SE463 Assignment #1 Report

Going Through the Motions of the Boundary Value Analysis Software Testing Techniques

By: Elizabeth Dixon

Table of Contents

Project Overview

After being in this class for almost a month, it has become apparent very quickly that there's not just one way to test code and/or the software that coincides with it. Although somewhat impractical for large-scale and nuanced projects, one of the most widely documented techniques for software testing is Boundary Value Analysis (BVA). It functions fairly accurately in a pinch by targeting the areas in testing that most often have errors, but, especially depending on the "flavor" of BVA being used, this method quickly becomes a very long, tedious, and subject-to-human error process – especially when manually keeping track of test cases. With that being said, it becomes clear why software testers, out in the industry or otherwise, often create scripts to automate their testing procedures. Moreover, the purpose of this project is not only to truly understand why software testers carry out more optimal plans, but also to gain full understanding of the inner workings and patterns that come with BVA testing and its techniques. Before heading into the time-consuming and delicate task that is carrying out BVA testing, one must first understand the formulaic and organizational aspects that will be of assistance.

Test Planning & Organization

The first thing I did when thinking about how to go about setting up my testing procedure was to start at the simplest iteration of BVA testing and its formula so that I could easily wrap my head around the unique context provided. Said context was that I was given an executable program that calculated the area of a Rectangle, Square, Rhombus, and Triangle based on the inputs of side (etc.) values input by the user. The simplest iteration of BVA to grasp, in my opinion, is Normal BVA wherein the number of test cases is determined by the formula 4(n) +1, with *n* being the number of variables in the context – in this case it is two variables as per the side inputs displayed in the executable. However, I noticed that the Square area portion was the only shape that didn't take two inputs so I figured that the calculations of test cases for each type of shape would have to be separated. Therefore the Rectangle, Rhombus, and Triangle number of test cases could all be calculated as [4(2) + 1] * 3 to get 27 test cases for the Normal BVA testing section and the Square area would just be 4(1) + 1 = 5 test cases. The same logic would follow for the rest of the "flavors" of BVA wherein Robust BVA uses 6n + 1, Worst-Case uses 5ˆn, and Worst-Case Robust uses 7ˆn. After establishing this, I could then move on to conjuring up a test case log spreadsheet to organize the test case index, input(s), outputs, preconditions, and postconditions of each technique. Then, I got into a flow of repetition: I separated the sheets into one for each technique with no square plus others for only the square. Then, I would organize the sheet so that I could execute the program, easily source the values from it, and enter them into the spreadsheet. With each step, I knew that the number of test cases would eventually line up with the spreadsheet (and vice versa), but I still double-checked each value to make sure that I didn't make any off-by-one type mistakes that would exponentially get worse.

Boundary Value Analysis Discussion

One very important aspect of organizing BVA testing that I had to get used to was to make sure that I included the correct boundaries to each type of testing and that the combination of such led to the correct total number of test cases. For example, in Normal and Robust testing, the nominal value I chose in the bounds (5) was always present on the opposite input side of the boundary values due to the Single Fault Assumption nature of those two techniques. To clarify, the specific base boundary values I chose (i.e. minimum, minimum+, nominal, maximum-, maximum respectively) were 1, 2, 5, 9, and 10 because they were easy numbers to manage but also with the intention that 0 would be one of the values in Robust cases so that some of my testing could prove that not all side (etc.) values constitute a shape's area – even if calculated correctly by the program. Furthermore, the patterns of organization in each "flavor"of BVA became especially apparent when I had to organize them in my spreadsheet. When you simply plug the correct variable amount into the test case number formula, it's difficult to visualize what that actually entails but when you manually put it into a visual, it becomes much clearer – especially in terms of the overall scale of the project. This would be especially clear in a different context wherein the order of inputs *would* matter as opposed to this one where it doesn't matter if a Rectangle has a side Input #1/Input#2 value of 10 or an Input#1 Input #2 value of 5 - the Rectangle's area will always be 50. The same goes for the Triangle and Rhombus' areas as well - not only were they equal to each other's, it did not matter what value was given to the base or the height or diagonal #1 or diagonal #2.

Test Cases & Results

As of 9/25/24 all test cases have passed as all of the areas have been calculated correctly. Please refer to the folders of "exe [type of BVA]BVA Screenshots" for screenshots of the executable results and "SE463 A1 Test Case Log" spreadsheet for test case results and conditions on the repository called se463_fall2024/A1 on GitHub.

Discussion/Recap

As discussed in the Project Overview, going through the motions of each type of Boundary Value Analysis was definitely a learning experience. I not only learned *what* BVA is, but I also learned what types of BVA there are, how to calculate the number of unique test cases per type of BVA and how that can help you to organize your test case log, how the boundaries of each type of BVA works (i.e. Normal vs. Robust, Single Fault Assumption vs. Rejecting, etc.), and the many patterns (among many other things) that can be recognized by going through the motions of BVA. A huge takeaway from this assignment was that I feel like I now know this testing technique inside and out, but I also have to remember to take into consideration other types of contexts. In this shape area context wherein there are two variables and the order of sides, diagonals, etc. don't matter in terms of inputs and their effect on the expected output, I feel as though I got off easy in comparison to other contexts. Although there was the slight caveat of having to divide the test case number formulas to cater to the Square's inputs, that was an easy enough bump in the road to get over. However, even the classic examples like Date/Time and the Triangle problems where there are not only 3 variables – making the number of test cases to do manually exponentially more – but more nuance (such as leap years and invalid triangle conditions) would be a *lot* more difficult to carry out – and even more so because I was able to trust the executable to be correct within my chosen boundaries.

Difficulties aside, this was an ultimately fruitful experience. Now that I have the flow of logging, organizing, and carrying out one of the techniques of software testing down, I feel confident that I'll be able to apply it to others. However, there's still a lot that I need to learn and consider when testing software in the future. For example, I would ask myself, *Were there better values to test to see if there were gaps in the program provided to me?*; *Where are there going to be gaps or clusters within the placements of my tests? How can I close them?* and so on. Of course, there are virtually never-ending possibilities for optimization - especially when it comes to something like software testing.