When implementing unit testing for the coding project. It was nearly entirely dependent on the requirements that the customer expressed. For example, when implementing the JUnit tests, I would write a separate testing class in which I would include multiple tests from the Junit module. I was even able to use Junit’s @repeatedTests to test the same functions multiple times. I have currently written 3 different services that were coverage tested through Junit. In module three I was still learning the ins and outs of Junit so my code came out a little underwhelming. I was however able to take the insightful comments and included video that my professor provided to gain a deeper understanding of assertions and coverage. With my new found knowledge, I was able to achieve over an eighty percent coverage rate with absolutely zero failures in the Junit testing. Eighty percent is the threshold that we use to ensure proper coverage. Overall, I still believe that my code could be improved drastically. For instance, I am not sure exactly how to fix the issues that the coverage test highlights. I tried a couple of methods to match what had already pass but could not manage to achieve any higher of a percentage. I included many exception handling techniques to ensure that my code would run properly. My favorite method of ensure my code was technically sound is the repeated tests that I implemented. Drawing inspiration from my professor’s example code, I included @RepeatedTest to test each function three times before moving on. This ensures that the code is able to run smoothly every single time it is ran. This can be seen in line 21 of the TaskServiceTest class.

20 @Tag("AddTask")

21 @RepeatedTest(3)

22 **public** **void** testAddTask() {

To ensure that my code was efficient, I added functionality in the add task function which ensures that there are no duplicate objects created inside of the program. This will ensure that there is no wasted memory when creating the program. It will also allow the code to run faster as parsing through a list of tasks will be completed faster on average due to the smaller number of objects having to be checked.

During the milestones for the previous modules, I was required to implement different testing techniques to ensure properly functioning code. These techniques also help to reduce the number of security risks that my code may introduce. There are two main types of testing methods that I used during my milestones.

The first of the two is something called static testing. This is a type of testing that allows me to debug the code without actually having to run the code. This is useful because it can be done quickly and easily as I write my code. That doesn't mean that I won’t check my entire codebase for potential bugs after I am done. The process I use for my code I can describe as plan, code, check, code and check again. This allows me to ensure that I have thoroughly thought out each step of my code.

The second type of testing that I implemented is dynamic testing. This is a testing technique that is performed when the code is executed. I utilized Junit testing to perform unit tests on my functions. Junit was extremely helpful because it would point out each error that I made so that I could go back and alter the code. Once I was able to clear all of my functions, the code passed all of the tests that Junit ran.

In eclipse, I was able to use a “Coverage As” feature that allowed me to see exactly what lines of code were executed during the Junit tests. Using the coverage as feature, I could ensure that all of my functions were being tested. Although it took me a little bit of time to figure out, the tester classes are not really relevant in the case of coverage. This is because they are only there to test the functions in our service classes.

There is one method of testing that does not really make sense to employ at this time. That testing method is performance testing. Performance testing allows us to test how well our code runs under high amounts of stress. This may include, stress testing where the developers incrementally increase the amount of workload that the code has to handle until failure. This can allow developers to tweak the code to maximize performance.

While working on this extensive project throughout the course, I would have to admit that I took very little caution during the process. This is mostly due to my style of development. While I am aware what I like to call “brute force” methodology of development may cause trouble in some projects, I also believe that it can be a very powerful technique. The “brute force” method I am referring to is slowly implementing small portions of code, testing it as if it were complete, then altering it if necessary. This was not out of a lack of appreciation of the complexity and interrelationships of the code, but quite the opposite. Because I knew that changes in one class may alter the coverage and test results of another. I would test every method that I introduced before moving forward. I found myself swapping back and forth between the object class and the object-test class respectively simply to find the coverage that I was missing.

Another variable that may cause problems when testing code is bias. Bias in development can cause developers to improperly perform tests inside of their code. For example, skipping testing of a certain method because it is “so simple” can leave errors unchecked. That is why I relied on the coverage tool to ensure that I did not skip any methods that required testing. Much to the same point, developers will often let the quality of their code suffer for marginal amounts of time gained in the process. While this may seem fine at first, developers will accumulate what is called technical debt. This is where over time, the accumulative lack of quality in code can cause necessary refactoring and undoubtedly wasting more time than was gained from the start. To avoid spending more time than was needed on this project, I ensured that the quality of my code met the standards of my fullest potential. I would never claim to be among the best of Java developers, but what I lack in talent, I make up for in willingness to learn. Along the development journey of this project, I learned many new skills. To keep with the theme of quality, I would back track on the code that I had previously written to implement my newly found knowledge. This way, I could submit a project that I was proud to develop. I fully intend to carry the same mindset into my professional career and look forward to the places it will take me.