Nick Thomas

In my opinion, my approach aligned closely with the software requirements. To achieve this, I tried to implement the requirements exactly as they were given to me. This claim is supported by viewing my code and the requirements side by side. You can clearly see that my work follows the requirements closely. By following the requirements closely, I was able to ensure that my approach to unit testing was closely aligned with the given software requirements.

I believe the quality of my JUnit tests for the Grand Strand Systems project to be very high. Based on coverage percentage, my JUnit tests were optimal in their effectiveness while not providing overkill. Since the test results were between 70% and 80%, I was able to ensure that I was delivering quality unit testing that did not provide overkill or unnecessary coverage.

To ensure that my code was technically sound, I made sure to iterate on my test cases until I was satisfied with my effective coverage percentage. Specifically, when it came to the various test classes, I tried to ensure that these classes were optimal in whatever it was they were testing.

To ensure that my code was efficient, I made sure that I was iterating on my design with express intent of improving my coverage percentage until it was optimal. Simultaneously, I tried to keep the industry standard best practices in mind when it came to how I went about achieving my desired results in my code.

There are many different software testing techniques that can be employed throughout the software development lifecycle. However, for each of the milestones, I only employed a couple different techniques. The first technique I employed was unit testing. Unit testing, in its simplest form, is code testing code. For each milestone I developed use cases that tested various aspects of the code that I developed. I also employed the use of Manual or Static testing. When I was first developing my code, I used this technique over unit testing; however, once I had more of my code developed, I employed the usage of unit testing. Both testing strategies played a key role in the successful completion of each milestone.

Additionally, there are other types of software testing techniques that I did not use for the milestones. These include techniques such as automated testing, functional and non-functional testing, acceptance testing, regression testing, exploratory testing, and user acceptance testing; There are even quite a few more that I didn’t include.

There are many practical uses and implications for the techniques discussed as it pertains to different software development projects and situations. During the software development life cycle various forms of these testing methods can be deployed depending on the situation. Even in more primitive projects, such as some of the ones required earlier in this degree, a simple static test can be employed to ensure that code is performing more optimally than if no testing had been done at all. In conclusion, there are a wide variety of testing techniques that can be applied to nearly every situation.

The mindset that I chose to adopt when working on this project can be described as deliberate. I employed caution to a degree because, to a certain point, it is necessary. However, I made it my intent to strike a balance in my approach. This is not to say that imprudence was employed; but rather, I was trying to avoid overkill. In this balance of caution, we can truly begin to appreciate the complexity and interrelationships of the code being tested. The code base developed for this assignment had a decent amount of complexity using a variety of unique classes, for example the java.util.Date class.

I tried to limit bias as much as possible in reviewing my code. I realized in the earlier milestones that by being more objective in my approach to testing I could deliver quality unit tests that were both technically sound and efficient. I can easily see how as a developer who is testing their own code that there would be bias. However, I hold the belief that the more objective and deliberate a developer is in their testing the better the end product will be.

Furthermore, being disciplined in the commitment to quality is one of the most important things to consider as a software engineering professional. There are numerous cases that exist of engineering professionals who cut corners in writing and testing code. In certain instances, such as Y2K, it cost billions to repair. I plan to avoid technical debt as a practitioner by being objective, disciplined, and deliberate in my approach to delivering quality. In holding such a high standard, I can ensure that I am avoiding technical dept as a software engineering professional.