 20 minutes. Running the test cases and recording the results took over 5 hours to perform.

The tool provides a percentage of code covered and missed instructions. Code is color-coded:

* Green for fully covered lines,
* Yellow for partly covered lines (some instructions or branches missed)
* Red for lines that have not been executed at all

[http://www.eclemma.org/userdoc/annotations.html]

It has a feature to “merge” code coverage runs. I used this feature to run all of the test cases listed below so I could find out which lines of code (in red) were never executed. The code coverage percentage of the first merged run is 91.5%. We removed some of the unnecessary code that was not needed (get statements that are never called, an empty constructor) and ran all the test cases again. We wrote test cases to ensure coverage of the rest of the code. Our final result is 97.8% code coverage. The only items left are

* The numerical precision statements in the right angle method, which are all in yellow (though we know they have been called). It may be looking for every possible value that falls in between the boundaries.
* The declaration of the class which contains the main method which is in red.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Input** | **Expected Results** | **Code Coverage percentage**  **Run 1** | **Code coverage percentage**  **Run 2 (removed get statements and empty constructor)** | **Code coverage percentage**  **Run 3**  **(new test cases added)** |
| HES-1 | Invalid side length | 0,1,1 | message stating side must be greater than 0 | 25.4% | 26.2% | 26.2% |
| HES-2 | Correct scalene, right-angle triangle | 3,4,5 | message stating this is a scalene right-angle triangle | 58.2% | 60.1% | 60.1% |
| HES-3 | Invalid values for input | a, 4, $ | message stating that invalid characters were entered | 9.5% | 9.8% | 9.8% |
| TEST-1 | isosceles triangle | 5,6,6 | isosceles triangle | 74.3% | 76.8% | 76.8% |
|  |  |  |  |  |  |  |
| TEST-2 | scalene and right-angle | 3,4,5 | scalene and right-angle | same as HES-2 above |  |  |
| TEST-3 | Invalid Input | 5,6, A | Invalid Input | 9% | 9.3% | 9.3% |
| Test Case 1  Invalid Input:  Side Length is Zero | SIDE EQUALS 0 | Side A Length = 3  Side B Length = 4  Side C Length = 0 | Error Message:  Invalid Input | 25.9% | 26.8% | 26.8% |
| Test Case 2  Invalid Input:  Side Length is Negative | NEGATIVE VALUE | Side A Length = 3  Side B Length = 4  Side C Length = -5 | Error Message:  Invalid Input | 25.9% | 26.8% | 26.8% |
| Test Case 3  No Input:  Side Length not Entered | NO VALUE | Side A Length = 3  Side B Length = 4  For Side C Length prompt, hit Enter | Error Message:  Invalid Input | 9% | 9.3% | 9.3% |
| Test Case 4  Invalid Input:  Side Length has Large Value | LARGE VALUE | Side A Length = 3  Side B Length = 4  Side C Length = 2,147,483,647 | Error Message: Invalid Input | 23% | 23.8% | 23.8% |
| Wk4-1 | Determine if a triangle is a right triangle, within 1%. Testing requirement 4b. | 3.01, 4.01, 5.01 | [a^2 + b^2 must be greater than or equal to 99% of c^2 or less than or equal to 101% of c^2. So if a^2 + b^2 is greater than 0.99\*c^2 and less than 1.01\*c^2.]  **This is a right triangle, scalene.** | 58.2% | 60.1% | 60.1% |
| Wk4-2 | Determine if a triangle with all sides equal is equilateral. Testing requirement 5a. | 8,8,8 | It is not a right and an equilateral (missing print statement in requirements) | 75.4% | 77.9% | 77.9% |
| Wk4-3 | Print Right Isosceles Triangle. Testing requirement 6a. | 1,1,1.4142 | Print “Right Isosceles Triangle” | 55.8% | 57.7% | 57.7% |
| Wk4-5 | Print Scalene Triangle. Testing Requirement 6e. | 7,8,9 | Print “Scalene Triangle” | 72.2% | 74.6% | 74.6% |
| **ADDITIONAL TEST CASES FOR WEEK 5 BELOW** | | | | | | |
| Wk5-1 | Invalid value for side 2 | 1, F, 3 | Print “Side 2 is not a number” |  |  | 9.3% |
| Wk5-2 | Invalid triangle: Side 2 is <= 0 | 5, -9, 4 | The sides entered do not form a valid triangle. |  |  | 23.2% |
| Wk5-3 | Invalid Triangle: Side 1 + Side 3 < Side 2 | 1, 10, 1 | The sides entered do not form a valid triangle. |  |  | 26.2% |
| Wk5-4 | Invalid Triangle: Side 2 + Side 3 < Side 1 | 10, 1, 1 | The sides entered do not form a valid triangle. |  |  | 28.7% |
| Wk5-5 | Isosceles with side 1 = side 3 | 6, 11, 6 | This is an isosceles triangle. |  |  | 69.9% |
| **MERGED (TOTAL) CODE COVERAGE (Run 1)** | | | | 91.5% |  |  |
| **MERGED (TOTAL) CODE COVERAGE (Run 2)** | | | |  | 94.5% |  |
| **MERGED (TOTAL) CODE COVERAGE (Run 3)** | | | |  |  | 97.8% |

**Part 3**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Test Cases boundary conditions | | | | | | | | | | | | |
| 1 | 2 | | 3 | | 4 | | 5 | | | 6 | 7 | 8 |
| Side 1 | <=0 | 1 | | 1 | | 1 | | 1 | | | 1 | 3 | 1 |
| Side 2 | 1 | <=0 | | 2 | | 1 | | 2 | | | 1 | 4 | 2 |
| Side 3 | 2 | 2 | | <=0 | | 1 | | 2 | | | 1.41 | 5 | 1.3 |
| Triangle Type | invalid | invalid | | invalid | | Equilateral | | Isosceles | | | Isosceles | Scalene | Scalene |
| Right Triangle | invalid | invalid | | invalid | | no | | no | | | yes | yes | no |
|  |  |  | |  | |  | |  | | |  |  |  |
|  |  |  | |  | |  | |  | | |  |  |  |
|  | Test Cases | | | | | | | | | | |  |  |
| 9 | | 10 | | 11 | | 12 | | 13 | 14 | |  |  |
| Side 1 | Non-Numeric Character | | 1 | | 1 | | 1 | | 10 | 1 | |  |  |
| Side 2 | 2 | | Non-Numeric Character | | 2 | | 2 | | 2 | 10 | |  |  |
| Side 3 | 3 | | 3 | | Non-Numeric Character | | 10 | | 1 | 2 | |  |  |
| Triangle Type | invalid | | invalid | | invalid | | invalid | | invalid | invalid | |  |  |
| Right Triangle | invalid | | invalid | | invalid | | invalid | | invalid | invalid | |  |  |

**Lessons Learned**

Quickly learning and using tools is a great way to get a feel for what they offer but there is so much more functionality offered in these tools if developers use them on a regular basis. Developers would be able to compare results across different programs and set benchmarks for new programs.

It is really interesting how testing code makes developers rethink how to write it altogether. Testing is definitely its own discipline and it complements the development of code.

**Honor Pledge**

We pledge on our honor that we have not given or received any unauthorized assistance on this assignment/examination. We further pledge that we have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source.