Mary Sanders

Dr. Karl Lewis

CS 320

Southern New Hampshire University

December 3, 2023

While writing my unit test code, I made sure that my tests were aligned to the software requirements by going through each one systematically. As I carefully read the requirements, I took notes and even wrote some preliminary pseudocode to begin the base of my coding. I ensured that each of the requirements were checked off the list and that each one had a corresponding block of code in the test classes. Once the object and service classes were written, I went through each variable and wrote a corresponding JUnit test for each possible parameter input. At first, I did not really understand how to check for JUnit coverage, but by the time I got to the assignment in module five, I had figured out how to read the outcome and increase the percentage. I was able to ensure test coverage of at least 80% for all the individual classes that I turned in for Project One.

One way that I ensured my code was efficient by using operators that could group variable requirements together. For example, in the following code I saved space and processing time by combining both value requirements for taskID into the same Illegal Argument Exception.

A computer code with text

Description automatically generated

The techniques that I used to perform my software tests are examples of component, or unit, testing. This is where blocks of code are tested to ensure they work according to the requirements. A block of code that is tested during a unit test could be as small as an individual method within a single class (Vogel, 2021). My tests are an example of white-box testing, as my intent was to test functionality based on input (GeeksforGeeks, 2023). Unit testing is done early in the SDLC, when mistakes are easier and cheaper to fix.

There are three other types of testing techniques that I did not use for these assignments, namely integration, systems, and acceptance testing. During integration testing, code that has passed the appropriate unit tests is now tested to ensure that various components work correctly as they are integrated. In turn, integrated components that have passed integration testing then undergo systems testing. This testing technique is designed to test the functionality of the entire system of a piece of software or application. Finally, acceptance testing is done at the end of the project by the final client or intended user, and the goal is to determine if the software is ready for release (Pearson, 2015).

I have only been writing code for about a year, and only for my assigned courses, so I am inexperienced. My mindset has always been to complete my coding assignments to match the requirements as closely and as simply as possible. Whenever I try to make things more complex, I run into problems that I am unable to solve. This inevitably leads to frustration and wasted time on my part. I am also not confident enough in my skills to really feel that I am biased towards the code that I wrote myself. In fact, it may be the reverse, that I don’t think my code is good enough until I get feedback. Generally, anyone that creates something can have the potential to turn a blind eye to any flaws in the creation. This is why it is a good idea to have someone else to test your code after you have run your own tests.

Discipline as a software engineering professional is important for several reasons. If the development of software is rushed, the quality of what is produced is often diminished. This can result in errors and bugs that are missed, which in turn can have a negative impact on customer sentiment and an increased reliance on customer service. In addition, software that is rushed may be released with security vulnerabilities that may be exploited (Mitton, 2023). The speed at which an application or piece of software is released must be weighed against the risk of any potential negative outcomes. I plan to avoid technical debt by being mindful of it and managing it carefully like any other expense. I will be aware of the importance of maintenance testing, and I will set high quality standards with the intent to meet them with every project.

References

GeeksforGeeks. (2023, November 16). *Unit testing - software testing*. GeeksforGeeks. https://www.geeksforgeeks.org/unit-testing-software-testing/

Mitton, L. (2023, November 7). *Technical Debt & How To Manage It*. Splunk-Blogs. https://www.splunk.com/en\_us/blog/learn/technical-debt.html

Pearson, L. (2015, September 11). The Four Levels of Software Testing | Segue Technologies. Segue Technologies. <https://www.seguetech.com/the-four-levels-of-software-testing/>

Vogel, L. (2021, July 31). Junit 5 tutorial – Learn how to write unit tests. https://www.vogella.com/tutorials/JUnit/article.html