Project 2: Summary and Reflections Report Brandon Glasgow

Summary

For project one I used unit testing and integration testing mainly. I first wrote the unit tests to ensure that the parts of the software worked as planned, then I ran it all together to ensure that the software parts all functioned properly together with one another. Some techniques I chose to use in my project included boundary-value analysis, decision tables, and equivalence partitioning. Using techniques like these ensured that my testing proved effective and that the software that I had created met the requirements set forth. My JUnit tests were written in a modular fashion so that the tests were easier to debug and maintain. Additionally, I used several assertions in verifying the test results, which ensured that the software was functioning properly. Lastly, I kept my code concise which helped in finding errors when they arose.

Reflection

Some testing techniques that I employed in project 1 as stated above were unit testing, boundary value analysis, equivalence partitioning, and decision tables. Unit testing, which is referred to as a white box testing technique, is a technique that involves testing individual sections of code. Unit testing is the most basic level of testing, but is at the same time one of the most essential levels of testing, being that it is responsible for ensuring that your code is correct and functioning as it should be. Boundary value analysis is a black box testing technique which includes testing the boundaries of input and output values. This technique helps to ensure that the code can correctly hand extreme values. Another black-box testing technique that I employed, equivalence partitioning, is a testing technique that includes testing the interactions between different units of code. Equivalence partitioning is typically done after unit testing, being that the units of code wont work together as intended if they themselves don’t work in the first place. I also used the black box testing technique of system testing, in which I tested the system as a whole to ensure it had the desired functionality. As discussed in project one, I did not use acceptance testing given the fact that in our project, we did not have an actual client reviewing our code and providing real time feedback. In terms of practical uses of the aforementioned techniques, the practicality of the techniques lies within the situations and the overall type of project being created. For example, equivalence partitioning is a good choice of technique when an input space is complex or large, decision tables are good when there are a large number of input output combinations, and boundary value analysis is a good choice in projects where the values of the input and output are critical.

I adopted the mindset that I need to be cautious yet precise. My knowledge of the complexity and relationships between the different parts of my code led me to focus on coding efficiently and correctly, testing all parts and possibilities within the software, and limiting bias when reviewing the code. In the software development world, bias can be concerning, especially when a developer is reviewing their own code. This bias usually occurs because developers are more likely to overlook errors and bugs in their own code. It is important to limit this bias using practices such as having others look over your work and training yourself to have a keen eye. Lastly, as in all software development, is it is important to focus on the quality of your work. The commitment to quality is shown through one’s lack of “cutting-corners” when it pertains to writing and testing your code. Commitment to quality leads to more accurate, efficient, reliable, and secure software for your users