**Assignment #2 Report**

For this assignment I had to test various types of tests and make conclusions on which was the most effective, most efficient, etc. The tests I had to test were BVA Normal, BVA Robust, BVA Worst Case, and BVA Robust Worst Case. I had to test each on a triangle program where the user inputted the three sides of a triangle, and the program would state whether the triangle was equilateral, isosceles, scalene, or even an invalid triangle (whether the values didn’t work or were out of the program’s range). Here are my overall results on the tests.

*Time to Generate Test Cases*

The time to generate most of the test cases was not that long for me. If took about 30 seconds for the BVA Normal which had 13 cases (with 2 redundancies). The BVA Robust test, with 19 cases and 2 redundancies, took about a minute. Most of the time came from generating the test cases for the BVA Worst Case and BVA Robust Worst Case tests, which had 125 and 343 test cases respectively. However, I was able to cut the time down a lot thanks to careful planning.

When I filled in all the variables I did so in a pattern, such that one variable stayed constant for a time, another stayed constant for a little less of a time, and the third was always changing. Then I cycled through which variable was always changing, somewhat constant, and constant for even longer. I also made the changing a constant pattern. Therefore there was a pattern of expected results. Once I figured out that pattern, I was able to copy and paste the pattern of expected results. I then checked by actually running my test cases before purposely inputting any faults. Therefore, since my code was correct, the results on the “Computed” column would be as well. When I checked the “Pass/Fail” column for both tests, every single cell was a pass, meaning I did not make a mistake inputting my expected results.

However, those tests still took the longest to generate the test cases for. BVA Worst Case took about 10 minutes and BVA Robust Worst Case took about 15 minutes. So overall, it took me about 26 and a half minutes total. In total I also had six outcomes. There was “Equilateral”, “Isosceles”, and “Scalene,” if the values were valid for a triangle. If the values were invalid, then the result was “Invalid”. If one of the sides was out of the range, then the result was “Out of Range”. Anything else, such as if one of the sides wasn’t an integer, then “Invalid Values” displayed.

*Purposely Inputted Faults*

The first fault I inputted was switching the initial “N1< 1” to “N1<=1”, changing the boundaries of the set of valid integer inputs. My second fault was changing the “N1+N2 <= N3” to ““N1-N2 <= N3”. Thus changing what counts for an invalid triangle. My third fault was changing “N1<>N2” to “N1=N2”, thus changing what counts for a scalene triangle. For my fourth fault, I changed one of the printed results from “Isosceles” to “Scalene”, thus changing the computed result. My last fault was changing an "N1<>N2" to "N1=N2", thus changing what counted for an isosceles triangle.

*Effectiveness*

The percentage of each fault’s exposure to each test is recorded on the excel sheet I recorded my data on. The overall exposure percentage of all the faults, as well as each of the tests, is recorded as well. In terms of how often the faults were exposed, (out of a test’s cases, how many times did the fault change the outcome) fault 2 was exposed the most with 26.00%, and BVA Normal exposed the most faults with 59.62%. However, I would not say BVA normal was the most efficient. Both BVA Normal and BVA Robust never exposed my third fault, which is the reason why I automatically say neither of them are the most efficient, even though they had the two highest percentages of exposes. At least BVA Worst Case and BVA Robust Worst Case showed at least one exposure of each fault, which is all you need to know the bug exists. If the product had been released with fault 3 and the only test(s) had been BVA Normal and/or BVA Robust, the product would have been flawed. Overall, BVA Normal and BVA Robust had at least one exposure of each fault 3/5 times. BVA Worst Case and BVA Robust Worst Case had at least one exposure of each fault 5/5 times. Therefore, faults 1, 2, 4, and 5 had at least one exposure 100% of the time, but fault 3 had at least one exposure only 60% of the time. I would say BVA Worst Case is the most effective, since it exposed all faults but also had a higher frequency of exposures than BVA Robust Worst Case.

*Efficiency*

BVA Normal had 13 test cases, BVA Robust had 19, BVA Worst Case had 125, and BVA Robust Worst Case had 343. The only cases with redundancies were BVA Normal and BVA Robust, which had 2 redundancies each. I would say BVA Robust Worse Case is the most efficient because if did not have any redundancies, but also had the most test cases. Another reason why BVA Normal and BVA Robust are much less efficient is because of the gaps. For both those tests, there are many gaps of important test cases. One clear example is the fact that none of my expected results for neither the BVA Normal nor BVA Robust was a scalene triangle. This is why those tests did not detect an error with my third fault, because it could only affect a triangle that could be classified as a scalene.

BVA Normal and BVA Worst Case do not test for the out of the boundary values as well, which is an important gap to test for. What if the user entered a value out of the range and there was a bug? BVA Normal and BVA Worst Case would not have been able to see the fault.

Even though BVA Robust Worst Case has every combination of numbers from the boundaries and outside of them, it is still missing some test cases. What if one of the inputs is not an integer? It could be a double, or even a character or a string. (All of the other test cases miss the gap as well). It’s possible for a user to input their answer that way. Even though BVA Robust Worst Case is the most efficient, it still has some inevitable gaps too. And it can be assumed that it always will since it only tests boundaries values and no other types of values.